GTM4.1-V1.30 | 2023-12-07

GTM IP Specification

BOSCH

Table of Contents

1 Disclaimer

2 Introduction

2.1 Overview

This document is the specification for the Generic Timer Module (GTM). It contains a module framework with submodules of different functionality. These submodules are combined in a configurable manner to form a complex timer module that serves different application domains and different classes within one application domain. Because of this scalability and configurability the timer is called generic.

The scalability and configurability is achieved with an architecture philosophy where dedicated hardware submodules are located around a central routing unit (called Advanced Routing Unit (ARU)). The ARU connects the submodules in a flexible manner. The connectivity is software programmable and is configured during runtime.

Nevertheless, the GTM-IP is designed to unload the CPU or a peripheral core from a high interrupt load. Most of the tasks inside the GTM-IP run (once setup by an external CPU) independently and in parallel to the CPU software. There may be special situations, where the CPU has to take action but the goal of the GTM design was to reduce these situations to a minimum.

The hardware submodules have dedicated functionalities, for example, there are timer input modules where incoming signals are captured and characterized together with a notion of time. By combination of several submodules through the ARU, complex functions are established. For example, the signals characterized at an input module are routed to a signal processing unit where an intermediate value about the incoming signal frequency is calculated.

The submodules are categorized into the following groups:

Interface components: The slave communication from CPU to GTM is covered by the submodules AXIS and GTM_AEI. The master communication from GTM to CPU is established by the submodule AXIM.

Infrastructure components: These submodules are needed to define common functionality in the GTM and ensure that data storage and exchange between other submodules is possible. They are in use to implement complex functions which have to pass data over multiple submodules to ensure the user application. These components are present in all GTM devices. However, the number of these components may vary from device to device.

IO Modules: They have connections with input or output signals to communicate from and to the GTM. They are built based on a regular architecture to fulfill typical timer functions, for example, usable as PWM generation units, input measurement units.

Programming components: These submodules are programmed by assembler or high level language (C) and control the overall submodules in a GTM device.

Dedicated functionality: These submodules are those fulfilling a dedicated functionality for a certain application domain, for example, the DPLL serves engine management applications.

Interrupt management: The module ICM is responsible for interrupt services.

Safety features: These submodules are able to support the implementation of safety functions to fulfill a defined safety level.

Therefore, each GTM-IP is built with submodules coming from these groups. This allows to tailor GTM devices to specific application domains. A scalability over multiple GTM-IP devices is possible by varying the number of components of those submodules, this allows to define and integrate different GTM devices into microcontroller families / SOCs.

2.2 Document Structure

The structure of this document is motivated by the aforementioned submodule groups. Chapter *3.1 "Overview"* describes the dedicated GTM-IP architecture. It gives an overview of the structuring of the implemented submodules.

The master interface to the CPU is described in *4 "AXI Master"* . The following chapters *5 "Advanced Routing Unit (ARU)"* up to *12 "Time Base Unit (TBU)"* deal with the so called infrastructure components for routing, clock management and common time base functions. Chapters *13 "Timer Input Module (TIM)"* to *16 "Dead Time Module (DTM)"* and *27 "Timer Input Output (TIO)"* describe the signal input and output modules while the following chapter *17 "Multi Channel Sequencer (MCS)"* explains the programmable core with its memory configuration *19 "Memory Configuration (MCFG)"* . The next chapters *20 "TIM0 Input Mapping Module (MAP)"* to *23 "Sensor Pattern Evaluation (SPE)"* provide detailed descriptions of application specific modules like the MAP, DPLL and SPE. Chapter *24 "Interrupt Concentrator Module (ICM)"* describes a module that bundles several interrupts coming from the other submodules and connect them to the outside world. The last chapters *25 "Output Compare Unit (CMP)"* to *26 "Monitor Unit (MON)"* provide information on the safety related module.

Table 1 Sub-module groups

Chapter	Sub-module	Group
3 "GTM Architecture"	Slave configuration interfaces GTM_AEI / AXI-S	Interface components
4 "AXI Master"	Advanced eXtensible Interface Master Bus (A-XIM)	Interface components
5 "Advanced Routing Unit (ARU)"	Advanced Routing Unit (ARU)	Infrastructure components
6 "Broadcast Module (BRC)"	Broadcast Module (BRC)	Infrastructure components
7 "First In First Out Module (FIFO)"	First In First Out Module (FIFO)	Infrastructure components
8 "AEI to FIFO Data Interface (AFD)"	AEI-to-FIFO Data Interface (AFD)	Infrastructure components

Chapter	Sub-module	Group
9 "FIFO to ARU Unit (F2A)"	FIFO-to-ARU Interface (F2A)	Infrastructure components
10 "Clock Management Unit (CMU)"	Clock Management Unit (CMU)	Infrastructure components
11 "Cluster Configuration Module (CCM)"	Cluster Configuration Module (CCM)	Infrastructure components
12 "Time Base Unit (TBU)"	Time Base Unit (TBU)	Infrastructure components
13 "Timer Input Module (TIM)"	Timer Input Module (TIM)	IO Modules
14 "Timer Output Module (TOM)"	Timer Output Module (TOM)	IO Modules
15 "ARU-connected Timer Output Module (ATOM)"	ARU-connected Timer Output Module (ATOM)	IO Modules
13 "Timer Input Module (TIM)"	Dead Time Module (DTM)	IO Modules
27 "Timer Input Output (TIO)"	Timer Input Output Module (TIO)	IO Modules
17 "Multi Channel Sequencer (MCS)"	Multi Channel Sequencer (MCS)	Programming components
18 "ADC Interface (ADCIF)"	ADC interface (ADCIF)	Programming components External ADC interface of MCS
19 "Memory Configuration (MCFG)"	Memory Configuration (MCFG)	Programming components Memory assignment for MCS
20 "TIM0 Input Mapping Module (MAP)"	TIM0 Input Mapping Module (MAP)	Dedicated functionality engine management services
21 "Digital PLL Module (DPLL)"	Digital PLL (DPLL)	Dedicated functionality engine management services
22 "MCS to DPLL Interface of DPLL Module (MCS2DPLL)"	MCS to DPLL Interface of DPLL Module (MCS2DPLL)	Dedicated functionality engine management services; DPLL access interface for MCS
23 "Sensor Pattern Evaluation (SPE)"	Sensor Pattern Evaluation Module (SPE)	Dedicated functionality BLDC support
24 "Interrupt Concentrator Module (ICM)"	Interrupt Concentrator Module (ICM)	Infrastructure components
25 "Output Compare Unit (CMP)"	Output Compare Unit (CMP)	Safety features
26 "Monitor Unit (MON)"	Monitoring Unit (MON)	Safety features

3 GTM Architecture

3.1 Overview

Find here a list of indices and their range which are used in all chapters with the same semantic:

- i:={0, 1,..., NCCM} index of cluster or module instance in the GTM device
- j in case the index refers to a cluster/module j!=i
- x:={0, 1,..., 15} index of a channel inside a module
- d:={0, 1,..., 11} index of DTM unit after TOM / ATOM / TIO
- g:={0, 1} index of group of resources / functions

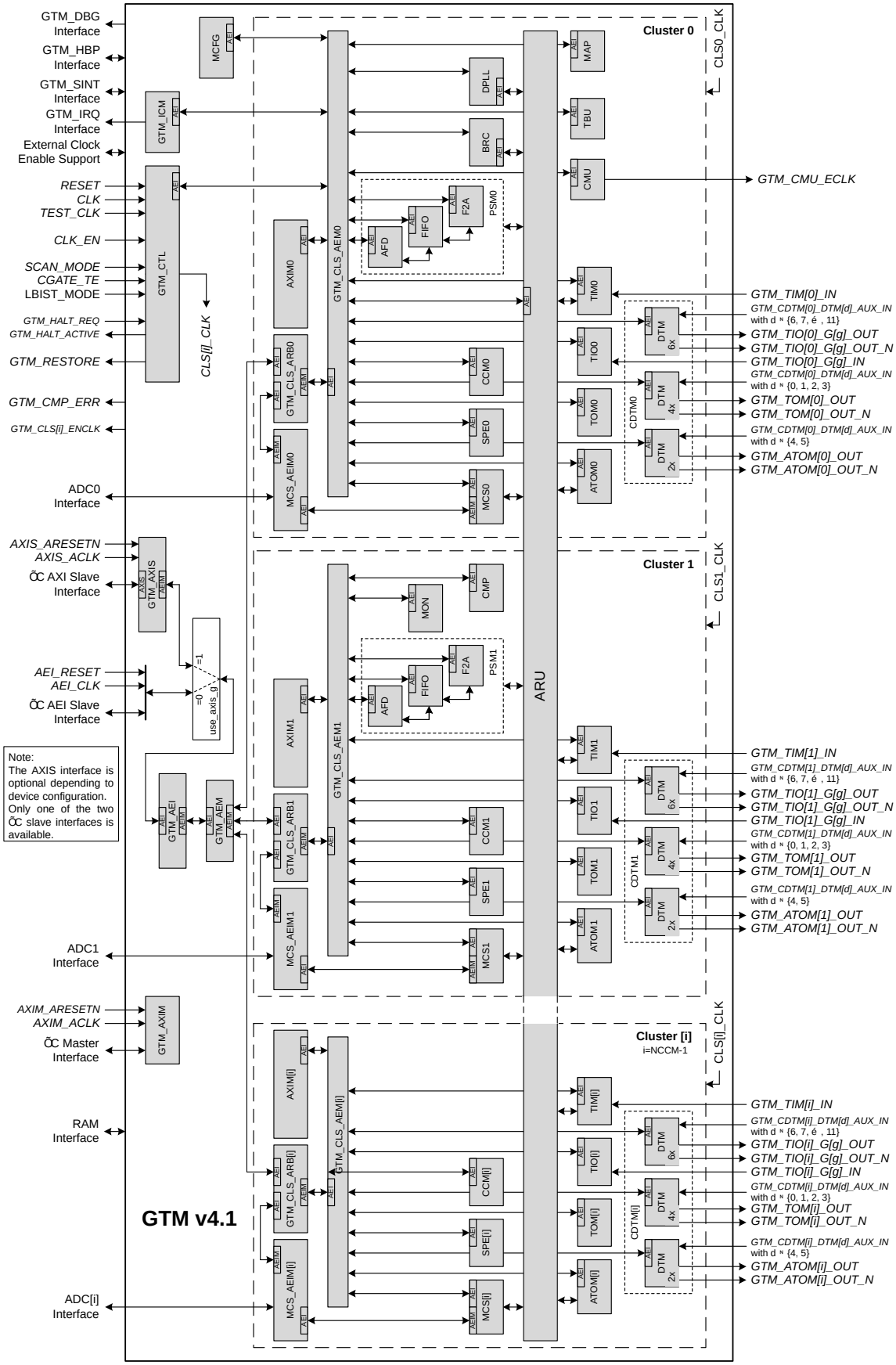
In addition, more indices might be introduced at the beginning of the following chapters. These additional indices with their corresponding ranges are used throughout the respective chapters they are defined in.

As already mentioned in chapter *2 "Introduction"* the GTM−IP forms a generic timer platform that serves different application domains and different classes within these application domains. Depending on these multiple requirements of application domains, multiple device configurations with different number of submodules (i.e. ATOM, BRC, MCS, PSM, SPE, TIM, TOM, and DTM) and different number of channels per submodule (if applicable) are possible.
The device−dependent configuration (i.e. the number of submodules) is listed in the device specific documentation.
The Parameter Storage Module (PSM) is only a virtual hierarchy and consists of the submodules F2A, FIFO and AFD.
The Cluster Dead Time Module is also a virtual hierarchy and consists of up to twelve DTM modules. It depends on the GTM device configuration which of the twelve DTM instances are available. In general, the first four DTM modules (d∈{0, 1, 2, 3}) are connected to the outputs of the TOM instance i of the cluster i. The DTM instances (d∈{4, 5}) are connected to the outputs of the ATOM instance i of this cluster i. The DTM instances (d∈{6, 7,..., 11}) are connected to the outputs of the TIO instance i of this cluster i.

The cluster view of a GTM−IP architecture is depicted in Figure 1 *1 "GTM Architecture Block Diagram"* . This is a generic figure which shows an exemplary GTM−IP device configuration.

The device dependent configuration (i.e. the count of submodules and channels per submodule) is listed in the device specific Appendix B of this document.

Figure 1 GTM Architecture Block Diagram

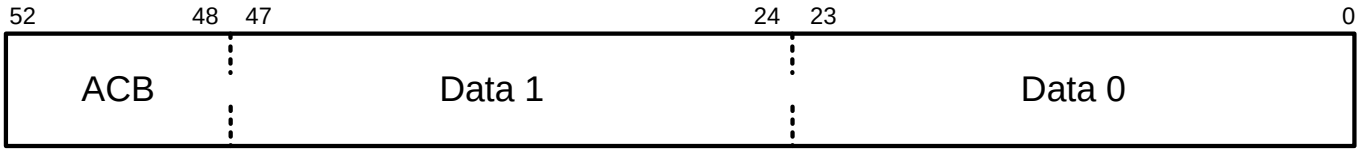


The GTM-IP is divided in multiple clusters 0...n. A certain number of modules exist in each cluster. The operating frequency of a cluster can be configured to OFF, *CLK* or *CLK* / 2 by appropriately configuring **GTM_CLS_CLK_CFG.CLS[i]_CLK_DIV** . The clock enable generation can be implemented internal to the GTM-IP or external. In case of an external enable generation, *CLK_EN* is used to generate the internal clocks. In addition, an enable watchdog is implemented to monitor the correctness of the externally applied enable signals *CLK_EN* . More about clocking in the GTM-IP follows in *3.2 "Introduction to Clocking"* .

The central component of the GTM–IP is the Advanced Routing Unit (ARU), where most of the submodules are located around and connected to. The ARU, together with the Broadcast (BRC) and the Parameter Storage Module (PSM), forms the infrastructural part of the GTM. The ARU is able to route data from a connected source submodule to a connected destination submodule. The routing is done in a deterministic manner with a Round Robin scheduling scheme of connected channels which receive data from ARU and with a worst case round–trip time.

The routed data word size of the ARU is 53 bit. The data word can logically be split into three parts. These parts are shown in figure *2 "ARU Data Word Description"* . Bits [23:0] and bits [47:24] typically hold data for the operation registers of the GTM–IP. This can be, for example, the duty cycle and period duration of a measured PWM input signal or the output characteristic of an output PWM to be generated. Another possible content of Data0 and Data1 is two 24 bit values of the GTM–IP time bases *TBU_TS0_BASE* , *TBU_TS1_BASE* and *TBU_TS2_BASE* . Bits [52:48] can contain control bits to send control information from one submodule to another. These ARU Control Bits (ACB) can have a different meaning for different submodules.
It is also possible to route data from a source to a destination and the destination acts later on as source for another destination. These routes through the GTM–IP are further called data streams. For a detailed description of the ARU submodule please refer to chapter *5 "Advanced Routing Unit (ARU)"* .

Figure 2 ARU Data Word Description



The BRC is able to distribute data from one source module to more than one destination modules connected to the ARU. The PSM submodule consists of three subunits, the AEI–to–FIFO Data Interface (AFD), FIFO–to–ARU Interface (F2A) and the FIFO itself. The PSM serves as a data storage for incoming data characteristics or as parameter storage for outgoing data. This data is stored in a RAM that is logically located inside the FIFO subunit, but physically the RAM is implemented and integrated by the silicon vendor with his RAM implementation technology. Therefore, the GTM–IP provides the interface to the RAM at its module boundary. The AFD subunit is the interface between the FIFO and the GTM SoC system bus interface AEI (please see section *3.5.1 "GTM–IP Generic Bus Interface (AEI)"* for detailed discussion). The F2A subunit is the interface between the FIFO subunit and the ARU.

Signals are transferred into the GTM–IP at the Timer Input Modules (TIM). These modules are able to filter the input signals and annotate additional information. Each channel is, for example, able to measure pulse high or low times and the period of a PWM signal in parallel and route the values to ARU for further processing. The internal operation registers of the TIM submodule are 24–bit wide.

The Clock Management Unit (CMU) which generates up to 14 clock resolutions for the submodules of the GTM and up to three GTM external clocks. The clock resolutions are *CMU_CLK_RES* [x:x] and *CMU_FXCLK_RES* [y:y], where x ∈ {0, 1, ..., 8} and y ∈ {0, 1, ..., 3}. These signals are routed to all clusters via ICPAs. Within a cluster, they are used as clock enables regulating the resolution (or rate) at which actions are performed. The external clocks are *CMU_ECLK* [z:z], where z ∈ {0, 1, 2}. In contrast to *CMU_CLK_RES* [x:x], these are approximately 50% duty cycle clocks. The primary clock source for this submodule is cluster 0 clock signal (*CLS0_CLK*) which can be configured via **GTM_CLS_CLK_CFG.CLS[0]_CLK_DIV** to OFF, *CLK* or *CLK* /2. For a detailed description of the CMU functionality and clocks please refer to chapter *10 "Clock Management Unit (CMU)"* .

The TBU provides up to three independent common time bases for the GTM–IP. In general, the number of time bases depends on the implemented device. If three time bases are implemented, two of these time bases can also be clocked with the digital PLL (DPLL) *SUB_INC1C* and *SUB_INC2C* outputs. The DPLL generates more frequent clock signals *SUB_INC1* , *SUB_INC2* , *SUB_INC1C* and *SUB_INC2C* on behalf of the frequencies of up to two input signals. These two input signals can be selected out of six incoming signals from the TIM0 submodule. In this submodule the incoming signals are filtered and transferred to the MAP submodule where two of these six signals are selected for further processing inside the DPLL.

Signal outputs are generated with the Dead Time Module (DTM), Timer Output Modules (TOM) and the ARU–connected TOMs (ATOM). Each TOM channel is able to generate a PWM signal at its output. Because of the integrated shadow register even the generation of complex PWM outputs is possible with the TOM channels by serving the parameters with the CPU. It is possible to trigger TOM channels for a successor TOM submodule through a trigger line between TOM[i] channel 15 and TOM[i+1] channel 0. But to avoid long trigger paths the GTM–IP integrator configures, after which TOM submodule instance, a pipeline register is placed into the trigger signal chain. Each pipeline register results in one CLK cycle delay of the trigger signal. Please refer to device specification of silicon vendor for unregistered trigger chain length.

In addition, each TOM submodule integrates functions to drive one BLDC engine. This BLDC support is established together with the TIM and Sensor Pattern Evaluation (SPE) submodule.

The ATOMs offer the additional functionality to generate complex output signals without CPU interaction by serving these complex waveform characteristics by other submodules that are connected to the ARU like the PSM or Multi Channel Sequencer (MCS). While the internal operation and shadow registers of the TOM channels are 16 bit wide, the operation and shadow registers of the ATOM channels are 24 bit wide to have a higher resolution and to have the opportunity to compare against time base values coming from the TBU.

It is possible to trigger ATOM channels for a successor ATOM submodule through a trigger line between ATOM[i] channel 7 and ATOM[i+1] channel 0. But to avoid long trigger paths the GTM–IP integrator can configure after which ATOM submodule instance a pipeline register is placed into the trigger signal chain. Each pipeline register results in one *CLK* cycle delay of the trigger signal. Please refer to device specification of silicon vendor for unregistered trigger chain length.

Together with the MCS the ATOM is able to generate an arbitrary predefined output sequence at the GTM–IP output pins. The output sequence is defined by instructions located in RAM connected to the MCS submodule. The instructions define the points, where an output signal should change or react to other signal inputs. The output points can be one or two time stamps (or even angle stamp in case of an engine management system) provided by the TBU. Since the MCS is able to read data from the ARU it is also able to operate on incoming data routed from the TIM. Additionally, the MCS can process data that is located in its connected RAMs. The MCS RAM is located logically inside the MCS while the silicon vendor has to implement its own RAM technology there.

The two modules Compare Module (CMP) and Monitor Module (MON) implement safety related features. The CMP compares two output channels of the DTM and sends the result to the MON submodule, where the error is signaled to the CPU. The MON module is also able to monitor the ARU and CMU activities.

In the described implementation, the submodules of the GTM-IP have a huge number of different interrupt sources. These interrupt sources are grouped and concentrated by the Interrupt Concentrator Module (ICM) to form a much easily manageable bunch of interrupts that are visible outside of the GTM-IP.

3.2 Introduction to Clocking

The GTM-IP offers many possibilities on different granularity levels to reduce the power budget during run-time. The possibilities include shutting off a particular cluster/module or clocking them at a reduced frequency, among others.

The first layer of configurability is on the cluster level. A cluster clock can be shut off completely or configured to use the GTM global clock or the GTM global clock halved by appropriately configuring **GTM_CLS_CLK_CFG** . In the GTM-IP, cluster clocks can be configured differently. The CCM module also offers the possibility to enable/disable different modules in the cluster by appropriately configuring **CCM[i]_CMU_CLK_CFG.CLK[y]_SRC** , where 0 <= i <= NCCM - 1 and 0 <= y <= 7.

The second layer of configurability is offered by the CMU module (exists only in cluster 0). This module takes the cluster clock as input and uses the clock dividers within to generate different clock resolutions (*CMU_CLK_RES* [x:x] (x ∈ {0, 1,..., 8})) that are routed through ICPA (refer to *3.10 "Inter-Cluster Pulse Adapter"*) to the CCM modules in all the clusters. These signals give the user the possibility to further regulate the rate (or the resolution) at which the functions in different modules run at (see *36 "CMU Block Diagram "*). A *CMU_CLK_RES* [x:x] signal can be enabled/disabled by appropriately configuring **GTM.CLS[0].CMU.CLK_EN** .

The third layer of configurability is offered by the CCM module (exists in every cluster). Different clusters can be clocked differently. This module samples the *CMU_CLK_RES* [x:x] resolution signals at the cluster clock. The sampled clocks are then routed to different modules within the cluster in which they are used as clock enables. This allows different modules within a cluster to run at different resolutions.

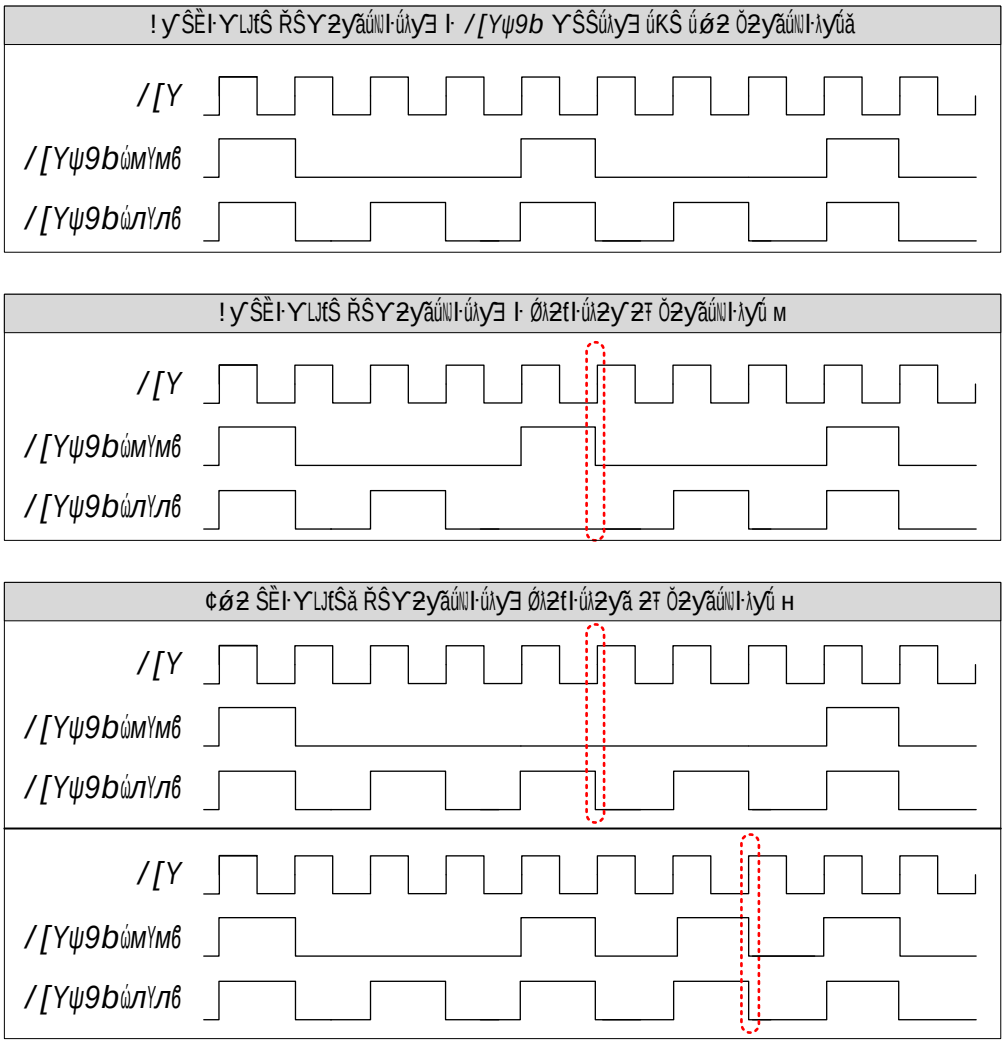
Attention: The GTM-IP offers various debugging mechanisms (more about that in *3.14 "GTM-IP Software Debugger Support"*). When debugging is active and *GTM_HALT_REQ* is asserted, the cluster clocks are disabled in order to stall the internal operations in the GTM-IP during the debugging session.

3.3 Support for External Clock Enables

If the GTM-IP is ordered with the device configuration parameter INT_CLK_EN_GEN = 0, external clock enable is applied. With that, *CLK_EN* [0:0] and *CLK_EN* [1:1] are gated with the GTM clock resulting in *CLK* and *CLK* / 2, respectively. The following constraints on *CLK_EN* must be fulfilled:

- If *CLK_EN* [1:1] = 1 at a rising clock edge of the GTM clock, then *CLK_EN* [0:0] must evaluate to 1 at that rising edge too. If violated, **GTM_IRQ_NOTIFY.CLK_PER_ERR** is set indicating that the period of *CLK* / 2 is incorrect (not double that of *CLK*) due to the unexpected *CLK_EN* [1:1] = 1.
- If *CLK_EN* [1:1] = 1 at a rising clock edge of the GTM clock, then it must evaluate to 0 at the next rising clock edge at which *CLK_EN* [0:0] = 1. If *CLK_EN* [1:1] = 0 and *CLK_EN* [0:0] = 1 at a rising clock edge of the GTM clock, then *CLK_EN* [1:1] must evaluate to 1 at the next rising clock edge at which *CLK_EN* [0:0] = 1. That is, in every two consecutive rising clock edges at which *CLK_EN* [0:0] is evaluated as 1, *CLK_EN* [1:1] must evaluate to 1 only once at one of the two edges. If either is violated, **GTM_IRQ_NOTIFY.CLK_EN_ERR** is set.

Figure 3 Demonstrating violations on CLK_EN



If one of the constraints is violated, internal operation of the GTM–IP is halted (*CLK* and *CLK* / 2 are tied off). During that time, newly issued AEI transactions at the GTM_AEI bridge will be accepted but no AEI transactions will be executed. Ongoing AEI transactions are halted. The internal operation of the GTM–IP and the halted AEI transactions are resumed once the constraints are fulfilled again.

If one of the constraints is violated, the erroneous state of *CLK_EN* is stored to **GTM_IRQ_NOTIFY.CLK_EN_ERR_STATE** and the expected state of *CLK_EN* is stored to **GTM_IRQ_NOTIFY.CLK_EN_EXP_STATE** . The two bit fields are only written on the first violation – succeeding violations will not update them.

The signal *GTM_WDG_ERR* is communicated to the outside and can be viewed as an error interrupt with its mode configured as "level interrupt mode" (see *3.12.1 "Level Interrupt Mode"*). If the first constraint is violated and **GTM_EIRQ_EN.CLK_PER_ERR_EIRQ_EN** = 1, *GTM_WDG_ERR* is asserted. If the second constraint is violated and **GTM_EIRQ_EN.CLK_EN_ERR_EIRQ_EN** = 1, *GTM_WDG_ERR* is asserted. A hardware clear on *GTM_ERR_IRQ_CLR* clears *GTM_WDG_ERR* .

If the first constraint is violated and **GTM_IRQ_EN.CLK_PER_ERR_IRQ_EN** = 1 or the second constraint is violated and **GTM_IRQ_EN.CLK_EN_ERR_IRQ_EN** = 1, then an interrupt is triggered on *GTM_AEI_IRQ* .

If the first constraint is violated and **GTM_EIRQ_EN.CLK_PER_ERR_EIRQ_EN** = 1, then an interrupt is triggered on *GTM_ERR_IRQ* . If the second constraint is violated and **GTM_EIRQ_EN.CLK_EN_ERR_IRQ_EN** = 1, then an interrupt is triggered on *GTM_ERR_IRQ* . In contrast to all other error interrupt sources in the GTM–IP, **GTM_EIRQ_EN.CLK_PER_ERR_EIRQ_EN** and **GTM_EIRQ_EN.CLK_EN_ERR_IRQ_EN** are initialized to 1 when reset. That is, the error interrupt source for the external clock enable is enabled by default.

Attention:

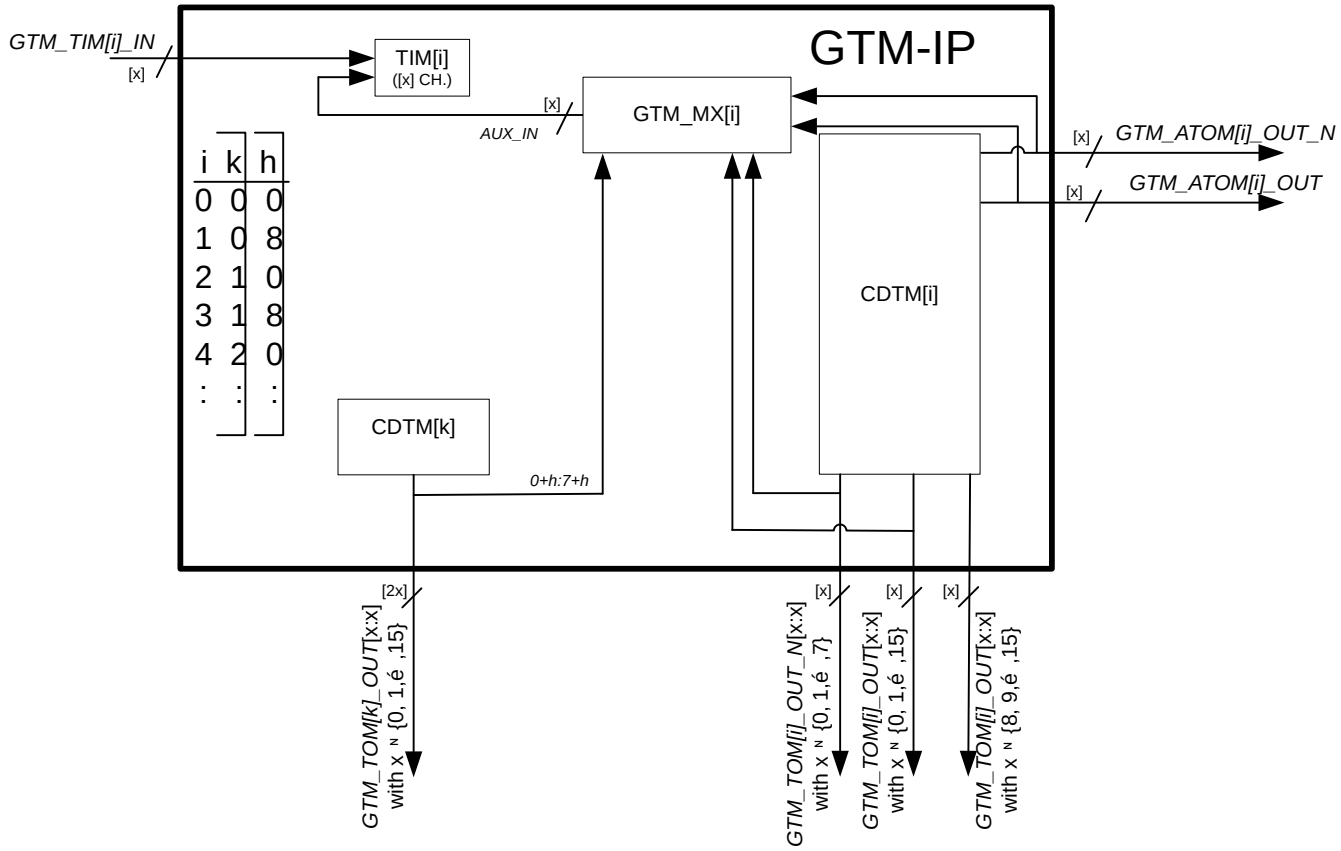
By writing a one to **GTM_IRQ_FORCINT.TRG_CLK_PER_ERR** / **GTM_IRQ_FORCINT.TRG_CLK_EN_ERR** , the bit field **GTM_IRQ_NOTIFY.CLK_PER_ERR** / **GTM_IRQ_NOTIFY.CLK_EN_ERR** is forced to 1. As long as the notify bit fields are not cleared, **GTM_IRQ_NOTIFY.CLK_EN_ERR_STATE** and **GTM_IRQ_NOTIFY.CLK_EN_EXP_STATE** will not update.

The initial state of the external clock enable after reset (provided by the silicon vendor) is important for the watchdog to detect the errors reliably. If the state of the external clock after reset is different than that provided by the silicon vendor, **GTM_IRQ_NOTIFY.CLK_EN_ERR** is set.

3.4 Connectivity between Modules

On the GTM-IP top level there are some configurable signal connections from the signal output of the DTM modules to the input signals of the TIM modules.

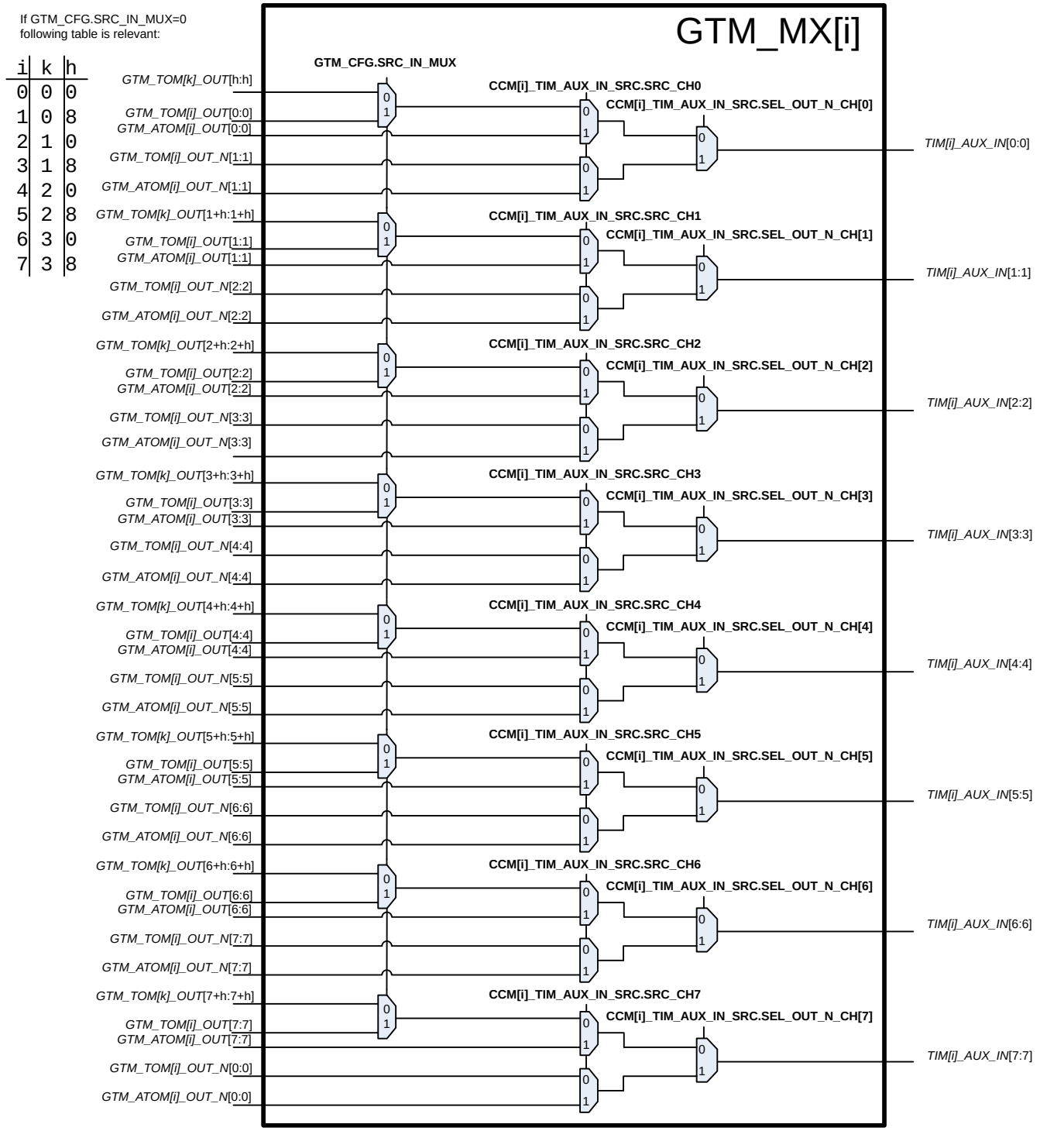
Figure 4 GTM-IP Signal Multiplex



The next diagram gives an overview of the connectivity for different configuration of bit field **GTM_CFG.SRC_IN_MUX** and the cluster configuration register **CCM[i]_TIM_AUX_IN_SRC** . The source selection is defined per channel with the bit fields **CCM[i]_TIM_AUX_IN_SRC.SRC_CH0** , **CCM[i]_TIM_AUX_IN_SRC.SRC_CH1** , **CCM[i]_TIM_AUX_IN_SRC.SRC_CH2** , **CCM[i]_TIM_AUX_IN_SRC.SRC_CH3** , **CCM[i]_TIM_AUX_IN_SRC.SRC_CH4** , **CCM[i]_TIM_AUX_IN_SRC.SRC_CH5** , **CCM[i]_TIM_AUX_IN_SRC.SRC_CH6** , **CCM[i]_TIM_AUX_IN_SRC.SRC_CH7** and **CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[x]** .

The calculation of the index k, h and the connectivity of the next figure covers devices with 8 TIM modules.
In case a device is equipped with more TIM instances as ATOM instances (NTIM > NATOM), the signals *GTM_ATOM[i]_OUT* [x:x], *GTM_ATOM[i]_OUT_N* [x:x] are tied to 0; i>NTIM.
GTM_CFG.SRC_IN_MUX =1: In case a device is equipped with more TIM instances as TOM instances (NTIM > NTOM), the signals *GTM_TOM[i]_OUT* [x:x], *GTM_TOM[i]_OUT_N* [x:x] are tied to 0; i>NTIM.
GTM_CFG.SRC_IN_MUX =0: In case a device is equipped with lesser TIM instances as TOM instances (2*NTIM < NTOM), the signals *GTM_TOM[i]_OUT* [x+h:x+h] are tied to 0; i>NTIM.

Figure 5 TIM Auxiliary Input Multiplexing



The trigger out of TIM (i.e. the signals *TIM_EXT_CAPTURE* [7:0] of each TIM instance i) are routed to ATOM instance i and TOM instance i with i∈{0, 1,..., NTIM−1} (NTIM defines the number of available TIM instances).
This TIM trigger is used to trigger inside the ATOM or TOM instance either a channel or the global control register of AGC or TGC0/TGC1 unit.

Figure 6 TIM External Capture Forwarding to TOM and ATOM

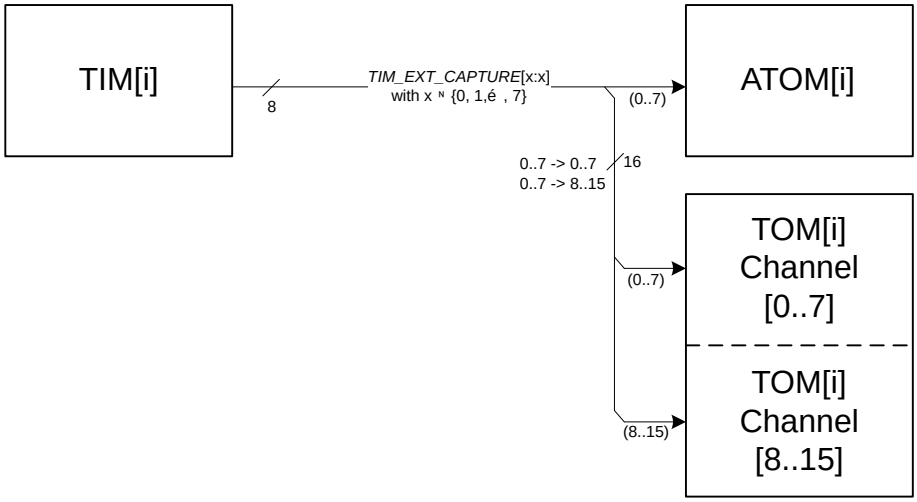
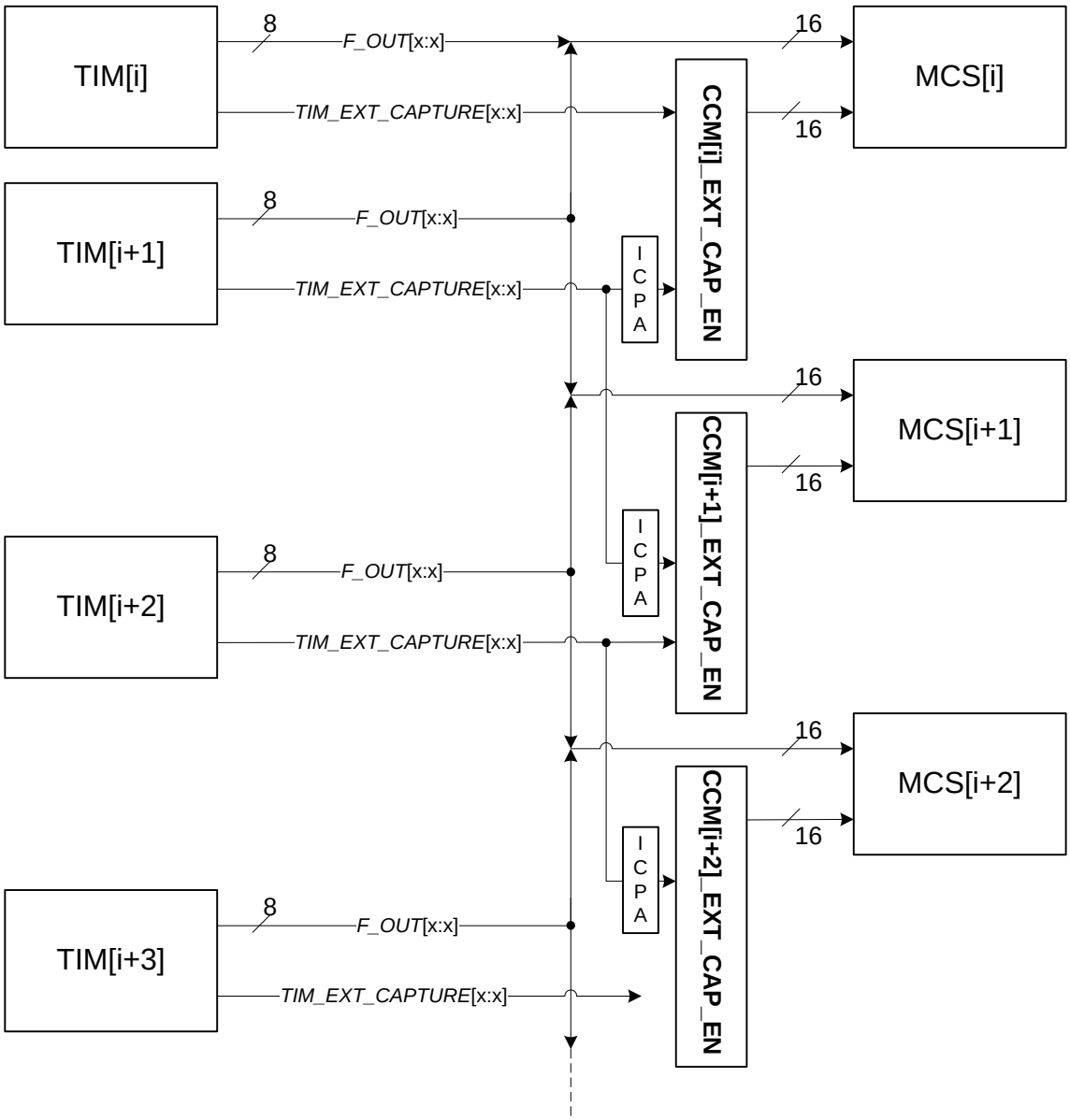


Figure 7 TIM to MCS Signal Forwarding

x=0,..,7; number of available TIM channel of instance



The trigger out of TIM (i.e. the signals *TIM_EXT_CAPTURE* [7:0] of each TIM instance i) is additionally routed to the MCS instance i. This trigger forwarding is enabled by register **CCM[i]_EXT_CAP_EN** .

The following two figures show, how signals are routed to the DTM module.

Figure 8 Routing of CMU Clock Signals to DTM Modules

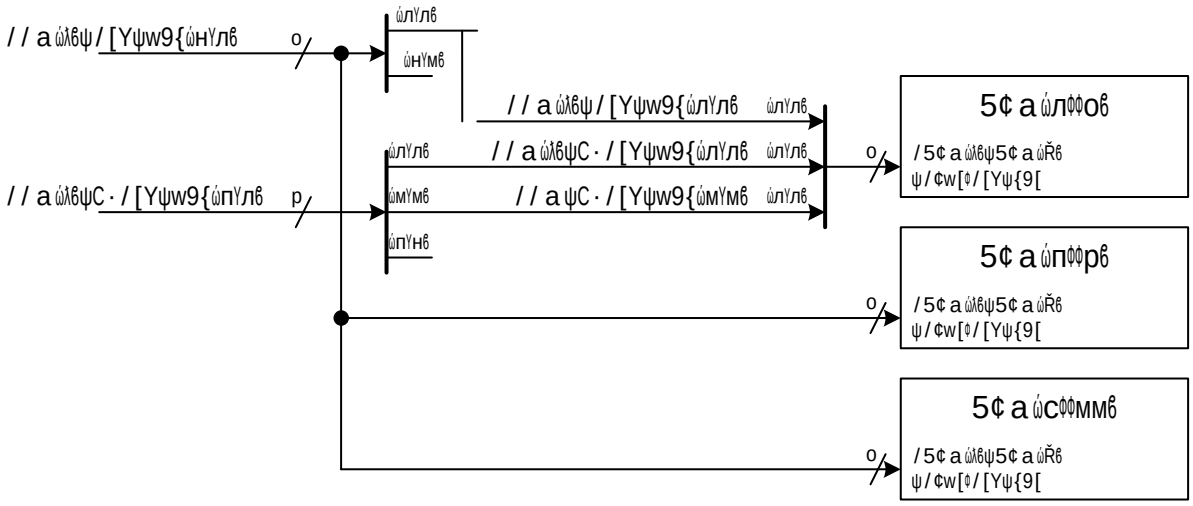
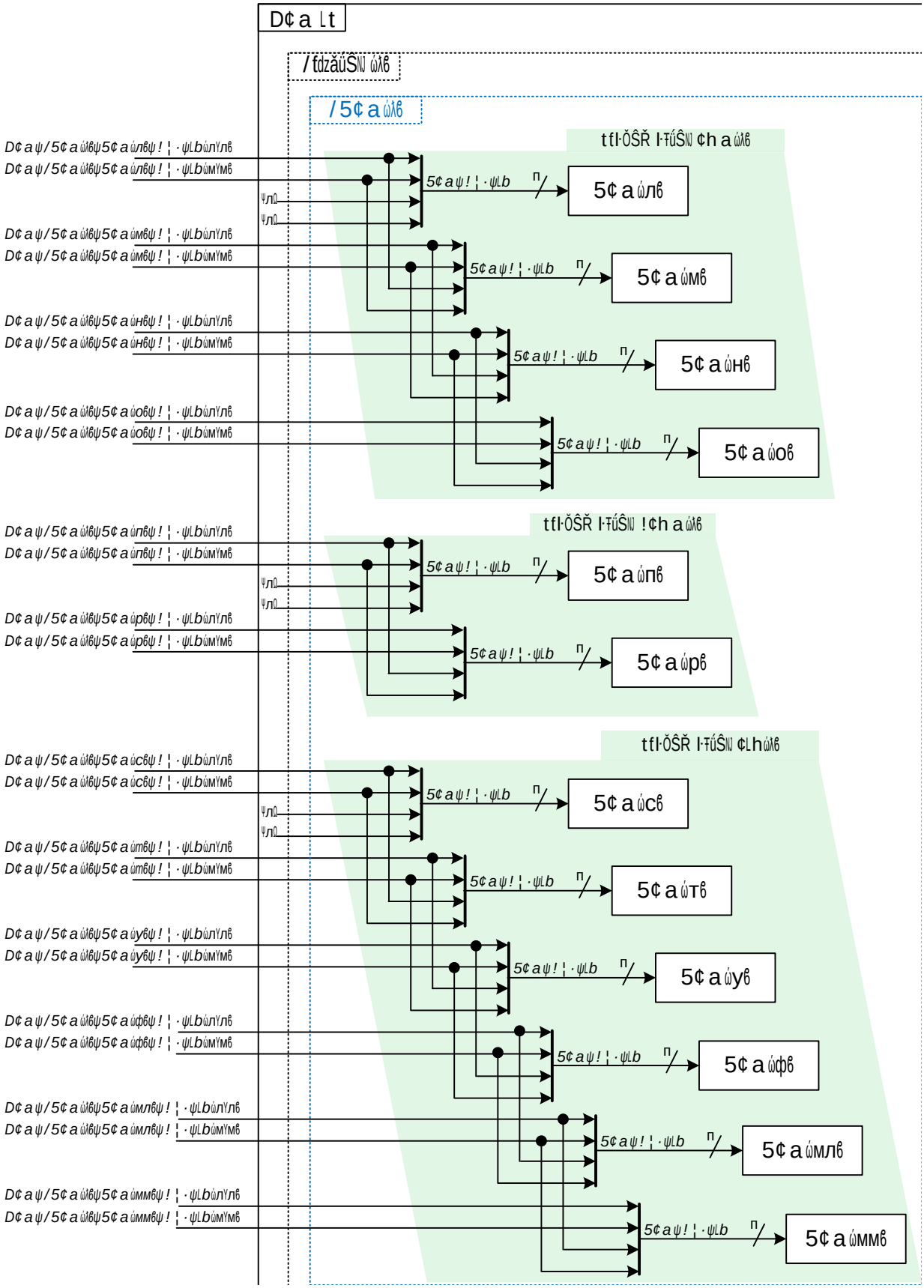


Figure 9 Routing of DTM AUX Input Signals



3.5 GTM–IP Interfaces

In general, the GTM-IP is divided into multiple interface groups. Two interface groups represent the ports of the GTM-IP where incoming signals are assembled and outgoing signals are created. These interfaces are therefore connected to the GTM-IP input submodule TIM and to the GTM-IP output submodules DTM.

Another interface is the bus master interface where the GTM-IP is connected to the SoC system bus and actively drive read and write operations onto the SoC system bus. This generic bus interface is described in more detail in section *3.5.1 "GTM-IP Generic Bus Interface (AEI)"* .

Another interface is the bus slave interface where the GTM-IP is connected to the SoC system bus to allow access to the GTM internal resources. Two options are provided for the slave interface: A generic bus interface is described in more detail in section *3.5.1 "GTM-IP Generic Bus Interface (AEI)"* , and an AXI slave described in section *3.5.2 "AXI Slave"* .

The last interface is the interrupt controller interface. The GTM-IP provides several interrupt lines coming from the various submodules. These interrupt lines are concentrated inside the ICM and have to be adapted to the dedicated microcontroller environment where each interrupt handling can look different. The interrupt concept is described in more detail in section *3.12 "GTM-IP Interrupt Concept"* .

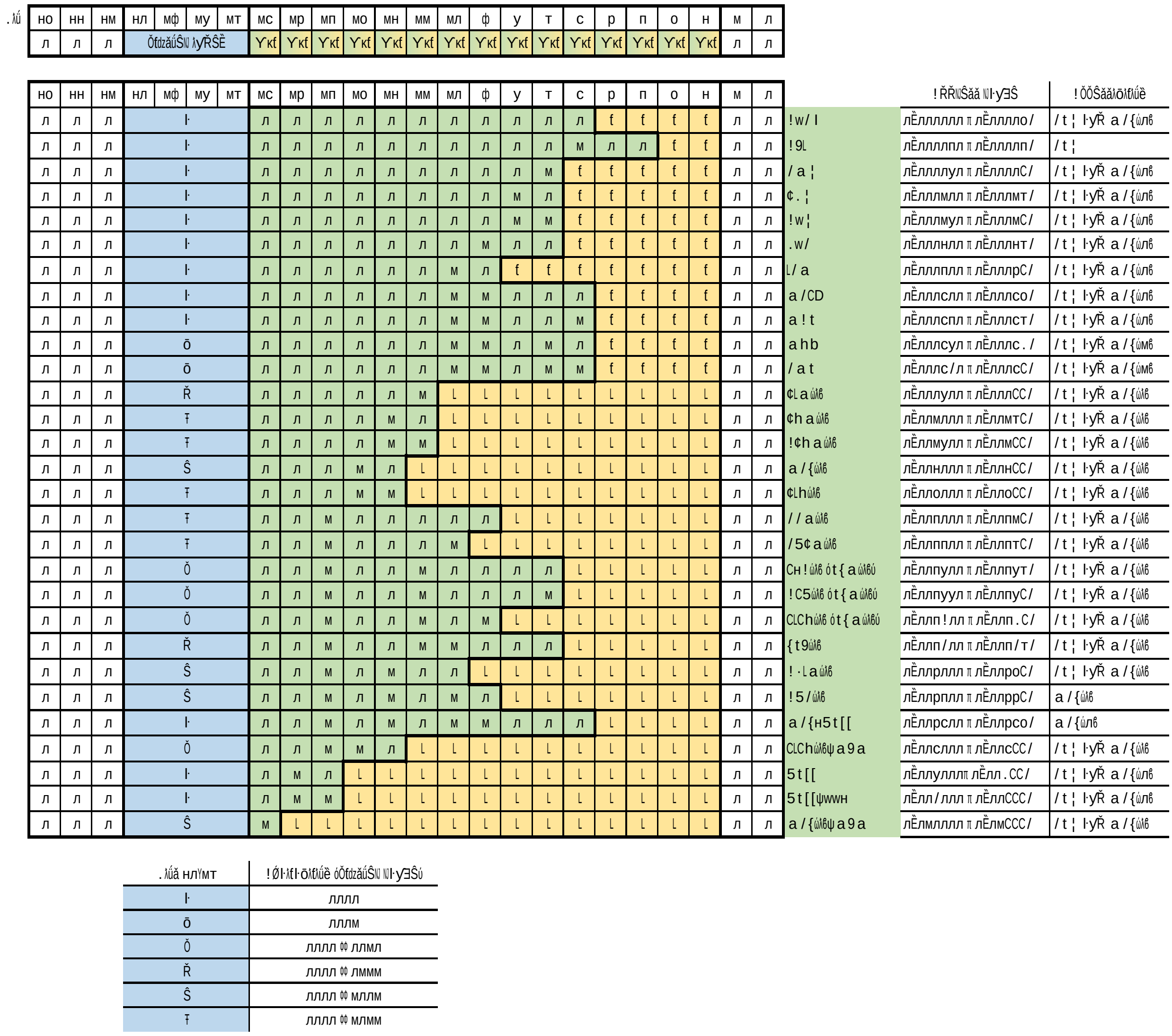
3.5.1 GTM-IP Generic Bus Interface (AEI)

The GTM-IP is equipped with a generic bus interface that is widely adapted to different SoC bus systems. This generic bus interface is called AE_Interface (AEI). The adaptation of the AEI to SoC buses is typically done with a bridge module translating the AEI signals to the SoC bus signals of the silicon vendor. The silicon vendor has to decode a GTM-IP base address (higher address bits 31:21), which drives the signal *AEI_SEL* =1. The AEI bus signals are listed in the signal description *3.19 "GTM TOP-Level Signal Description"* .

The address map in GTM-IP is word-addressable (the two LSBs in the address bus are expected to be 0b00). An attempt for byte-addressing shall return an illegal address access (0b10) on *AEI_STATUS* .

The GTM-IP comprises up to 12 clusters each of which comprising several modules depending on the configuration (see *84 "GTM Device Configuration Variables, Define Available Module Resources"* and *1 "GTM Architecture Block Diagram"*). Figure *10 "Overview of GTM Address Space"* summarizes the address map of the GTM-IP with a superset of GTM modules ordered ascendingly based on their 16:0 addresses. The absolute address of a register is composed of the cluster address (bits 20:17) followed by the module address (16:ZZ) and the local address of the register (ZZ-1:0) where ZZ varies amongst the modules (4 <= ZZ <= 16).

Figure 10 Overview of GTM Address Space



The modules (ARCH, AEI, CMU, TBU, BRC, ICM, MCFG, F2A, AFD, FIFO, CMP, DPLL, MCS2DPLL) and DPLL memory (DPLL_RR2) may only be available in cluster 0 address space.

The modules (MON, CMP) may only be available in cluster 1 address space.

The modules (F2A, AFD, FIFO) and the FIFO memory (FIFO_MEM) may only be available in cluster {0, 1, 2} address spaces, and may only fill out the clusters starting from cluster 0 onwards.

The (TIM, TOM, ATOM, MCS, CCM, SPE, AXIM, DTM) modules and the MCS memory (MCS_MEM) may only fill out the clusters starting from cluster 0 onwards.

Only the TIO module may arbitrarily be available in any cluster.

The ARCH configuration registers (accessible from CPU and MCS[0]) and the AEI configuration registers (accessible only from CPU) share the following properties:

- They are responsible for configuring functionalities spanning/impacting multiple clusters (**MCS_AEM_DIS** , **GTM_RST** , **GTM_CFG** , **GTM_CLS_CLK_CFG** , **GTM_ARU_COM_DIS**).
- They are responsible for atomically switching on/off functionalities across multiple clusters (**MCS_AEM_DIS** , **GTM_RST** , **GTM_CFG** , **GTM_CLS_CLK_CFG** , **GTM_ARU_COM_DIS**).
- They are responsible for configuring and providing status of the interfaces (**GTM_CTRL** , **GTM_IRQ_NOTIFY** , **GTM_AEI_ADDR_XPT** , **GTM_AEI_STA_XPT** , **MCS_AEM_DIS** , **GTM_RST**).
- They are still accessible when cluster 0 is switched off.

The AEI status signal may drive one of the following values:

Table 2 AEI Status Signal

AEI_STATUS coding	Description
0b00	No Error
0b01	Illegal Byte Addressing
0b10	Illegal Address Access
0b11	Unsupported Address

The evaluation of the status of a read / write access is executed based on a priority scheme. The ordering of priorities is defined in the following:

1. PRIO1 checks are validated first, if a check condition matches, the corresponding *AEI_STATUS* is returned, otherwise checking continues with PRIO2.
2. PRIO2 checks are validated next, if a check condition matches, the corresponding *AEI_STATUS* is returned, otherwise checking continues with PRIO3.
3. PRIO3 checks are validated next, if a check condition matches, the corresponding *AEI_STATUS* is returned, otherwise checking continues with PRIO4.
4. PRIO4 checks are validated next, if a check condition matches, the corresponding *AEI_STATUS* is returned, otherwise *AEI_STATUS* = 0b00 is returned.

The signal value 0b01 is returned:

- PRIO1: if the bus address is not an integer multiple of 4 (byte addressing).

For cluster or module clock disable functionality, the following address range definitions exist:

Table 3 Cluster address range access is disabled by GTM_CLS_CLK_CFG.CLS[i]_CLK_DIV=0b00

Cluster i	Address range
0	0x000050 – 0x0003FF; 0x000600 – 0x01FFFF
1	0x020000 – 0x03FFFF
2	0x040000 – 0x05FFFF
3	0x060000 – 0x07FFFF
4	0x080000 – 0x09FFFF
5	0x0A0000 – 0x0BFFFF
6	0x0C0000 – 0x0DFFFF
7	0x0E0000 – 0x0FFFFF
8	0x100000 – 0x11FFFF
9	0x120000 – 0x13FFFF
10	0x140000 – 0x15FFFF
11	0x160000 – 0x17FFFF

Table 4 Module address range access is disabled by module clock enable = 0b0

module clock enable	implemented in cluster i if condition is true	module name	module address offset range
CCM[i]_CFG.EN_TIM	i < NTIM	TIM	0x000800 – 0x000FFF
CCM[i]_CFG.EN_TOM_SPE_TDTM	i < NTOM	TOM	0x001000 – 0x017FFF
CCM[i]_CFG.EN_TOM_SPE_TDTM	i < NTOM & CDTM[i][d] (d∈{0, 1, 2, 3})	TDTM	0x004400 – 0x0044FF
CCM[i]_CFG.EN_TOM_SPE_TDTM	i < NSPE	SPE	0x004C00 – 0x004C7F
CCM[i]_CFG.EN_ATOM_ADTM	i < NATOM	ATOM	0x001800 – 0x01FFFF
CCM[i]_CFG.EN_ATOM_ADTM	i < NATOM & CDTM[i][d] (d∈{4, 5})	ADTM	0x004500 – 0x00457F
CCM[i]_CFG.EN_MCS	i < NMCS	MCS	0x002000 – 0x002FFF
CCM[i]_CFG.EN_MCS	i < NMCS	MCS memory	0x010000 – 0x01FFFF
CCM[i]_CFG.EN_DPLL_MAP	i < NDPLL	DPLL	0x008000 – 0x0081FF

module clock enable	implemented in cluster i if condition is true	module name	module address offset range
CCM[i]_CFG.EN_DPLL_MAP	i < NDPLL	DPLL RAM1A	0x008200 – 0x0083FF
CCM[i]_CFG.EN_DPLL_MAP	i < NDPLL	DPLL RAM1BC	0x008400 – 0x0089FF
CCM[i]_CFG.EN_DPLL_MAP	i < NDPLL	DPLL	0x008A00 – 0x00BFFF
CCM[i]_CFG.EN_DPLL_MAP	i < NDPLL	DPLL RAM2	0x00C000 – 0x00FFFF
CCM[i]_CFG.EN_DPLL_MAP	i < NDPLL	MCS2DPLL	0x005600 – 0x00563F
CCM[i]_CFG.EN_DPLL_MAP	i < NMAP	MAP	0x000640 – 0x00067F
CCM[i]_CFG.EN_BRC	i < NBRC	BRC	0x000200 – 0x00027F
CCM[i]_CFG.EN_PSM	i < NPSM	F2A	0x004800 – 0x00487F
CCM[i]_CFG.EN_PSM	i < NPSM	AFD	0x004880 – 0x0048FF
CCM[i]_CFG.EN_PSM	i < NPSM	FIFO	0x004A00 – 0x004BFF
CCM[i]_CFG.EN_PSM	i < NPSM	FIFO memory	0x006000 – 0x006FFF
CCM[i]_CFG.EN_CMP_MON	i == NMON & i == 1	MON	0x000680 – 0x0006BF
CCM[i]_CFG.EN_CMP_MON	i == NCMP & i == 1	CMP	0x0006C0 – 0x0006FF
CCM[i]_CFG.EN_TIO_DTM	i < NCCM & CTIO[i]	TIO	0x003000 – 0x003FFF
CCM[i]_CFG.EN_TIO_DTM	i < NCCM & CDTM[i][d] (d∈{6, 7, ..., 11})	DTM after TIO	0x004580 – 0x0046FF

The signal value 0b10 is returned for an access where the target address is not accessible (cluster / module shut off or internal operation ongoing):

- PRIO2: on a read or write, if the address is located in the cluster i address range, which is disabled by **GTM_CLS_CLK_CFG.CLS[i]_CLK_DIV** = 0b00
- PRIO2: on a read or write, if the address is located in the module TIM[i] address range, which is disabled by **CCM[i]_CFG.EN_TIM** = 0b0
- PRIO2: on a read or write, if the address is located in the module TOM[i], SPE[i], CDTM[i]_DTM[3:0] address range, which is disabled by **CCM[i]_CFG.EN_TOM_SPE_TDTM** = 0b0
- PRIO2: on a read or write, if the address is located in the module ATOM[i], CDTM[i]_DTM[5:4] address range, which is disabled by **CCM[i]_CFG.EN_ATOM_ADTM** = 0b0
- PRIO2: on a read or write, if the address is located in the module MCS[i] address range, which is disabled by **CCM[i]_CFG.EN_MCS** = 0b0
- PRIO2: on a read or write, if the address is located in the module DPLL, MAP address range, which is disabled by **CCM[i]_CFG.EN_DPLL_MAP** = 0b0
- PRIO2: on a read or write, if the address is located in the module BRC address range, which is disabled by **CCM[i]_CFG.EN_BRC** = 0b0
- PRIO2: on a read or write, if the address is located in the module PSM[i] address range, which is disabled by **CCM[i]_CFG.EN_PSM** = 0b0
- PRIO2: on a read or write, if the address is located in the module CMP, MON address range, which is disabled by **CCM[i]_CFG.EN_CMP_MON** = 0b0
- PRIO2: on a read or write, if the address is located in the module TIO[i] address range, CDTM[i]_DTM[11:6] address range, which is disabled by **CCM[i]_CFG.EN_TIO_DTM** = 0b0
- PRIO4: on a read or write, if the address is located in the DPLL memory RAM1A for which actually a memory initialization is executed **DPLL_RAM_INI.INIT_1A** = 0b1
- PRIO4: on a read or write, if the address is located in the DPLL memory RAM1BC for which actually a memory initialization is executed **DPLL_RAM_INI.INIT_1BC** = 0b1
- PRIO4: on a read or write, if the address is located in the DPLL memory RAM2 for which actually a memory initialization is executed **DPLL_RAM_INI.INIT_2** = 0b1
- PRIO4: on a read or write, if the address is located in the MCS[i] memory space for which actually a memory initialization is executed **MCS[i]_CTRL_STAT.RAM_RST** = 0b1
- PRIO4: on a read or write, if the address is located in the DPLL memory RAM1A which is actually inaccessible due to DPLL internal activity **DPLL_CTRL_1.DEN** = 0b1

The signal value 0b10 is returned for a protected write access:

- PRIO4: if for the written register all writable bit fields are equipped with the same write protection condition and the protection is active (independently of the written value).

In case of an illegal write access signaled by status 0b10 the register will not be modified.

The signal value 0b11 is returned:

- PRIO3: if for the requested address no GTM register or GTM memory location is defined
- PRIO3: if the requested address is not implemented in the actual device (device dependency attribute of that address evaluates to false)

The signal value 0b00 is returned:

- If no error occurred during AEI access.
- If all register bit fields are defined as read only, a write to this register will not modify any content.
- If a register bit field is defined as read only, a write to this register bit field will not modify the content. All bit fields of the same register which are writable will be written as requested.
- If a register contains a protected bit field and the protection is active the bit field will not be modified. All bit fields of the same register which have no protection will be written as requested.
- If a bit field of a register is defined as reserved, any write access to this bit field has no side effect and a read access to this bit field will always return the value 0.
- If a bit field of a register is not implemented in the actual device (device dependency attribute of that bit field evaluates to false) this is treated as reserved, any write access to this bit field has no side effect and a read access to this bit field will always return the value 0.
- If one or more encoding values of a bit field are defined as reserved values and nothing else is specified, the following default behavior is defined: The written value can be read back and the entire behavior of the circuit is undefined.

3.5.1.1 GTM AEI Bridge Software Reset

The configuration bit field **BRIDGE_MODE.BRG_RST** is used for software reset of the GTM_AEI module.

Writing the value 0b1 to the bitfield **BRIDGE_MODE.BRG_RST** via the configuration interface, will immediately reset the content of the following registers to the initial hardware reset state.

- **BRIDGE_MODE** : Details about which bit fields are reset follow in **BRIDGE_MODE** register description.
- **BRIDGE_PTR1**
- **BRIDGE_PTR2**
- **GTM_AEI_ADDR_XPT**
- **GTM_AEI_STA_XPT**
- Internal registers inside the GTM_AEI module which are not memory mapped are reset too. Exception exists for devices which are equipped with an AXI slave interface, the additional registers which store the AEI sideband signals (e.g. AXIS transaction ID) are not reset.

3.5.2 AXI Slave

3.5.2.1 Functional Characteristics

The AXI slave to AEI split protocol adapter (gtm_axis) implements an AXI compliant slave interface and converts the access to be AEI split compatible. In addition, it implements checks for the correctness of the AXI transaction within the given constraints of the AEI protocol.

Following functionality is supported:

- Full AMBA AXI 3.0 compliant
- Only 32 bit accesses
- 32 bit address
- 32 or 64 bit data (configurable via device configuration variable AXIS_DATA_WIDTH) see chapter *3.5.2.2.1.9 "64bit AXI support "*
- No byte/halfword support
- Only aligned accesses
- Alignment check
- No interleaved ID accesses
- Single burst address counter
- ID check for write data is executed
- Cache, protection and lock signals are not available (are ignored)
- Single clock operation

- Single low active asynchronous reset signal
- Use of gtm_aei AEI split protocol
- Support for two AXI Debugger IDs, see chapter *3.5.2.2.1.7 "Debugger ID comparators "*

3.5.2.2 AXI to AEI Split Adapter Block

3.5.2.2.1 Functional View

Figure 11 Block Diagram AXI Slave

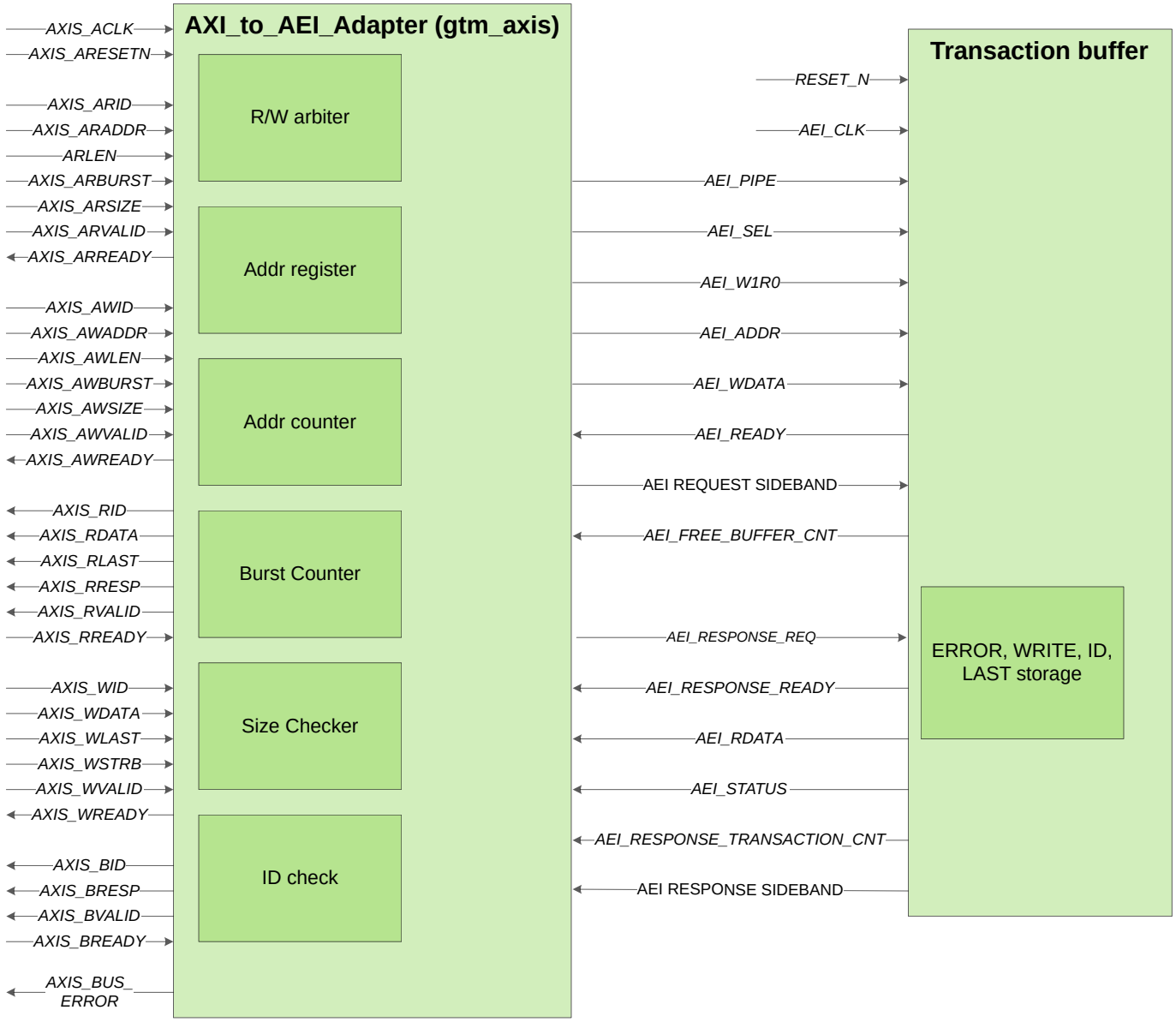


Figure *11 "Block Diagram AXI Slave"* shows the basic modules of the adapter block. The transaction buffer is not part of the adapter block, but part of the block the adapter is integrated in.

3.5.2.2.1.1 GTM-IP base address decoding

The AXI slave interface is equipped with 32-bit address buses *AXIS_ARADDR* and *AXIS_AWADDR* . The GTM-IP internal address space is 21-bit wide. The silicon vendor has to decode a GTM-IP base address (higher address bits 31:21), which will drive the signals *AXIS_ARVALID* / *AXIS_AWVALID* to 1.

3.5.2.2.1.2 Read/Write arbiter

Accesses are executed in order of the request on the address channels. In case two accesses are raised in the same cycle, the Read/Write arbiter decides which access to handle first. In this case, the execution of the read access takes priority over the write access.

3.5.2.2.1.3 Address register and counter

The address register holds the address during a transaction (as it is removed on the AXI before finalizing the full transaction). The counter feature is used to create the individual addresses of burst requests. All AXI burst modes are supported.

3.5.2.2.1.4 Burst counter

The burst counter counts the beats of a requested burst and ensures that all burst accesses are executed and returned to the AXI master.

3.5.2.2.1.5 Size checker

The size checker ensures that only 32bit accesses are executed. Other accesses return an error code on the AXI bus.

3.5.2.2.1.6 ID checker

As on the AXI bus the transaction ID could change on the write data channel, this checker ensures that only the ID of the current on-going transaction is used. If another ID is detected the *AXIS_BUS_ERROR* is raised as interleaved write data is not supported.

3.5.2.2.1.7 Debugger ID comparators

The GTM AXIS includes two comparators for AXI IDs which are assigned to debugger accesses. When a debugger access is detected the *AEI_DEBUG_ACCESS* line is raised. For debugger accesses it is ensured, that any read access does not change the internal state of the GTM (e.g. read to a FIFO does not change the fill level).
Each comparator can be enabled individually and has a mask (bits to be compared must be 1) and a value setting. Following device configuration variables control the debugger ID comparators 0,1:

- AXIS_DEBUGGER_ID0_ENABLE: type integer; AXI Debugger ID comparator enable 0
- AXIS_DEBUGGER_ID0_MASK: type integer; AXI Debugger ID compare mask 0
- AXIS_DEBUGGER_ID0_VALUE: type integer; AXI Debugger ID compare value 0
- AXIS_DEBUGGER_ID1_ENABLE: type integer; AXI Debugger ID comparator enable 1
- AXIS_DEBUGGER_ID1_MASK: type integer; AXI Debugger ID compare mask 1
- AXIS_DEBUGGER_ID1_VALUE: type integer; AXI Debugger ID compare value 1

3.5.2.2.1.8 Error handling

If an unsupported AXI transaction is detected, this transaction in converted into an access to the unsupported address 0xFC on the AEI side to avoid any side-effects of the access. When the response to this transaction is available the error is reported on the AXI side.
This behavior is required due to the fully pipelined implementation of the adapter. Multiple outstanding AXI transactions are supported.
In case of an illegal AXI transaction (not defined by the AXI standard) the *AXIS_BUS_ERROR* signal is raised. On the AXI side the transaction is executed in the same way as an unsupported transaction.

3.5.2.2.1.9 64bit AXI support

The GTM supports 32bit or 64bit AXI data bus width configurations (selected via device configuration variable AXIS_DATA_WIDTH). Despite this, only 32bit accesses to the gtm_aei are supported. If the AXI data bus width is 64bit the gtm_axis "only" captures the 32-bit data word from the corresponding data lines (based on the address and control information as a result of having narrow transfers) and replicates the read data on the upper and the lower 32-bit data lines.

3.5.2.2.2 AXIS Ports

The block has an AXI slave interface on the "left" side and an AEI split master interface on the "right" side.
Table *3.18.4 "GTM toplevel AXIS interface "* describes the toplevel AXIS ports.

3.6 AEI Bus System

The AEI bus system in the GTM-IP allows any external component (CPU, DMA,..) in the SOC, where the GTM-IP is embedded, to communicate with the GTM internal modules. Communicate means the data can be written or read in the storage elements (registers, memory elements) in GTM.

Table 5 Definition of Terms Related to the AEI Bus System

Term	Definition
AEI bus system	Infrastructure in the GTM–IP which allows communication with any external or internal bus master.
Master	A component which is able to initiate bus transactions.
Slave	A component which is connected to the bus system and reacts on bus transactions.
Transaction	A transaction is a set of bus accesses which are initiated by a master. A transaction consists of one or more accesses. They can be distinguished as read and write transactions. All accesses in a transaction have the same type (read or write). A CPU typically issues only single access transactions, but a DMA typically issues multiple accesses (also known as bursts). They can be used to read/write addresses with a defined address increment.
Access	A bus access is a single read or write to an address on the bus system. In case of a write, the written data is transferred with the address and is stored to the target register/memory defined by the address. In case of a read, the value of the data available in the register/memory defined by the address, is sent back as read data to the bus master which initiated the access. Each access on the AEI bus system will additionally be qualified with an AEI_STATUS information, which is sent back to the bus master too.
AEI_STATUS	Information indicating the status of the bus access (see *2 "AEI Status Signal"*).
Wait state	An access might take several clock cycles until it terminates. The number of wait states defines how long an access is delayed until it terminates. The resolution is specified in number of bus clock cycles. A value of 0 means, the access terminates in the same clock cycle and the master can operate on the received data in the next clock cycle. The number of wait states depends also on the bus hierarchy. On module level there are typically 0 wait states for register accesses. On the cluster level, there is an increase of wait states due to the implemented pipeline stages and the cluster clock frequency. Each pipeline register increases the number of wait states by 1.
Access latency	The access latency defines how long it takes until an access on the AEI bus system is acknowledged on the GTM top level. The access latency of a single read access correlates to the number of wait states inserted. There are other factors beyond the wait states which can impact the overall access latency. E.g: operating frequency of the top level AEI/AXI interface or the operational mode of the GTM_AEI bus bridge.
Postponed transaction / postponed access	The GTM–IP is equipped with an access buffer in the GTM_AEI bus bridge. This allows that write accesses can be accepted in the access buffer and the access is acknowledged to the bus master immediately. In the background the operation will be performed. This has the benefit that multiple consecutive write accesses occupy the external bus system for a lesser amount of time and that the overall throughput on the external bus system is increased.
Split transaction / split access	The AXI protocol and the AEI split protocol support split accesses. A split protocol serves the access request independent from the access response. This contributes to a better overall throughput on the external bus system. The effectiveness of the split protocol correlates to the buffer size of the buffer in the GTM_AEI bridge. Whether the GTM supports split accesses or not depends on how it is integrated in the SOC. It should be written in the documentation of the SOC.

The AEI bus system is built out of multiple components visible in the block diagram. The next table gives an overview of the dedicated functions.

Table 6 Functionality of AEI Bus System Modules

Module	Functionality
GTM_AEI	Implements the top level bus bridge. It allows to operate the GTM AEI bus interface on an individual clock frequency. It supports 3 different AEI protocol versions which have different performance properties. The semiconductor vendor will choose a protocol variant which suits best for the integration of the GTM in the SOC. In the module a transaction buffer is implemented to store incoming transactions and support for postponed transactions.

Module	Functionality
GTM_AXIS	Enables the GTM to operate as a slave on the AXI bus protocol. Only available for devices which support AXIS communication (USE_AXIS = 1). It operates AEI bus master split protocol access to the GTM_AEI module.
GTM_AEM	Decodes the GTM internal address space into the cluster address ranges. Transactions will only be forwarded to the cluster which is addressed.
GTM_CLS_ARB	Performs arbitration of the 2 possible cluster AEI masters (MCS_AEIM, GTM_AEM). Arbitration will be performed on each single AEI access.
MCS_AEIM	Implements the bus bridge which allows the MCS to act as a master on the AEI bus of the local cluster. It is equipped with a transaction buffer to support the MCS channel pipelining and respond the requested data back to the MCS. Furthermore, it implements the interface to GTM external ADC functionality.
GTM_CLS_AEM	Decodes the cluster internal address space into the module address ranges. Transactions will only be forwarded to the module which is addressed.
AEI	Interface of a module which acts as AEI slave.
AEM	Interface of a module which acts as AEI master.

3.6.1 GTM_AEM Module

This module receives accesses from the GTM_AEI bridge and decodes (based on the provided address *AEI_ADDR* to which cluster i) the access has to be forwarded. The cluster index is decoded by i = *AEI_ADDR* [10:17]. In case the targeted cluster is switched off (**GTM_CLS_CLK_CFG.CLS[i]_CLK_DIV** = 0), the access is not forwarded and *AEI_STATUS* = b10 with *AEI_RDATA* = 0x0 is responded for this access. In case the targeted cluster i is not implemented in this device, the access is responded with *AEI_STATUS* = b11 and *AEI_RDATA* = 0x0.

3.6.2 GTM_CLS_ARB Module

This module located in cluster i receives single AEI accesses from 2 masters (GTM_AEM, MCS_AEIM). The arbitration will be performed based on following rules:

- In case **MCS_AEM_DIS.DIS_CLS[i]** = 0b11: MCS_AEIM accesses will never be arbitrated. The originating MCS channel will suspend until 0b01 is written to **MCS_AEM_DIS.DIS_CLS[i]** .
- In case 0b10 is written to **MCS_AEM_DIS.DIS_CLS[i]** while an MCS_AEIM access is ongoing, it will be aborted and only resumed when 0b01 is written to **MCS_AEM_DIS.DIS_CLS[i]** . In the meanwhile, the originating MCS channel will suspend. There is a risk of the access being executed twice, if the aborted access was to the PSM register **AFD[i]_CH[x]_BUF_ACC** . In this case the application has to ensure that this condition can never occur by means of semaphores or other methods which guarantee exclusive access.
- In case **MCS_AEM_DIS.DIS_CLS[i]** = 0b00, any MCS_AEIM access requests will be arbitrated with higher priority than the GTM_AEM ones. In case a GTM_AEM access is arbitrated while a new MCS_AEIM access is requested, the arbitration of the MCS_AEIM request will be delayed until the GTM_AEM access is serviced or aborted.
- In case GTM halt mode is requested by *GTM_HALT_REQ* = 1, any arbitrated MCS_AEIM access will be stopped immediately to ensure that GTM_AEM accesses will be possible while GTM halt mode is active (*GTM_HALT_ACTIVE* = 1). After leaving GTM halt mode (set *GTM_HALT_REQ* = 0), the restore phase starts (*GTM_RESTORE* = 1), which will restore the stopped MCS_AEIM access. This ensures that MCS bus accesses will be resumed correctly.

The GTM-IP is equipped with the possibility to introduce pipeline stages in the AEI bus system for accesses from the CPU. Details how to control the pipeline stage insertion can be found in the module integration guide. In big devices, the decoding of the combinatorial address space for the register accesses and the Rdata multiplexing will not be possible (due to limited performance of technology) in one clock cycle.

- AEI_ADDR_PIPELINE_STAGE: Pipeline registers are included for all AEI request signals in the AEI address decoding path at the input of GTM_CLS_ARB for the signals which are driven by the GTM_AEM. The signals (*AEI_SEL* , *AEI_ADDR* , *AEI_WDATA* , *AEI_W1R0* , *AEI_DEBUG_ACCESS*) will be delayed by 1 GTM clock (*CLK*).
- AEI_RDATA_PIPELINE_STAGE: Pipeline registers are included for all AEI response signals in the AEI access multiplexing path at the output of the GTM_CLS_ARB for the signals which are received by GTM_AEM. The signals (*AEI_RDATA* , *AEI_READY* , *AEI_STATUS*) will be delayed by 1 GTM clock (*CLK*). As a consequence, the latency of bus transaction from the CPU will increase in case pipeline registers are put in place.

3.6.3 GTM_CLS_AEM Module

This module receives accesses from the GTM_CLS_ARB module and decodes to which module the access has to be forwarded based on the provided address *AEI_ADDR* . The sub module decoding can be found in *10 "Overview of GTM Address Space"* . In case the targeted module is switched off (corresponding enable bit in **CCM[i]_CFG** is set to 0), the access is not forwarded and *AEI_STATUS* = b10 with *AEI_RDATA* = 0x0 is responded for this access. In case the targeted module is not implemented in this device, the access is responded with *AEI_STATUS* = b11 and *AEI_RDATA* = 0x0.

3.6.4 MCS_AEIM Module

The MCS_AEIM module receives read / write accesses on its slave interface bus from the MCS. Due to the MCS pipeline stages, consecutive accesses can be requested by the MCS on every cluster clock cycle. The MCS expects a response in the same clock cycle the request is issued indicating if the request of the access was accepted by MCS_AEIM and if it will be executed as a fast or long access.

The MCS_AEIM module is equipped with a transaction buffer. As long as the transaction buffer of the MCS_AEIM is not full, any access request will be accepted.

The transaction buffer depth is cluster dependent:

- Cluster 0: buffer_depth = 8
- Cluster 1..9: buffer_depth = 2

If the transaction buffer is full, the MCS access request is denied. In this case, the MCS channel has to repeat the same access when it is scheduled the next time. Typically, the MCS accesses will be served as fast accesses (0 wait states). A summary describing the situations for which an access must be signaled as long as access follows. For every fast access the MCS_AEIM will deliver the read data in the expected MCS pipeline stage. In case a long access is signaled the MCS must suspend the MCS channel and will resume operation as soon as the read data is available.

Under the following conditions a long access must be signaled:

- The target address of the access will respond with a delayed response. The following table lists all register and memory locations which respond with at least 1 cluster clock cycle wait state.
- The GTM_CLS_ARB has arbitrated a GTM_AEM access which will not terminate its response in the current cluster clock cycle. The ongoing GTM_AEM access will prevent the newly accepted MCS_AEIM access from being executed as fast access.
- As long as a long access was signaled, all following newly requested MCS_AEIM accesses have to be signaled as long accesses too. The earliest point in time an MCS_AEIM access request can be operational as fast access again will be if all transactions stored in the transaction buffer have been executed and the response was consumed by the MCS.
- The GTM_CLS_ARB is configured to **MCS_AEM_DIS.DIS_CLS[i]** = 0b11. This will prevent the execution of any accesses of the MCS_AEIM.

Table 7 List of Register / Memory which will Respond with Number of Wait States Greater Than 0

Register	Memory	Number of wait states (READ)	Number of wait states (WRITE)
GTM_RST		0	1
GTM_CLS_CLK_CFG		0	1
TIM[i]_RST		0	1
BRC_RST		0	1
DPLL_CTRL_1		0	1
AFD[i]_CH[x]_BUF_ACC		3	0
TOM[i]_TGC[g]_GLB_CTRL		0	1
ATOM[i]_AGC_GLB_CTRL		0	1
MCS[i]_CTRG		0	>= 1
MCS[i]_STRG		0	>= 1
	FIFO_MEMORY	3	0
DPLL_PSA[n] ... **DPLL_DTA[n]**	DPLL_RAM1A	>= 4	>= 3
DPLL_TS_T ... **DPLL_DT_S[p]**	DPLL_RAM1B/C	>= 4	>= 3
DPLL_RDT_T[p] ... **DPLL_DT_T[p]**	DPLL_RR2	>= 4	>= 3
	MCS[i]_MEM	>= 3	>= 0

3.6.4.1 ADC Interface Decoding

In case the targeted address is in the address range of the ADC interface, the access will be executed on the ADC interface signals of this cluster. The ADC interface will respond with the read data in the next cluster clock cycle in any case. The read data will be stored in the AEIM

transaction buffer. Any access to the ADC address range will be executed with 0 wait states. Typically, each ADC access will be signaled as fast access.

Previous MCS accesses, which have been responded as long accesses will lead to a long access signaling for ADC accesses too until all transactions stored in the transaction buffer have been executed and the data was read by the MCS module.

It has to be mentioned, that the MCS access response ordering will be maintained by the transaction buffer. It is possible that due to an arbitrated GTM_AEM access with high wait states, a MCS_AEIM access to the GTM_CLS_ARB in the transaction buffer is delayed and executed later than an ADC access on the ADC interface.

3.7 GTM_CTL Module

3.7.1 AEI Bus System Transaction Status Monitoring for Transactions Applied to the GTM−IP

GTM_CTL allows monitoring of transactions initiated from GTM−IP external bus masters (CPU, DMA,..). Monitoring takes place on the signals after the GTM_AEI bridge and operates on the *CLK* clock.

Each transaction response on the AEI bus system will indicate (with the *AEI_STATUS* signal) the current state of execution of the transaction (see *2 "AEI Status Signal"*). To allow an application software to react on unexpected state of transactions (*AEI_STATUS* != 0b00), a transaction monitoring and signaling with an interrupt on *GTM_AEI_IRQ* is implemented.

For monitoring the first transaction which occurs with *AEI_STATUS* != 0, GTM_CTL stores the transaction type in **GTM_AEI_STA_XPT.W1R0** and the transaction address in **GTM_AEI_STA_XPT.ADDR** . Following transactions with (*AEI_STATUS* != 0b00) will not change the content of **GTM_AEI_STA_XPT** . Furthermore, for each type of *AEI_STATUS* an individual interrupt unit (see **GTM_IRQ_NOTIFY.AEI_USP_ADDR** , **GTM_IRQ_NOTIFY.AEI_IM_ADDR** , and **GTM_IRQ_NOTIFY.AEI_USP_BE**) is implemented. This allows applications to individually select for which kind of transaction response exception, an interrupt *GTM_AEI_IRQ* (see **GTM_IRQ_EN.AEI_USP_ADDR_IRQ_EN** , **GTM_IRQ_EN.AEI_IM_ADDR_IRQ_EN** , and **GTM_IRQ_EN.AEI_USP_BE_IRQ_EN**) or/and error interrupt *GTM_ERR_IRQ* (**GTM_EIRQ_EN.AEI_USP_ADDR_EIRQ_EN** , **GTM_EIRQ_EN.AEI_IM_ADDR_EIRQ_EN** , and **GTM_EIRQ_EN.AEI_USP_BE_EIRQ_EN**) can be generated.

Any write access of value 1 to one of the flags **GTM_IRQ_NOTIFY.AEI_USP_ADDR** , **GTM_IRQ_NOTIFY.AEI_USP_BE** , **GTM_IRQ_NOTIFY.AEI_IM_ADDR** will clear the flags and the transaction status exception register **GTM_AEI_STA_XPT** .

3.7.2 Additional Transaction Status Monitoring in a Device With an A-XI Slave

The AXI slave is able to detect invalid transactions which cannot be operated correctly. In these cases, the AXIS module initiates an error signaling read transaction to the GTM_AEI. The address in use will be 0xFC. This address is an unimplemented address in the GTM_CTL. Any transaction to this address can trigger the *GTM_AEI_IRQ* interrupt or store the exception in **GTM_AEI_STA_XPT** .

3.7.3 AEI Bus System Transaction Status Monitoring for Transactions Initiated by MCS Modules

GTM_CTK allows monitoring of transactions initiated by any MCS bus master. Monitoring takes place on the signals between the MCS_AEIM[i] and the MCS[i] module in each cluster i. This monitoring operates on the *CLS[i]_CLK* clock.

Each MCS transaction response on the cluster MCS_AEIM bus system will indicate (with the *CCM[i]_AEI_STATUS*) signal the current state of execution of the transaction (see *2 "AEI Status Signal"*). To allow an application software to react on unexpected state of transactions (*CCM[i]_AEI_STATUS* != 0b00) a transaction monitoring and signaling with an interrupt on *GTM_AEI_IRQ* is implemented.

For monitoring the first transaction which occurs with *CCM[i]_AEI_STATUS* != 0, GTM_CTL stores the transaction status in **CCM[i]_AEIM_STA.AEIM_XPT_STA** and the transaction address in **CCM[i]_AEIM_STA.AEIM_XPT_ADDR** . Additionally, in the bitfield **GTM_CTRL.AEIM_CLUSTER** the number of the cluster which caused the transaction exception gets stored. Following transactions with (*CCM[i]_AEI_STATUS* != 0b00) will not change the content of **GTM_CTRL.AEIM_CLUSTER** .

Following transactions in cluster i with (*CCM[i]_AEI_STATUS* != 0b00) will not change the content of **CCM[i]_AEIM_STA** . Furthermore, for each type of *CCM[i]_AEI_STATUS* an individual interrupt unit (see **GTM_IRQ_NOTIFY.AEIM_USP_ADDR** , **GTM_IRQ_NOTIFY.AEIM_IM_ADDR** , and **GTM_IRQ_NOTIFY.AEIM_USP_BE**) is implemented. This allows applications to individually select for which kind of transaction response exception an interrupt *GTM_AEI_IRQ* (see **GTM_IRQ_EN.AEIM_USP_ADDR_IRQ_EN** , **GTM_IRQ_EN.AEIM_IM_ADDR_IRQ_EN** , and **GTM_IRQ_EN.AEIM_USP_BE_IRQ_EN**) or/and error interrupt *GTM_ERR_IRQ* (see **GTM_EIRQ_EN.AEIM_USP_ADDR_EIRQ_EN** , **GTM_EIRQ_EN.AEIM_IM_ADDR_EIRQ_EN** , and **GTM_EIRQ_EN.AEIM_USP_BE_EIRQ_EN**) can be generated.

Clearing of MCS initiated bus exceptions has to be done in two steps:

1. Any write access of value 1 to one of the flags **GTM_IRQ_NOTIFY.AEIM_USP_ADDR** , **GTM_IRQ_NOTIFY.AEIM_USP_BE** , **GTM_IRQ_NOTIFY.AEIM_IM_ADDR** will clear the flags and the cluster number in the MCS transaction status exception register **GTM_CTRL.AEIM_CLUSTER** and reenable the monitoring of MCS initiated bus exceptions for all clusters.
2. Any write access to **CCM[i]_AEIM_STA** will clear the register and reenable the monitoring of new MCS initiated bus transaction exceptions in this cluster i.

Any MCS initiated bus transaction exception can stop the execution of the MCS program code, for further details see last paragraph of *17.5 "AEI Bus Master Interface"* .

3.7.4 AEI Bus System Transaction Length Monitoring

GTM_CTL allows monitoring of transactions initiated from GTM-IP external bus masters (CPU, DMA,..). The monitoring takes place on the signals *AEI_SEL* , *AEI_READY* of the GTM_AEI bridge master, the monitoring operates on the *CLK* clock. The transaction length monitoring can be enabled by writing **GTM_CTRL.TO_VAL** to a value unequal to 0. For each transaction the length will be counted in clock resolutions. The length timeout counter will increment on *CLK* as long as *AEI_SEL* = 1 and *AEI_READY* = 0. Each end of a transaction will reset the timeout counter. As soon as the length timeout counter is equal to **GTM_CTRL.TO_VAL** , one of the 4 possible reactions configurable with **GTM_CTRL.TO_MODE** will be triggered:

- b00: Observe once. On the first trigger, store the transaction information in **GTM_AEI_ADDR_XPT** . There is no impact on the ongoing transaction. Further triggers will be ignored.
- b01: Abort once. On the first trigger, store the transaction information in **GTM_AEI_ADDR_XPT** and immediately abort the ongoing transaction. The GTM_AEI bridge master will abort the ongoing transaction by *AEI_SEL* = 0. In the transaction buffer, the access is terminated and the data and status are assumed to be 0. The subsequent triggers will not impact the transactions. Further triggers will be ignored.
- b10: Retry. On the first trigger, store the transaction information in **GTM_AEI_ADDR_XPT** and immediately stop and repeat the ongoing transaction. The GTM_AEI bridge master will stop the ongoing transaction by applying *AEI_SEL* = 0 for 1 clock cycle, afterwards with *AEI_SEL* = 1 the transaction is continued. This will be applied for every occurring trigger.
- b11: Abort. On the first trigger, store the transaction information in **GTM_AEI_ADDR_XPT** and immediately abort the ongoing transaction. The GTM_AEI bridge master will abort the ongoing transaction by *AEI_SEL* = 0. In the transaction buffer, the access is terminated and the data and status are assumed to be 0. This will be applied for every occurring trigger.

The first occurring trigger will store the transaction address *AEI_ADDR* in **GTM_AEI_ADDR_XPT.TO_ADDR** and the transaction type *AEI_W1R0* in **GTM_AEI_ADDR_XPT.TO_W1R0** . Following timeout triggers will not change the content of **GTM_AEI_ADDR_XPT** . Furthermore, an individual interrupt unit (see *3.12 "GTM-IP Interrupt Concept"*) is implemented. This allows applications to receive an interrupt based on the transaction access length. The interrupt *GTM_AEI_IRQ* or/and error interrupt *GTM_ERR_IRQ* can be generated.

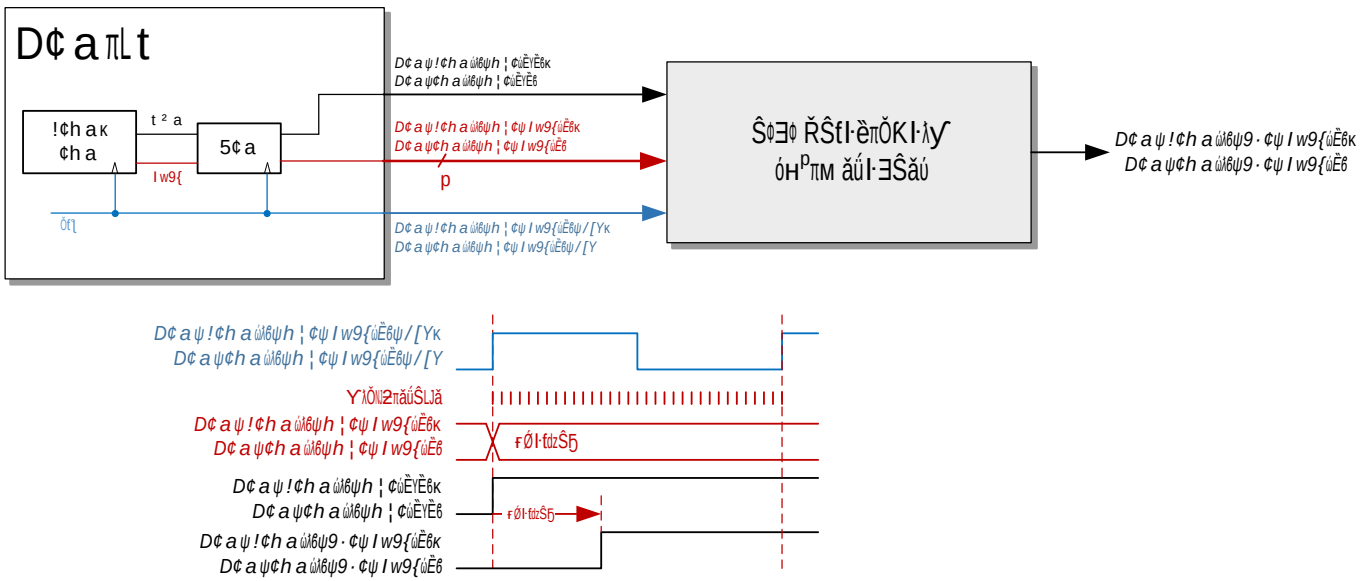
Any write access of value 1 to the flag **GTM_IRQ_NOTIFY.AEI_TO_XPT** will clear the flag and the transaction length monitoring register **GTM_AEI_ADDR_XPT** . This will reenable the timeout trigger generation.

3.8 High Resolution PWM Support

To realize a high resolution PWM a suitable circuit is needed, e.g. a delay chain, which additionally delays the PWM output signal by a defined number of micro-steps.

This circuit is not part of the GTM-IP, but to control this circuit, GTM-IP delivers output ports *GTM_TOM[i]_OUT_HRES[x]* and *GTM_ATOM[i]_OUT_HRES[x]* in addition to the PWM output ports *GTM_TOM[i]_OUT* [x:x] and *GTM_ATOM[i]_OUT* [x:x].

Figure 12 High Resolution PWM Support



The width of *GTM_TOM[i]_OUT_HRES[x]* and *GTM_ATOM[i]_OUT_HRES[x]* is defined to 5. Therefore, the number of micro-steps is $2^5 - 1 = 31$.

The width of one micro-step depends on the clock period: $\text{micro-step}_{\text{width}} = CLK_{\text{period}} / 2^5$

A clock period of 5 ns (200 MHz) results in a micro-step width of: 5 ns / 2^5 = 156.25 ps.

The additional output signal *GTM_ATOM[i]_OUT_HRES[x]* and *GTM_TOM[i]_OUT_HRES[x]* will be generated in TOM and ATOM submodules beside the PWM output signal *GTM_TOM[i]_OUT* [x:x] and *GTM_ATOM[i]_OUT* [x:x]. Afterwards the signal is connected through the DTM submodule before being directed outside of GTM-IP.

Furthermore, the internal clock signal (clk) with which the GTM-IP output signals are generated will be delivered by GTM-IP.

The DTM submodule provides the possibility to input dead time on the output signal and the inverse output signal. Based on the high resolution feature it is possible to adjust the dead time length on micro-step level.

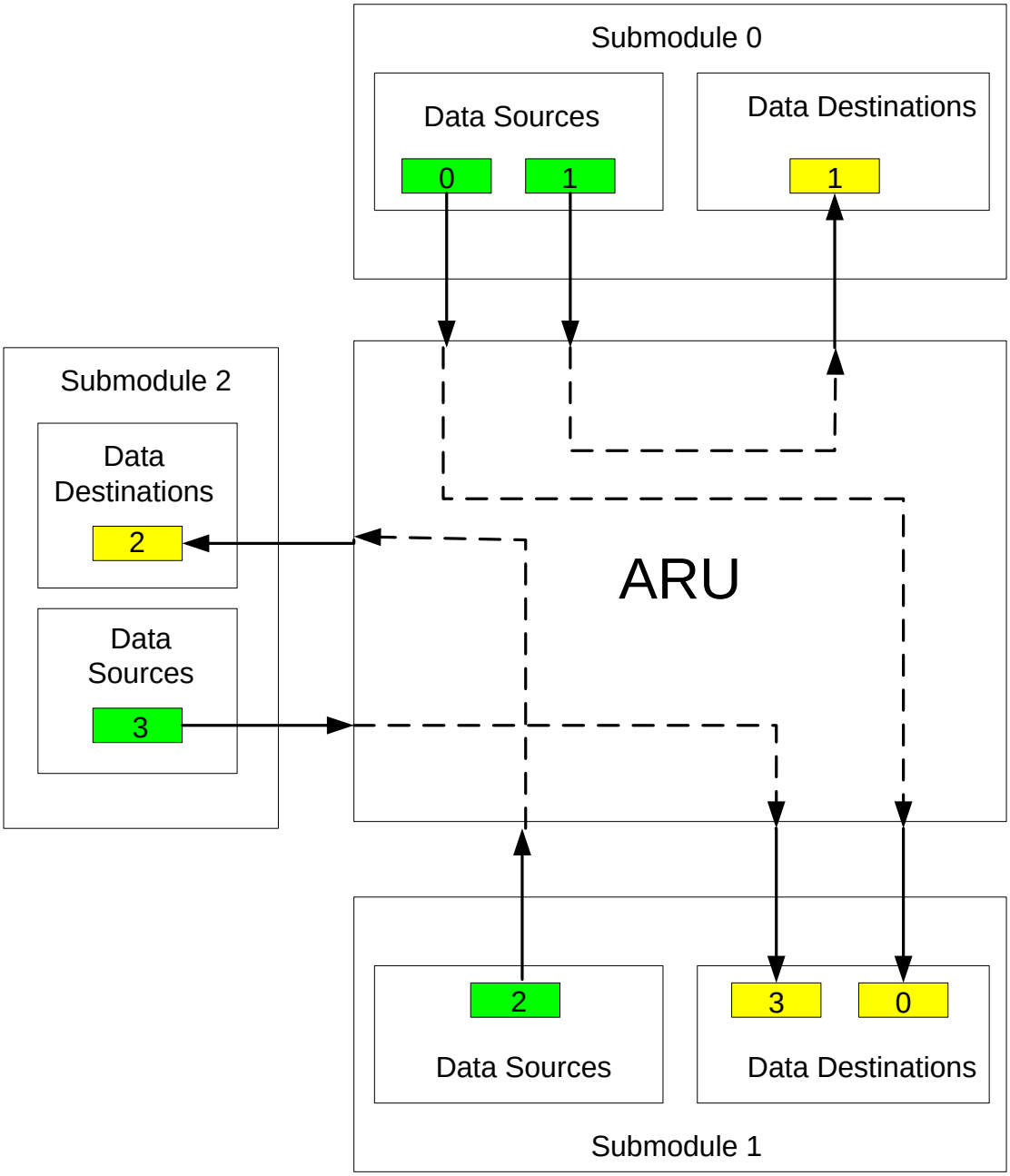
For detailed information about the high resolution PWM support please refer to the chapter of the submodules TOM, ATOM and DTM.

3.9 ARU Routing Concept

One central concept of the GTM-IP is the routing mechanism of the ARU submodule for data streams. Each data word transferred between the ARU and its connected submodule is 53 bit wide. It is important to understand this concept in order to use the resources of the GTM-IP effectively. Each module that is connected to the ARU may provide an arbitrary number of the read and write channels of the ARU. In the following, the ARU write channels are named data sources and the ARU read channels are named data destinations.

The concept of the ARU intends to provide a flexible and resource efficient way for connecting any data source to an arbitrary data destination. In order to save resource costs, the ARU does not implement a switch matrix, but it implements a data router with serialized connectivity providing the same interconnection flexibility. Figure *13 "Principle of Data Routing using ARU"* shows the ARU data routing principle. Data sources are marked with a green rectangle and the data destinations are marked with yellow rectangles. The dashed lines in the ARU depict the configurable connections between data sources and data destinations. A connection between a data source and a data destination is also called a data stream.

Figure 13 Principle of Data Routing using ARU



The data streams are configured in the following manner. Each data source has its fixed and unique source address: The ARU write address. The fixed address of each data source is pointed out by the numbers in the green boxes of figure *13 "Principle of Data Routing using ARU"* . The address definitions of all available data sources in the GTM-IP can be obtained from table *93 "ARU Write Address Overview"* . The connection from a specific data source to a specific data destination is defined by configuring the corresponding address of a data source (i.e. the ARU write address) in the desired data destination. The configured address of each data destination is pointed out by the numbers in the yellow boxes of figure *13 "Principle of Data Routing using ARU"* .

Normally, the destination is idle and waits for data from the source. If the source offers new data, the destination does a destructive read, processes the data and goes idle again. The same data is never read twice.

There is one submodule for which this destructive read access does not hold. This is the BRC submodule configured in Maximal Throughput Mode. For a detailed description of this module please refer to chapter *6 "Broadcast Module (BRC)"* .

The functionality of the ARU is as follows: The ARU sequentially polls the data destinations of the connected modules in a Round Robin order. If a data destination requests new data from its configured data source and the data source has data available, the ARU delivers the data to the destination and it informs both, the data source and destination that the data is transferred. The data source marks the delivered ARU data as invalid which means that the destination consumed the data.

It should be noted that each data source should be connected to a single data destination only because the destinations consume the data. If two destinations would reference the same source one destination would consume the data before the other destination could consume it. Since the data transfers are blocking, the second destination would block until it receives new data from the source. If a data source should be connected to more than one data destination the submodule Broadcast (BRC) has to be used. On the other hand, the transfer from a

data source to the ARU is also blocking, which means that the source channel can only provide new data to the ARU when an old data word is consumed by a destination. In order to speed up the process of data transfers, the ARU handles two different data destinations in parallel.

Following table gives an overview about the number of data destinations and data sources of each GTM-IP instance type.

Table 8 ARU Source and Destination Address Count per Instance

Sub-module	Number of data sources per instance	Number of data destinations per instance
ARU	1	0
DPLL	NOAC	NOAC
TIM	8	0
MCS	24	8
BRC	22	12
TOM	0	0
ATOM	8	8
DTM	0	0
PSM	8	8
ICM	0	0
CMP	0	0
MON	0	0
CCM	0	0

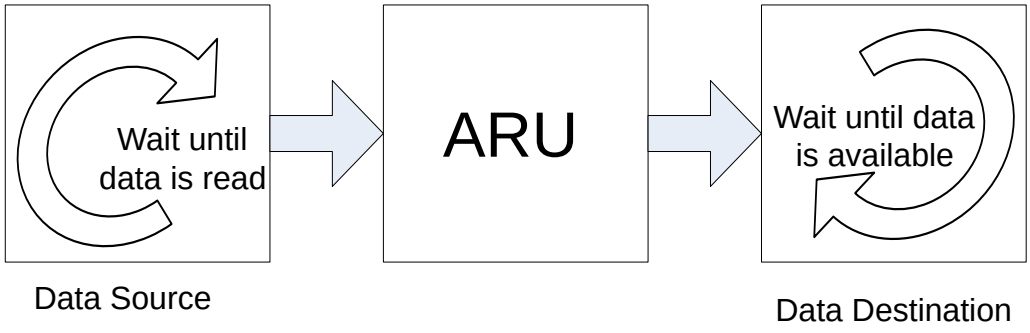
3.9.1 ARU Round Trip Time

The ARU uses a Round Robin arbitration scheme with a fixed round trip time for all connected data destinations. This means, the time between two adjacent read requests resulting from a data destination channel always takes the round trip time, irrespective of the success or failure of the read request.

3.9.2 ARU Blocking Mechanism

Another important concept of the ARU is its blocking mechanism that is implemented for transferring data from a data source to a data destination. This mechanism is used by ARU connected submodules to synchronize the submodules to the routed data streams. Figure *14 "Graphical Representation of ARU Blocking Mechanism"* explains the blocking mechanism.

Figure 14 Graphical Representation of ARU Blocking Mechanism



If a data destination requests data from a data source over the ARU but the data source does not have any data yet, it has to wait until the data source provides new data. In this case, the submodule that owns the data destination may perform other tasks. When a data source produces new data faster than a data destination can consume the data the source raises an error interrupt and signals that the data could not be delivered in time. The new data is marked as valid for further transfers and the old data is overwritten.

In any case the round trip time for the ARU has a fixed reset value for a specific device configuration. The end value of the roundtrip counter can be changed with a configuration register **ARU_CADDR_END** inside the ARU. For more details see the ARU specific chapter.

It is possible to reset the ARU roundtrip counter **ARU_CADDR** manually synchronous to CMU clock enable from configuration register inside CMU module. Please refer to CMU specific chapter for more details.

One exception is the BRC submodule when configured in Maximal Throughput Mode. Please refer to chapter *6 "Broadcast Module (BRC)"* for a detailed description.

3.9.3 ARU Cluster Isolation

The configuration register **GTM_ARU_COM_DIS** allows to disable for each cluster k all kind of ARU transfers listed next:

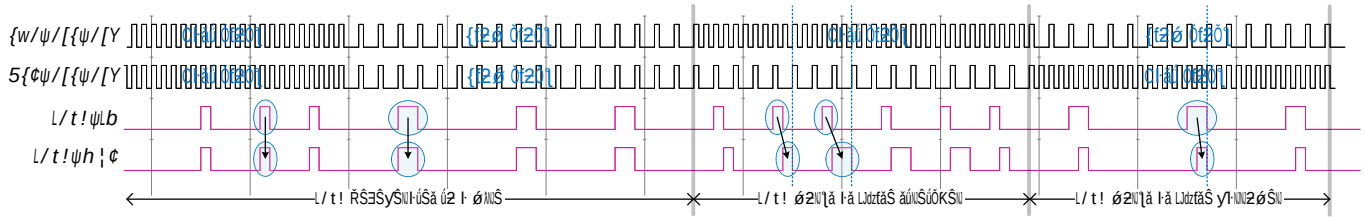
- If a data destination located in cluster k requests data from a data source located outside the cluster k.
- If a data destination outside of cluster k requests data from a data source located in cluster k.
- If a data destination located in cluster k requests data from the **ARU_ACCESS** source in the module ARU.
- If the **ARU_ACCESS** destination in the module ARU requests data from a data source located in cluster k.

In all above mentioned cases, on a disabled ARU transfer the behavior is as follows:
When a data destination requests new data from its configured data source, the data source reports that no valid data are available. No data will be delivered to the target. The data source will never deliver ARU data (even if it is available) and the data are never consumed in the data source.

3.10 Inter–Cluster Pulse Adapter

Clusters can be configured to use different clock frequencies. For that, an additional control scheme (called here inter–cluster pulse adapter (ICPA)) is needed to coordinate the exchange of signals between two clusters. The concept of ICPA is demonstrated in *15 "Demonstration of the ICPA Behavior Handling Pulses at Different Source and Destination Cluster Frequencies"* .

Figure 15 Demonstration of the ICPA Behavior Handling Pulses at Different Source and Destination Cluster Frequencies



ICPA attempts to transform a pulse (of one cycle duration) generated in the source cluster in order to make it sampled exactly once in the destination cluster (regardless of the clocks used in both).

When the clocks in the two clusters are identical, the ICPA degenerates into a wire.

When a pulse is communicated from a cluster running at a slower clock rate to a faster clock rate, the ICPA narrows the pulse appropriately so that it is sampled once in the destination cluster.

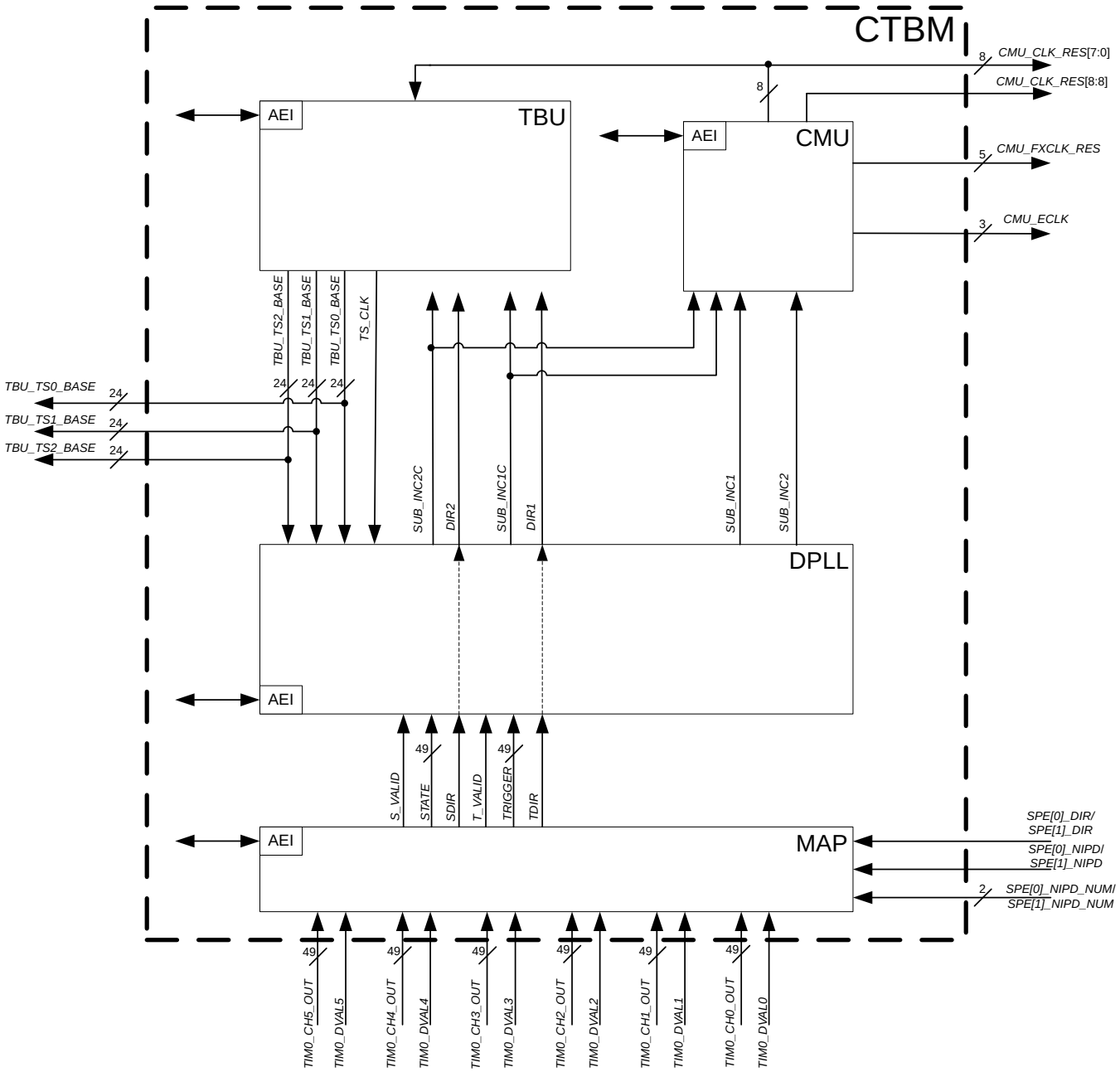
When a pulse is communicated from a cluster running at a faster clock rate to a cluster running at a slower clock rate, the ICPA stretches it appropriately so that it is sampled once in the destination cluster.

It is worth mentioning that in the latter case, some pulses might get missed. This happens for example when the rate of pulse generation at the source cluster is higher than the clock rate in the destination cluster.

3.11 GTM–IP Clock and Time Base Management (CTBM)

Inside the GTM–IP, several submodules are involved in the clock and time base management of the whole GTM. Figure *16 "GTM–IP Clock and time base management architecture"* shows the connections and submodules involved in these tasks. The submodules involved are called Clock and Time Base Management (CTBM) modules further on.

Figure 16 GTM–IP Clock and time base management architecture



One important module of the CTBM is the Clock Management Unit (CMU) which generates up to 14 clock resolutions for the submodules of the GTM and up to three GTM external clocks. The clock resolutions are *CMU_CLK_RES* [x:x] and *CMU_FXCLK_RES* [y:y], where x ∈ {0, 1, ..., 8} and y ∈ {0, 1, ..., 4}. These signals are routed to all clusters via ICPAs. Within a cluster, they are used as clock enables regulating the resolution (or rate) at which actions are performed. The external clocks are *CMU_ECLK* [z:z], where z ∈ {0, 1, 2}. In contrast to *CMU_CLK_RES* [x:x], these are approximately 50% duty cycle clocks. For a detailed description of the CMU functionality and clocks please refer to chapter *10 "Clock Management Unit (CMU)"* .

The five *CMU_FXCLK* [y:y] (y ∈ {0, 1, ..., 4}) clock resolutions are used by the TOM submodule for PWM generation.

A maximum of nine *CMU_CLK_RES* [x:x] (x∈{0, 1,..., 8}) clocks are used by other submodules of the GTM for signal generation.

Inside the Time Base Unit (TBU) one of *CMU_CLK_RES* [x:x] (x ∈ {0, 1, ..., 7}) clock resolutions is used per channel to generate a common time base for the GTM. Besides the *CMU_CLK_RES* [x:x] signals, the TBU can use the compensated *SUB_INC1C* and *SUB_INC2C* signals coming from the DPLL submodule for time base generation. This time base typically represents an angle clock for an engine management system. For more details about compensated (*SUB_INC1C* and *SUB_INC2C*) and uncompensated (*SUB_INC1* and *SUB_INC2*) DPLL signals please refer to the DPLL chapter *21 "Digital PLL Module (DPLL)"* . The *SUB_INC1C* and *SUB_INC2C* signals in conjunction with the two direction signal lines *DIR* control the TBU time base. The TBU functionality is described in *12 "Time Base Unit (TBU)"* .

The TBU submodule generates the three time base signals *TBU_TS0_BASE* , *TBU_TS1_BASE* and *TBU_TS2_BASE* which are widely used inside the GTM as common time bases for signal characterization and generation.

Besides the time base 1 and 2 which may represent a relative angle clock for an engine management system it is helpful to have an absolute angle clock for CPU/MCS internal angle algorithm calculations. This absolute angle clock is represented by the TBU base 3. The TBU

channel 0 up to 2 are widely used inside the GTM as common time (channel 0, 1 and/or 2) or angle (channel 1 and/or 2) bases for signal characterization and generation. The TBU channel 3 is only configurable and readable by MCS0 or CPU.

As stated before, the DPLL submodule provides the four clock signals *SUB_INC1* and *SUB_INC2* and *SUB_INC1C* and *SUB_INC2C* which can be seen as a clock multiplier generated out of the two input signal vectors *TRIGGER* and *STATE* coming from the MAP submodule. For a detailed description of the DPLL functionality please refer to chapter *21 "Digital PLL Module (DPLL)"* .

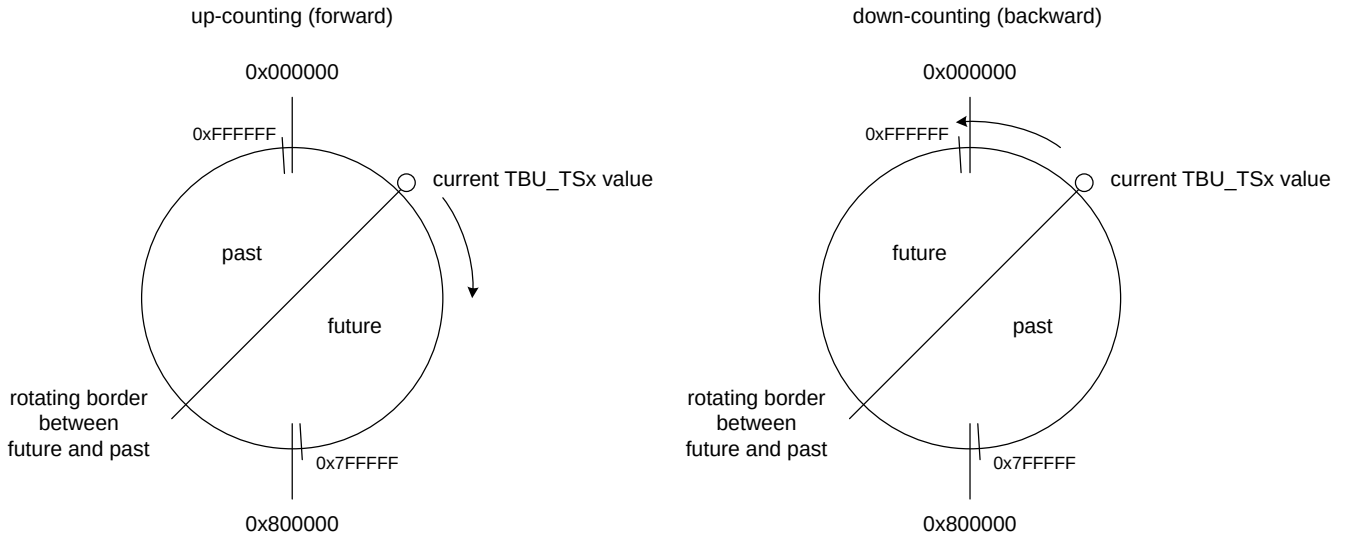
The MAP submodule is used to select the *TRIGGER* and *STATE* signals for the DPLL out of six input signals coming from TIM0 submodule. Besides this, the MAP submodule is able to generate a *TDIR* (TRIGGER Direction) and *SDIR* (STATE Direction) signal for the DPLL and TBU coming from the SPE0 and SPE1 signal lines. The direction signals are generated out of a defined input pattern. For a detailed description of the MAP submodule please refer to section *20 "TIM0 Input Mapping Module (MAP)"* .

3.11.1 Cyclic Event Compare

With the time base module (TBU) the GTM provides three counters, where the counter of TBU channel 0 represents a time and the counter of TBU channel 1 or 2 may represent a time (if clock source is *CMU_CLK_RES* generated inside CMU) or an angle (if clock source is a DPLL *SUB_INC* signal provided via CMU).

Deciding if an event is to happen in the future (or has happened in the past) is vital in many applications. Using conventional relational operators to determine if an event has happened in the past (or is to happen in the future) is not reliable. The reason is attributed to the inevitable wrap-arounds associated with using limited word length representations for the time bases. In order to avoid this, cyclic event compares have been introduced in GTM (see Figure *17 "Cyclic Event Counter Representing Time or Angle"*). The dynamic range of tbu ∈ { *TBU_TS0_BASE* , *TBU_TS1_BASE* , *TBU_TS2_BASE* } is represented as a circle with a diameter starting from the current value of tbu and splitting the circle into two regions (future and past). The two regions rotate with tbu (clockwise when up-counting and counter-clockwise when down-counting). Depending on the direction of counting and the time/position associated with an event, it can be readily determined if an event has happened in the past or is to happen in the future. The definitions for the cyclic compare operators are provided in the following algorithm:

Figure 17 Cyclic Event Counter Representing Time or Angle



The two points on the intersection between the circumference and the diameter belong to the 'past' regardless of the time counting direction.

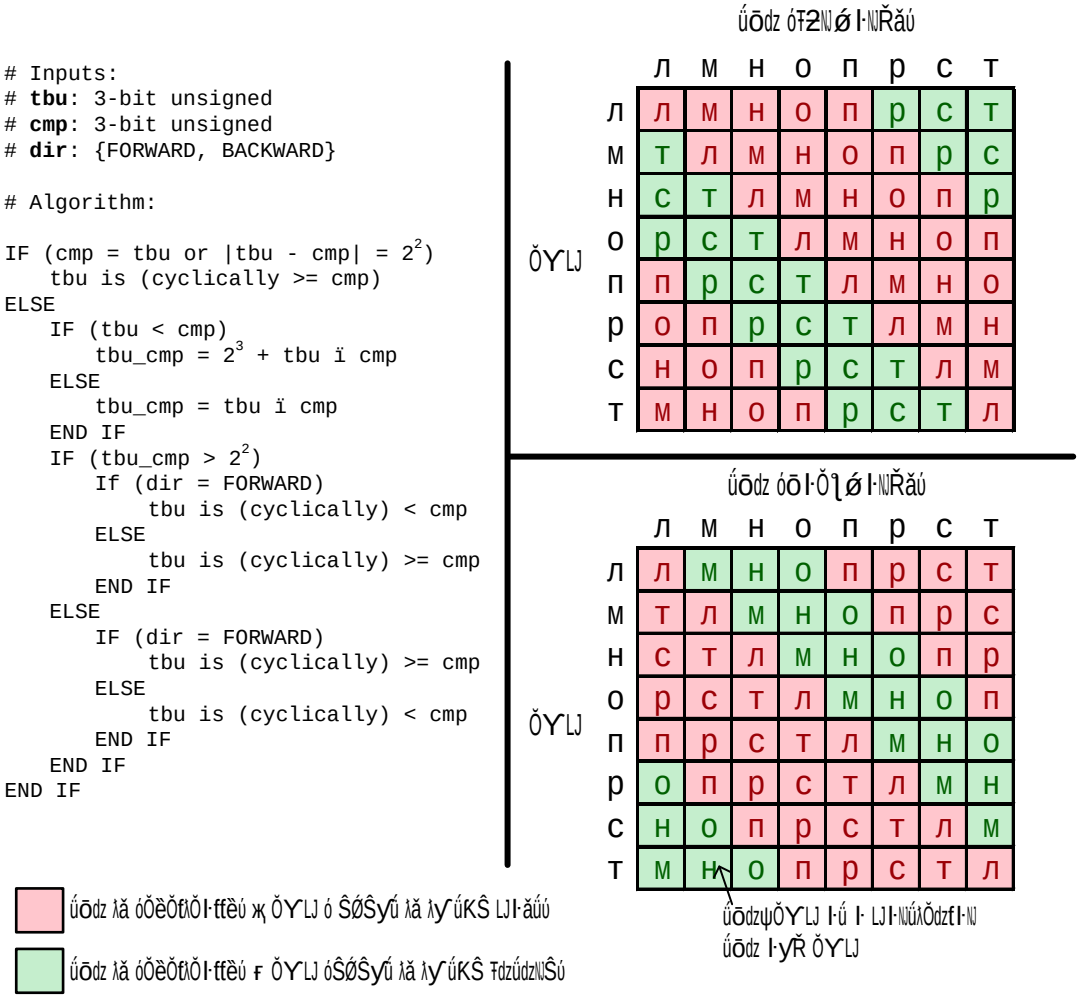
Inputs: tbu: 24-bit unsigned (time/angle value of the timebase); cmp: 24-bit unsigned (the time/position associated with an event); dir ∈ {FORWARD, BACKWARD}.
If (cmp = tbu or |tbu − cmp| = 2^{23})
tbu is (cyclically) >= cmp
Else
IF (tbu < cmp)
tbu_cmp = 2^{24} + tbu - cmp
ELSE
tbu_cmp = tbu - cmp
END IF
IF (tbu_cmp > 2^{23})
If (dir = FORWARD)
tbu is (cyclically) < cmp
Else
tbu is (cyclically) >= cmp
END IF
ELSE
If (dir = FORWARD)
tbu is (cyclically) >= cmp
Else
tbu is (cyclically) < cmp

END IF
END IF
END IF

An event is assumed to have happened in the past if tbu is 'cyclically greater than or equal to' the time associated with the event. Otherwise, the event is assumed to happen in the future.

Figure *18 "Demonstration of Cyclic Event Compares"* demonstrates how cyclic event compares in conjunction with the counting direction of tbu can be used to identify if an event has happened in the past or is to happen in the future. In the figure, both tbu and cmp are assumed to be 3-bit unsigned.

Figure 18 Demonstration of Cyclic Event Compares



By adapting the cyclic event compares and taking the counting direction of tbu (up-counting or down-counting) into account, it is possible to identify if an event is to happen in the future or has happened in the past reliably as long as |tbu-cmp| < 0x800000. Having a difference larger than that should be avoided as it erroneously pushes an event from 'past' to 'future' (or the other way around) without crossing 'now' (the current time stamp on tbu). For example, if an event is to be scheduled in the far future, cmp should be carefully selected such that the difference does not exceed 0x7FFFFF. Having a difference larger than that will result in classifying the associated event as 'has happened in the past'.

3.12 GTM-IP Interrupt Concept

The submodules of the GTM-IP can generate thousands of interrupts on behalf of internal events. This high number of interrupts is combined inside the Interrupt Concentrator Module (ICM) into interrupt groups. In these interrupt groups the GTM-IP submodule interrupt signals are bundled to a smaller set of interrupts. From these interrupt sets, a smaller number of interrupt signals is created and signaled outside of the GTM-IP on a signal with the name "GTM_<MOD>[i]_IRQ", where <MOD>[i] identifies the name of the corresponding GTM-IP submodule.

In the following subchapters the interrupt signals are described on the basis of the TIM instance i. TIM[i] in the name of the signals can be replaced by the name of the other submodules of the GTM-IP and are valid for all following modules.

- PSM[i]
- MCS[i]
- TOM[i]
- ATOM[i]

- TIO[i]_G[g]
- SPE[i]
- ARU
- DPLL
- BRC
- CMP

Moreover, each output signal *GTM_TIM[i]_IRQ* has a corresponding input signal *GTM_TIM[i]_IRQ_CLR* that can be used for clearing the interrupts. These input signals can be used by the surrounding microcontroller system as:

- Acknowledge signal from a DMA controller
- Validation signal from ADC
- Clear signal from a GTM-external interrupt controller to do an atomic clear while entering an ISR routine

The individual interrupts are controlled inside the submodules. If a submodule consists of several submodule channels that are most likely to work independent from each other (like TIM, PSM, MCS, TOM, ATOM, TIO), each submodule channel has its own interrupt control and status register set, named as interrupt set IRQ_SET= <MOD>[i]_CH[x:x] in the following. Other submodules (SPE, ARU, DPLL, BRC, CMP and global GTM functionality) have a common interrupt set IRQ_SET= <MOD>[i] for the whole submodule.

In the following subchapters the interrupt set registers are described on the basis of the TIM module. TIM[i]_CH[x] can be replaced in the names of the registers for all modules of the GTM-IP and are valid for all modules. TIM[i]_CH[x] can be replaced by the following IRQ_SET:

- FIFO[i]_CH[x]
- MCS[i]_CH[x]
- MCS[i]_SINT
- MCS[i]_HBP[h]
- TOM[i]_CH[x]
- ATOM[i]_CH[x]
- TIO[i]_G[g]_CH[c]
- SPE[i]
- ARU
- DPLL
- BRC
- CMP
- GTM

The interrupt set consists of four registers: The **TIM[i]_CH[x]_IRQ_EN** register, the **TIM[i]_CH[x]_IRQ_NOTIFY** register, the **TIM[i]_CH[x]_IRQ_FORCINT** register, and the **TIM[i]_CH[x]_IRQ_MODE** register. While the registers **TIM[i]_CH[x]_IRQ_EN** , **TIM[i]_CH[x]_IRQ_NOTIFY** , and **TIM[i]_CH[x]_IRQ_FORCINT** signalize the status and allow controlling of each individual interrupt source within an interrupt set, the register **TIM[i]_CH[x]_IRQ_MODE** configures the interrupt mode that is applied to all interrupts that belong to the same interrupt set.

In order to support a wide variety of microcontroller architectures and interrupt systems with different interrupt signal output characteristics and internal interrupt handling the following four modes can be configured:

- Level mode
- Pulse mode
- Pulse-Notify mode
- Single-Pulse mode

These interrupt modes are described in more details in the following subsections.

The register **TIM[i]_CH[x]_IRQ_EN** allows the enabling and disabling of an individual interrupt within an interrupt set. Independent of the configured mode, only enabled interrupts can signalize an interrupts on its signal *GTM_TIM[i]_IRQ* .

The register **TIM[i]_CH[x]_IRQ_NOTIFY** collects the occurrence of interrupt events. The behavior for setting a bit in this register depends on the configured mode and thus it is described later on in the mode descriptions.

Independent of the configured mode any write access with value '1' to a bit in the register **TIM[i]_CH[x]_IRQ_NOTIFY** always clears the corresponding **TIM[i]_CH[x]_IRQ_NOTIFY** bit.

Moreover, the enabling of a disabled interrupt source with a write access to the register **TIM[i]_CH[x]_IRQ_EN** also clears the corresponding bit in the **TIM[i]_CH[x]_IRQ_NOTIFY** register but only if the error interrupt source **TIM[i]_CH[x]_EIRQ_EN** is disabled. However, if the

enabling of a disabled interrupt is simultaneous to an incoming interrupt event, the interrupt event is dominant and the register **TIM[i]_CH[x]_IRQ_NOTIFY** is not cleared.

Attention:
Each write access to **TIM[i]_CH[x]_IRQ_MODE** clears all bits in **TIM[i]_CH[x]_IRQ_NOTIFY** . Write access to **TIM[i]_CH[x]_IRQ_MODE** shall only be executed if all interrupts of the corresponding channel are disabled. Otherwise, interrupt events can get lost or interrupt logic can go into an unpredictable state.

To secure a way for reconfiguring the interrupt mode of an interrupt set, proceed as follows:

1. Disable all interrupts via **TIM[i]_CH[x]_IRQ_EN** and all error interrupts via **TIM[i]_CH[x]_EIRQ_EN**
2. Define the new interrupt mode via **TIM[i]_CH[x]_IRQ_MODE**
3. Enable the desired interrupts via **TIM[i]_CH[x]_IRQ_EN** and desired error interrupts via **TIM[i]_CH[x]_EIRQ_EN**

The register **TIM[i]_CH[x]_IRQ_FORCINT** is used by software for triggering individual interrupts with a write access with value '1'. Since a write access to **TIM[i]_CH[x]_IRQ_FORCINT** only generates a single pulse, **TIM[i]_CH[x]_IRQ_FORCINT** is not implemented as a true register and thus any read access to **TIM[i]_CH[x]_IRQ_FORCINT** always results with a value of '0'. A write access with a one to particular bit field in this register shall force the corresponding bit field in **TIM[i]_CH[x]_IRQ_NOTIFY** to one.

The mechanism for triggering interrupts with **TIM[i]_CH[x]_IRQ_FORCINT** is globally disabled after reset. It has to be explicitly enabled by clearing the bit **GTM_CTRL.RF_PROT** .

For the modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE and CMP the interrupt set is extended by the register **TIM[i]_CH[x]_EIRQ_EN** . Each interrupt can be configured to trigger an error interrupt instead of the normal interrupt if it is enabled by the corresponding error interrupt enable bit in the **TIM[i]_CH[x]_EIRQ_EN** register.
It is possible for one source to enable the normal interrupt and the error interrupt in parallel. Because both interrupt clear signals can reset the notify bit, this is expected to cause problems in a system and it is strongly recommended not to enable both interrupt types at the same time.

Similar to enabling an interrupt, the enabling of a disabled error interrupt source with a write access to the register **TIM[i]_CH[x]_EIRQ_EN** also clears the corresponding bit in the **TIM[i]_CH[x]_IRQ_NOTIFY** register only if the interrupt source **TIM[i]_CH[x]_IRQ_EN** is disabled. However, if the enabling of a disabled error interrupt is simultaneous to an incoming interrupt event, the interrupt event is dominant and the register **TIM[i]_CH[x]_IRQ_NOTIFY** is not cleared.

All enabled error interrupts are OR-combined inside the ICM and assigned to the dedicated GTM port *GTM_ERR_IRQ* . A corresponding input *GTM_ERR_IRQ_CLR* allows the reset of this error interrupt from outside the GTM (hardware clear).

To be able to detect the module source of the error interrupt the ICM provides the register **ICM_IRQG_MEI** .

By evaluating the ICM register **ICM_IRQG_CEI0** , the channel causing the error interrupt for the module FIFO can be determined.

By evaluating the ICM register **ICM_IRQG_CEI1** , **ICM_IRQG_CEI2** , the channel causing the error interrupt can be determined for the module TIM.

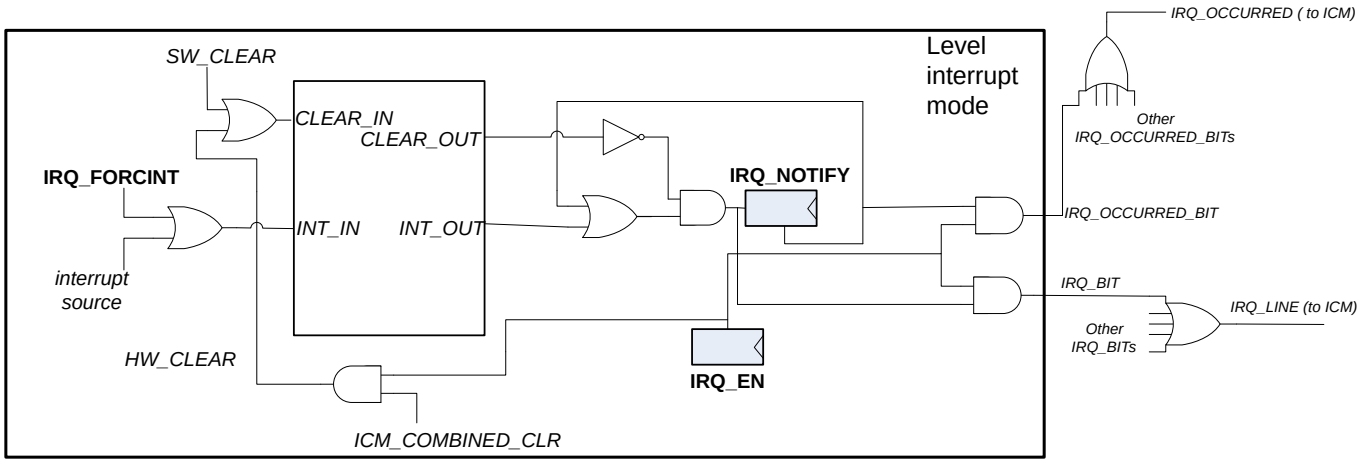
By evaluating the ICM register **ICM_IRQG_MCS[i]_CEI** , the channel causing the error interrupt can be determined for the module MCS with all possible channels.
In case of usage of only the first 8 channels of each MCS, the channel causing the error interrupt can be determined by evaluating the ICM register **ICM_IRQG_CEI3** , **ICM_IRQG_CEI4** .

3.12.1 Level Interrupt Mode

The default interrupt mode is the Level Interrupt Mode. In this mode each occurred interrupt event is collected in the register **TIM[i]_CH[x]_IRQ_NOTIFY** , independent of the corresponding enable bit of register **TIM[i]_CH[x]_IRQ_EN** and **TIM[i]_CH[x]_EIRQ_EN** .

An interrupt event, which is defined as a pulse on the signal *INT_OUT* of figure *19 "Level interrupt mode scheme"* , may be triggered by the interrupt source of the sub-module or by software performing a write access to the corresponding register **TIM[i]_CH[x]_IRQ_FORCINT** , with a cleared bit **GTM_CTRL.RF_PROT** .

Figure 19 Level interrupt mode scheme



A collected interrupt bit in register **TIM[i]_CH[x]_IRQ_NOTIFY** may be cleared by a clear event, which is defined as a pulse on signal *CLEAR_OUT* of figure *19 "Level interrupt mode scheme"* . A clear event can be performed by writing '1' to the corresponding bit in the register **TIM[i]_CH[x]_IRQ_NOTIFY** leading to a pulse on signals *SW_CLEAR* . A clear event may also result from an externally connected signal *GTM_TIM[i]_IRQ_CLR* , which is routed to the signal *HW_CLEAR* of figure *19 "Level interrupt mode scheme"* . However, the hardware clear mechanism is only possible, if the corresponding interrupt is enabled by register **TIM[i]_CH[x]_IRQ_EN** .

As table *9 "Priority of Interrupt Events and Clear Events"* shows, interrupt events are dominant in the case of a simultaneous interrupt event and clear event.

Table 9 Priority of Interrupt Events and Clear Events

INT_IN	CLEAR_IN	INT_OUT	CLEAR_OUT
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	0

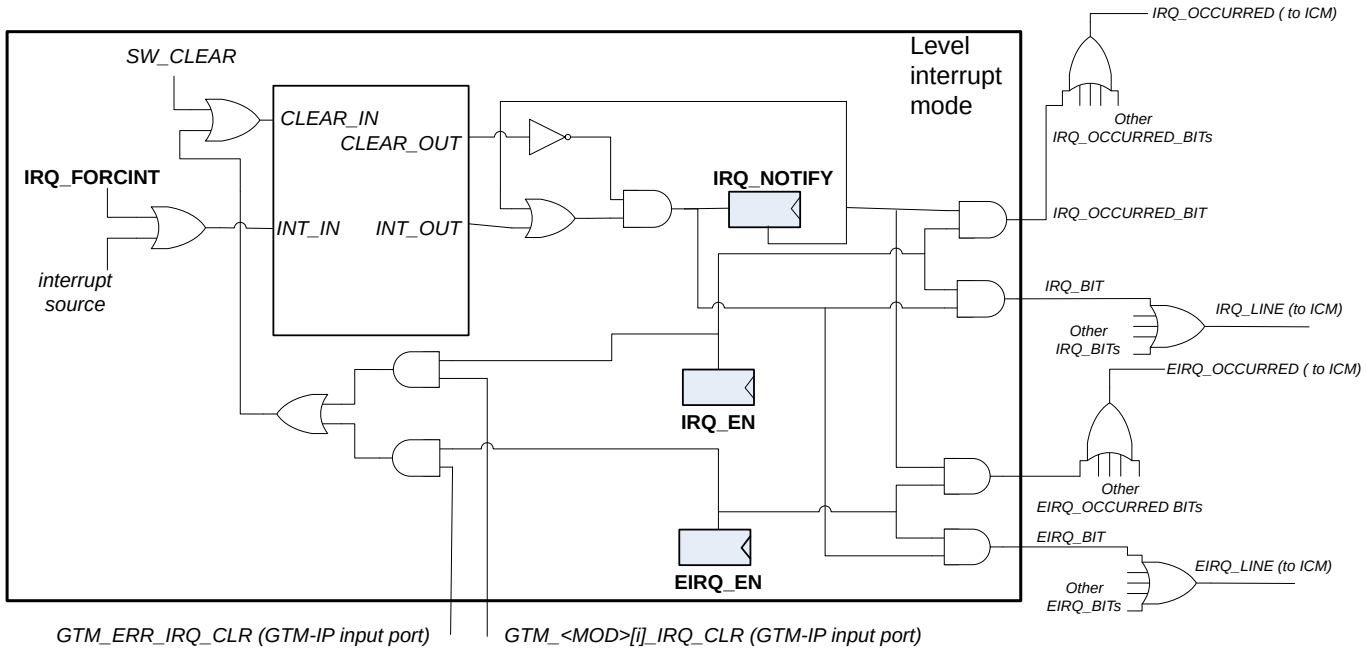
As shown in figure *19 "Level interrupt mode scheme"* an occurred interrupt event is signaled as a constant signal level with value 1 to the signal *IRQ_BIT* , if the corresponding interrupt is enabled in register **TIM[i]_CH[x]_IRQ_EN** . Moreover, the enabling of a disabled interrupt source with a write access to the register **TIM[i]_CH[x]_IRQ_EN** also clears the signal *IRQ_BIT* , but only if the error interrupt source **TIM[i]_CH[x]_EIRQ_EN** is disabled. However, if the enabling of a disabled interrupt is simultaneous to an incoming interrupt event, the interrupt event is dominant and signal *IRQ_BIT* , is not cleared.

With exception of the sub-modules ARU and DPLL, the signal *IRQ_BIT* is OR-combined with the neighboring *IRQ_BIT* signals of the same interrupt set and they are routed as a signal *IRQ_LINE* to the interrupt concentrator module (ICM). The interrupt signals *IRQ_BIT* of the sub-modules DPLL and ARU are routed directly as a signal *IRQ_LINE* to the sub-module ICM. In some cases (sub-modules TOM and ATOM) the ICM may further OR-combine several *IRQ_LINE* signals to an outgoing interrupt signal *GTM_TIM[i]_IRQ* . In the other cases the *IRQ_LINE* signals are directly connected to the outgoing signals *GTM_TIM[i]_IRQ* , within the sub-module ICM.

The signal *IRQ_OCCURRED* is connected in a similar way as the signal *IRQ_LINE* , however this signal is used for monitoring the interrupt state of the register **TIM[i]_CH[x]_IRQ_NOTIFY** in the registers of the ICM.

The additional error interrupt enable mechanism for level interrupt is shown below.

Figure 20 Level interrupt scheme for modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP



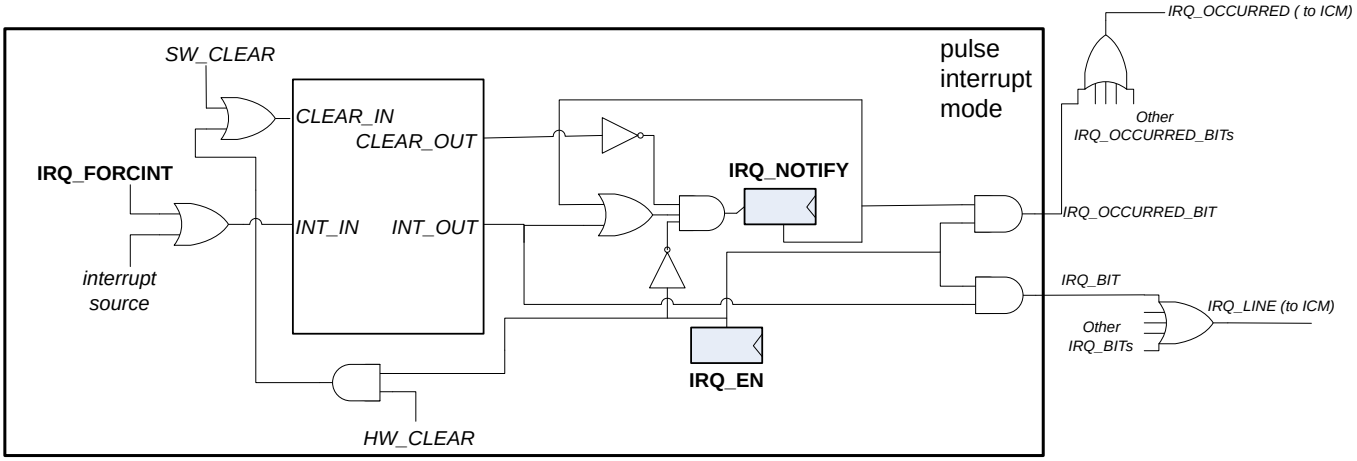
A collected interrupt bit in register **TIM[i]_CH[x]_IRQ_NOTIFY** may be cleared by a clear event, which is defined as a pulse on signal *CLEAR_OUT* of figure *20 "Level interrupt scheme for modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP"* . A clear event can be performed by writing '1' to the corresponding bit in the register **TIM[i]_CH[x]_IRQ_NOTIFY** leading to a pulse on signals *SW_CLEAR* . A clear event may also result from the externally connected signal *GTM_TIM[i]_IRQ_CLR* or *GTM_ERR_IRQ_CLR* , which is routed as an *HW_CLEAR* to *CLEAR_IN* of figure *20 "Level interrupt scheme for modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP"* . However, the hardware clear mechanism is only possible, if the corresponding interrupt or error interrupt is enabled by register **TIM[i]_CH[x]_IRQ_EN** or **TIM[i]_CH[x]_EIRQ_EN** .

As it can be seen from the figure *20 "Level interrupt scheme for modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP"* an occurred interrupt event is signaled as a constant signal level with value 1 to the signal *IRQ_BIT* , if the corresponding interrupt is enabled in register **TIM[i]_CH[x]_IRQ_EN** .

3.12.2 Pulse Interrupt Mode

The Pulse interrupt mode behavior can be observed from figure *21 "Pulse Interrupt Mode Scheme"* .

Figure 21 Pulse Interrupt Mode Scheme



In Pulse Interrupt Mode each Interrupt Event will generate a pulse on the *IRQ_BIT* signal if **TIM[i]_CH[x]_IRQ_EN** is enabled.

As it can be seen from the figure, the interrupt bit in **TIM[i]_CH[x]_IRQ_NOTIFY** register is always cleared if **TIM[i]_CH[x]_IRQ_EN** is enabled.

However, if an interrupt is disabled in the register **TIM[i]_CH[x]_IRQ_EN** , an occurred interrupt event is captured in the register **TIM[i]_CH[x]_IRQ_NOTIFY** , in order to allow polling for disabled interrupts by software.

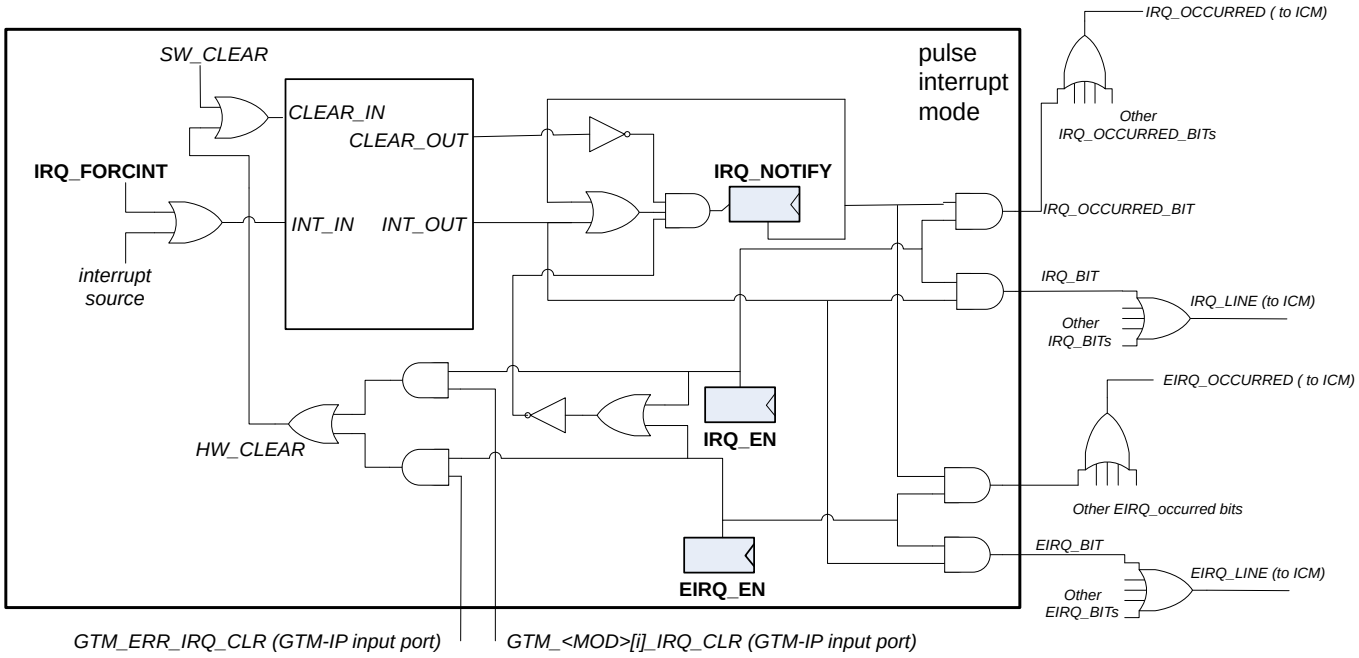
Disabled interrupts may be cleared by an interrupt clear event.

In case of a simultaneous interrupt event and clear event, the interrupt events are dominant (see details Table *9 "Priority of Interrupt Events and Clear Events"*).

In Pulse Interrupt Mode, the signal IRQ_OCCURRED is always 0.

The additional error interrupt enable mechanism for pulse interrupt is shown below.

Figure 22 Pulse Interrupt Scheme for Modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP



In Pulse Interrupt Mode each Interrupt Event will generate a pulse on the *EIRQ_BIT* signal if **TIM[i]_CH[x]_EIRQ_EN** is enabled.

As it can be seen from the figure, the interrupt bit in **TIM[i]_CH[x]_IRQ_NOTIFY** register is always cleared if **TIM[i]_CH[x]_EIRQ_EN** or **TIM[i]_CH[x]_IRQ_EN** are enabled.

However, if an error interrupt is disabled in the register **TIM[i]_CH[x]_EIRQ_EN** , an occurred error interrupt event is captured in the register **TIM[i]_CH[x]_IRQ_NOTIFY** , in order to allow polling for disabled error interrupts by software.

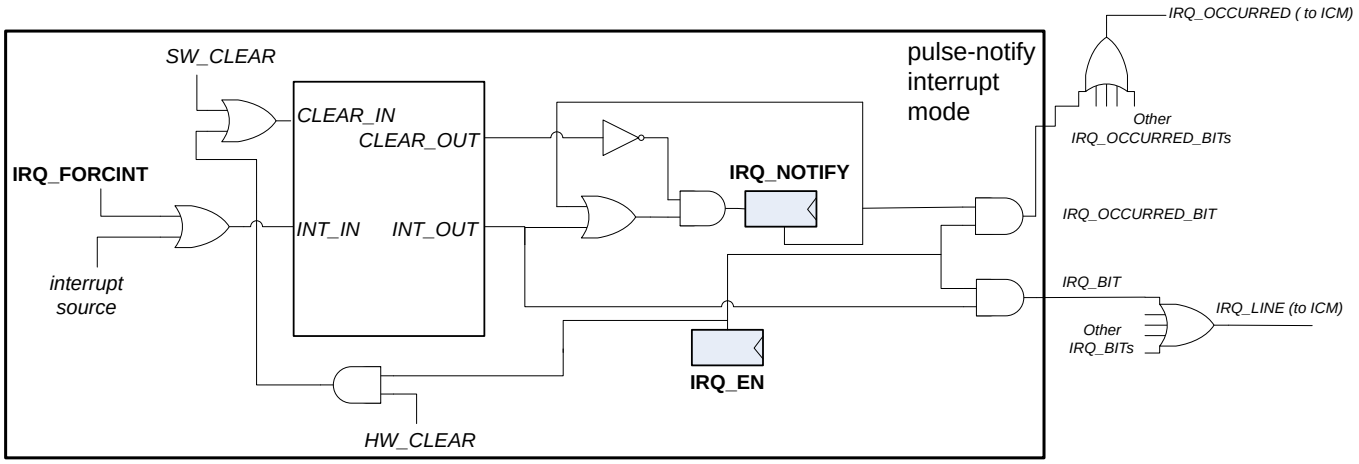
Disabled error interrupts may be cleared by an error interrupt clear event.

In Pulse interrupt mode, the signal EIRQ_OCCURRED is always 0.

3.12.3 Pulse-notify Interrupt Mode

In Pulse-notify Interrupt Mode, all interrupt events are captured in the register **TIM[i]_CH[x]_IRQ_NOTIFY** . If an interrupt is enabled by the register **TIM[i]_CH[x]_IRQ_EN** , each interrupt event will also generate a pulse on the *IRQ_BIT* signal. The signal *IRQ_OCCURRED* will be high if interrupt is enabled in register **TIM[i]_CH[x]_IRQ_EN** and the corresponding bit of register **TIM[i]_CH[x]_IRQ_NOTIFY** is set. The Pulse-notify interrupt mode is shown in figure *23 "Pulse-notify Interrupt Mode Scheme"* .

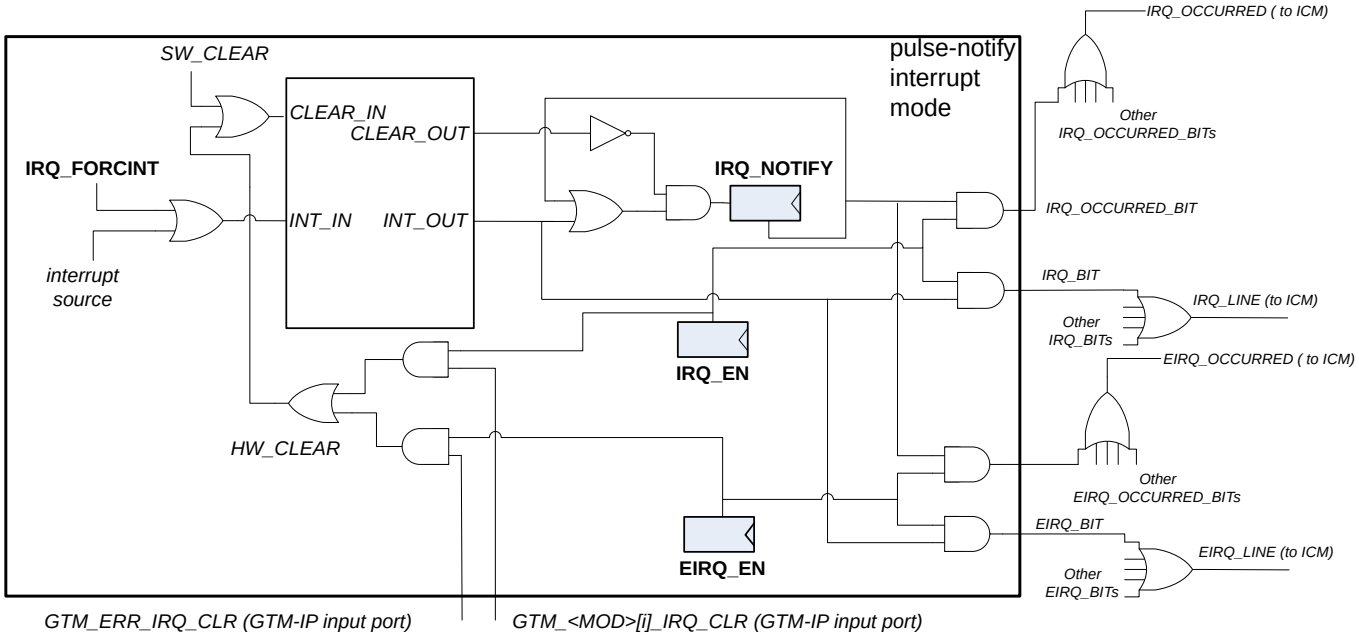
Figure 23 Pulse-notify Interrupt Mode Scheme



In case of a simultaneous interrupt event and clear event, the interrupt events are dominant (see details Table *9 "Priority of Interrupt Events and Clear Events"*).

The additional error interrupt enable mechanism for pulse-notify interrupt is shown below

Figure 24 Pulse-notify Interrupt Scheme for Modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP



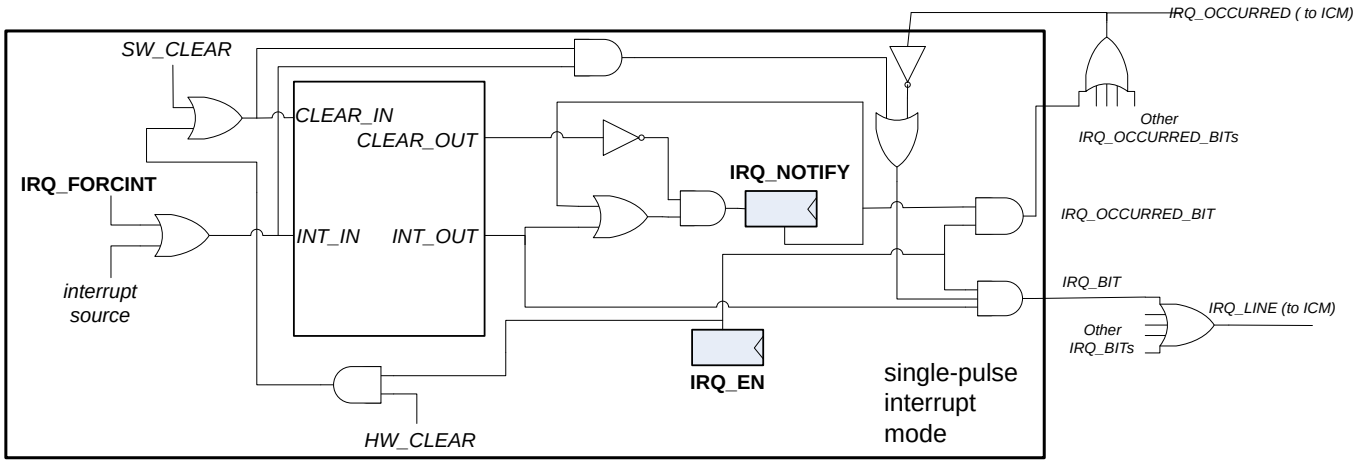
In Pulse-notify Interrupt mode, all error interrupt events are captured in the register **TIM[i]_CH[x]_IRQ_NOTIFY** . If an error interrupt is enabled by the register **TIM[i]_CH[x]_EIRQ_EN** , each error interrupt event will also generate a pulse on the *EIRQ_BIT* signal. The signal *EIRQ_OCCURRED* will be high if error interrupt is enabled in register **TIM[i]_CH[x]_EIRQ_EN** and the corresponding bit of register **TIM[i]_CH[x]_IRQ_NOTIFY** is set. The Pulse-notify interrupt mode for error interrupts is shown in figure *24 "Pulse-notify Interrupt Scheme for Modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP"* .

3.12.4 Single-pulse interrupt mode

In Single-pulse Interrupt Mode, an interrupt event is always captured in the register **TIM[i]_CH[x]_IRQ_NOTIFY** , independent of the state of **TIM[i]_CH[x]_IRQ_EN** . However, only the first interrupt event of an enabled interrupt within a common interrupt set is forwarded to signal *IRQ_LINE* . Additional interrupt events of the same interrupt set cannot generate pulses on the signal *IRQ_LINE* , until the corresponding bits in register **TIM[i]_CH[x]_IRQ_NOTIFY** of enabled interrupts are cleared by a clear event. This requirement is not valid when one of the following two cases is fulfilled. The first case is fulfilled if there is a simultaneous occurrence of interrupt event and an event on *HW_CLEAR* (or/and *SW_CLEAR*) input. The second case is fulfilled if an interrupt event occurs in the same clock cycle in which the corresponding disabled interrupt is enabled and the corresponding bit in **TIM[i]_CH[x]_IRQ_NOTIFY** is at 1. That means, the interrupt event in these cases will result in setting of *IRQ_LINE* output, even when register **TIM[i]_CH[x]_IRQ_NOTIFY** was previously not cleared. The *IRQ_OCCURRED* signal line will be high, if the **TIM[i]_CH[x]_IRQ_EN** and the **TIM[i]_CH[x]_IRQ_NOTIFY** register bits are set. The single-pulse interrupt mode is shown in figure *25 "Single-pulse interrupt mode scheme"* .

The only exceptions are the modules ARU and DPLL. In these modules the *IRQ_OCCURRED* bit of each interrupt is directly connected (without OR-conjunction of neighboring *IRQ_OCCURRED* bits) to the inverter for suppressing further interrupt pulses.

Figure 25 Single-pulse interrupt mode scheme

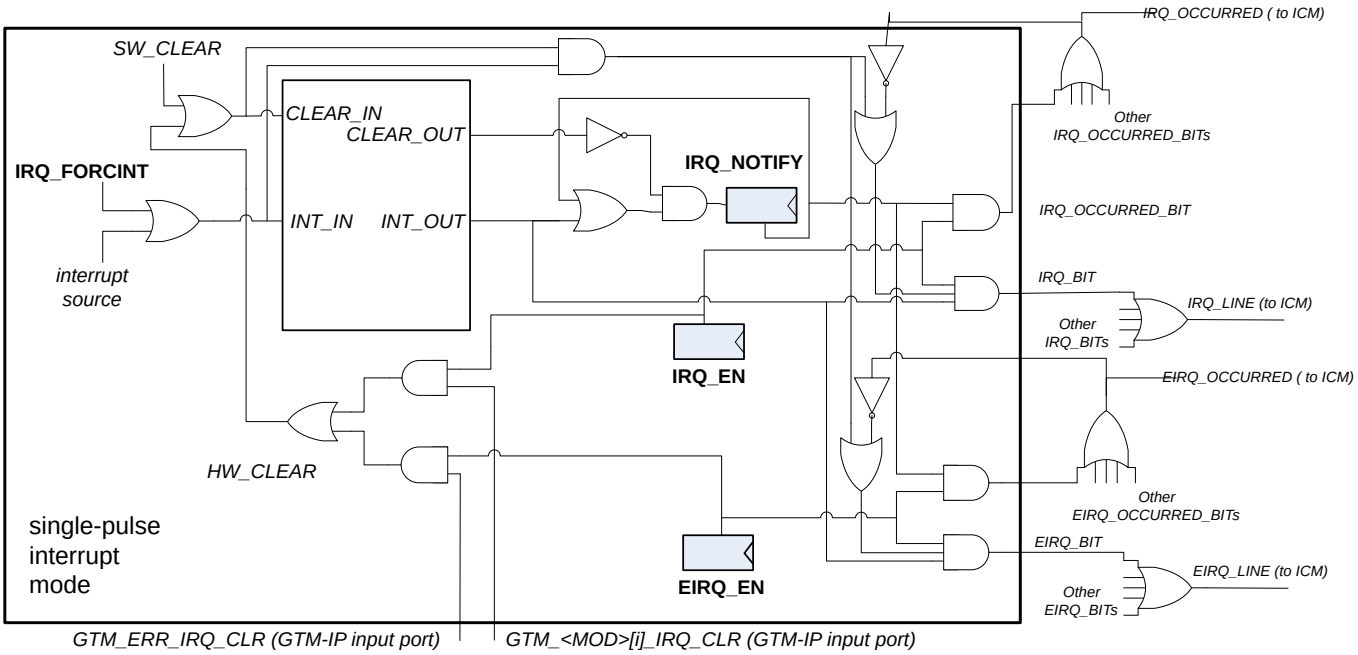


In case of a simultaneous interrupt event and clear event, the interrupt events are dominant (see details Table *9 "Priority of Interrupt Events and Clear Events"*).

To avoid unexpected IRQ behavior in the single pulse mode, all desired interrupt sources should be enabled by a single write access to **TIM[i]_CH[x]_IRQ_EN** and the notification bits should be cleared by a single write access to the register **TIM[i]_CH[x]_IRQ_NOTIFY** .

The additional error interrupt enable mechanism for single-pulse interrupt is shown below

Figure 26 Single-pulse interrupt scheme for modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP



In Single-pulse Interrupt Mode, an error interrupt event is always captured in the register **TIM[i]_CH[x]_IRQ_NOTIFY** , independent of the state of **TIM[i]_CH[x]_EIRQ_EN** . However, only the first error interrupt event of an enabled error interrupt within a common error interrupt set is forwarded to signal *EIRQ_LINE* . Additional error interrupt events of the same error interrupt set cannot generate pulses on the signal *EIRQ_LINE* , until the corresponding bits in register **TIM[i]_CH[x]_IRQ_NOTIFY** of enabled error interrupts are cleared by a clear event. The *EIRQ_OCCURRED* signal line will be high, if the **TIM[i]_CH[x]_EIRQ_EN** and the **TIM[i]_CH[x]_IRQ_NOTIFY** register bits are set. The Single-pulse interrupt mode for error interrupts is shown in figure *26 "Single-pulse interrupt scheme for modules AEI-bridge, BRC, FIFO, TIM, MCS, DPLL, SPE, CMP"* .

To avoid unexpected EIRQ behavior in the single pulse mode, all desired error interrupt sources should be enabled by a single write access to **TIM[i]_CH[x]_EIRQ_EN** and the notification bits should be cleared by a single write access to the register **TIM[i]_CH[x]_IRQ_NOTIFY** .

The only exceptions are the modules ARU and DPLL. In these modules the *EIRQ_OCCURRED* bit of each error interrupt is directly connected (without OR-conjunction of neighboring *EIRQ_OCCURRED* bits) to the inverter for suppressing further error interrupt pulses.

3.12.5 GTM-IP Interrupt Concentration Method

Because of the grouping of interrupts inside the ICM, it can be necessary for the software to access the ICM submodule first to determine the interrupt set that is responsible for an interrupt. A second access to the responsible register **TIM[i]_CH[x]_IRQ_NOTIFY** is then necessary to identify the interrupt source, serve it and to reset the interrupt flag in register **TIM[i]_CH[x]_IRQ_NOTIFY** afterwards. The interrupt flags are never reset by an access to the ICM. For a detailed description of the ICM submodule please refer to chapter *24 "Interrupt Concentrator Module (ICM)"* .

3.13 External GTM-IP Input Signals Which Can Influence / Control the GTM-IP Behavior

More details to these signals will be documented in the SOC where the GTM-IP is included.

3.14 GTM-IP Software Debugger Support

For software debugger support the GTM-IP comes with several features. E.g. status register bits must not be altered by a read access from a software debugger. To avoid this behavior to reset a status register bit by software, the CPU has to write a '1' explicitly to the register bit to reset its content.

The table *10 "Register Behavior in Case of Software Debugger Accesses"* describes the behavior of some GTM-IP registers with special functionality on behalf of read accesses from the AEI bus interface.

Table 10 Register Behavior in Case of Software Debugger Accesses

Module	Register	Description
AFD	**AFD[i]_CH[x]_BUF_ACC**	All FIFO fill level related registers (**FIFO[i]_CH[x]_RD_PTR** , **FIFO[i]_CH[x]_FILL_LEVEL** , **FIFO[i]_CH[x]_IRQ_NOTIFY** , **FIFO[i]_CH[x]_STATUS**) are not altered on behalf of a debugger read access to this register.
TIM	**TIM[i]_CH[x]_GPR0** / **TIM[i]_CH[x]_GPR1**	The overflow bit is not altered in case of a debugger read access to this register.
TIM	**TIM[i]_CH[x]_ECNT**	While **TIM[i]_CH[x]_CTRL.TIM_EN** = 0, the register is not cleared in case of a debugger read access to this register.
ATOM	**ATOM[i]_CH[x]_SR0** / **ATOM[i]_CH[x]_SR1**	In SOMC mode a read access to this register by the debugger does not release the channel for a new compare/match event.

The GTM-IP offers several debugging possibilities. For an exhaustive survey the user is referred to the module integration guide, and to the modules' respective chapters. It is up to the silicon vendor how to integrate/use these debugging possibilities and to which extent.

Upon the assertion of the *GTM_HALT_REQ* signal the cluster clocks are disabled stalling the internal operations within the GTM-IP. Once the cluster clocks are disabled, *GTM_HALT_ACTIVE* is asserted after a few clock cycles and the debugger can perform read/write access over the AEI interface to all memory mapped registers within the GTM-IP. When the external debugger deasserts *GTM_HALT_REQ* signal to mark the end of the debug accesses, the GTM-IP asserts *GTM_RESTORE* for a few clock cycles before it deasserts it again. During the assertion of *GTM_RESTORE* , the GTM-IP restores all transactions which were active on the internal AEI interface and were canceled when *GTM_HALT_REQ* was asserted. This restoration phase is introduced to make sure that the GTM-IP can resume working appropriately as if there was no debugger access at all.

In addition to enabling debug access to the memory addressable registers, the GTM-IP makes some important internal signals from different modules (MCS, DPLL, ARU, TIM, SPE, TOM, ATOM, TBU) available to the outside. An external debugger can monitor such signals and, for example, trace them or halt the GTM when the signals fulfill a predefined condition. Again, it is up to the silicon vendor to which extent to use these debugging possibilities.

3.15 GTM-IP Programming Conventions

To serve different application domains the GTM-IP is a highly configurable module with many configuration modes. In principle the submodules of the GTM-IP are intended to be configured at system startup to fulfill certain functionality for the application domain the microcontroller runs in.

For example, a TIM input channel can be used to monitor an application specific external signal, and this signal has to be filtered. Therefore, the configuration of the TIM channel filter mode will be specific to the external signal characteristic. While it can be necessary to adapt the filter thresholds during runtime an adaptation of the filter mode during runtime is not reasonable. Thus, the change of the filter mode during runtime can lead to an unexpected behavior.

In general, the programmer has to be careful when reprogramming configuration registers of the GTM-IP submodules during runtime. It is recommended to disable the channels before reconfiguration takes place to avoid unexpected behavior of the GTM-IP.

3.16 ARCH Configuration Registers Description

3.16.1 GTM_REV

Description	GTM version control register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_REV
Address	$0x0$
C-Name	GTM.CLS[0].ARCH.REV

Interface: MCS[i]

Name	GTM_REV
Address	$0x0$
C-Name	

REL_ITER	
Description	Delivery number
Loop	-
Bit Range	$[3:0]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	GTM_REV_REL_ITER
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Defines GTM-IP release iteration.

REL_BASE	
Description	Release step
Loop	-
Bit Range	$[11:4]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	GTM_REV_REL_BASE
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Defines GTM-IP release base.

VENDOR_CODE	
Description	Device encoding digit 1
Loop	-
Bit Range	$[19:16]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

VENDOR_CODE	
Initial value	GTM_REV_VENDOR_CODE
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Defines vendor code.

DEVICE_CODE	
Description	Device encoding digit 0
Loop	–
Bit Range	[23 : 20]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	GTM_REV_DEVICE_CODE
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Defines device code.

VER_MINOR	
Description	Minor version number
Loop	–
Bit Range	[27 : 24]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	GTM_REV_VER_MINOR
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Defines minor version number of GTM-IP specification.

VER_MAJOR	
Description	Major version number
Loop	–
Bit Range	[31 : 28]
Access Type	R
Volatile	False

VER_MAJOR	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	GTM_REV_VER_MAJOR
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Defines major version number of GTM–IP specification.

Note:
The numbers are encoded in BCD. Values "A" – "F" are characters.

Note:
See device specific appendix B for reset value.

3.16.2 GTM_RST

Description	GTM global reset register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	ARCH_CLK_ENABLE == 1

Interface: CPU

Name	GTM_RST
Address	$0x4$
C–Name	GTM.CLS[0].ARCH.RST

Interface: MCS[i]

Name	GTM_RST
Address	$0x4$
C–Name	

RST	
Description	GTM–IP Reset
Loop	–
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 0, 0) == 1)
Reset group	HW_RESET: async GTM_RESET: async

RST	
RW–Coding	0 : No reset action 1 : Initiate reset action for all sub–modules
	Note: This bit is automatically cleared by hardware after it was written. Therefore, the register is always read as zero (0) by the software. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [0:0] == 1 && *AEI_W1R0* ==1)

BRIDGE_MODE_WRDIS	
Description	BRIDGE_MODE write disable
Loop	–
Bit Range	$[27 : 27]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 27) != GTM_RST.BRIDGE_MODE_WRDIS)
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Writing of **BRIDGE_MODE** register is enabled 1 : Writing of **BRIDGE_MODE** register is disabled
	Note: This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [27:27] != **GTM_RST.BRIDGE_MODE_WRDIS** && *AEI_W1R0* ==1).

3.16.3 GTM_CTRL

Description	GTM global control register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_CTRL
Address	$0x8$
C–Name	GTM.CLS[0].ARCH.CTRL

Interface: MCS[i]

Name	GTM_CTRL
Address	$0x8$
C–Name	

RF_PROT	
Description	RST and FORCINT protection
Loop	–
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : SW RST (global), SW interrupt FORCINT, and SW RAM reset functionality is enabled 1 : SW RST (global), SW interrupt FORCINT, and SW RAM reset functionality is disabled

TO_MODE	
Description	AEI timeout mode
Loop	–
Bit Range	[2 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Observe once: If condition timeout_counter= **GTM_CTRL.TO_VAL** occurs the address *AEI_ADDR* and *AEI_W1R0* signal will be stored to the **GTM_AEI_ADDR_XPT** register, the timeout flag in **GTM_IRQ_NOTIFY.AEI_TO_XPT** will be set. Following timeout_counter = **GTM_CTRL.TO_VAL** accesses will not overwrite the first entry in the **GTM_AEI_ADDR_XPT** register. Clearing the timeout flag **GTM_IRQ_NOTIFY.AEI_TO_XPT** will reenable the store the next access for which the condition timeout_counter= **GTM_CTRL.TO_VAL** is fulfilled. 0b01 : Abort once: Same behavior as defined for observe once. Additionally, if condition timeout_counter= **GTM_CTRL.TO_VAL** occurs the pending access will be aborted by signaling an illegal module access on aei_status and ready to be sent. In case of a read access *AEI_RDATA* =0 is responded. Following accesses will not be aborted, this could lead to a dead lock if the access will never terminate with *AEI_READY* =1. 0b10 : Retry: Same behavior as defined for observe once. Additionally, if condition timeout_counter= **GTM_CTRL.TO_VAL** occurs each pending access is paused by signaling *AEI_SEL* =0 for 1 clock cycle, afterwards the access is active again. This leads to a retry of the actual access until the target address responds with *AEI_READY* =1. 0b11 : Abort: Same behavior as defined for observe once. Additionally, if condition timeout_counter= **GTM_CTRL.TO_VAL** occurs each pending access will be aborted by signaling an illegal module access on aei_status and ready to be sent. In case of a read access *AEI_RDATA* =0 is responded.

TO_VAL	
Description	AEI timeout value
Loop	–
Bit Range	[11 : 4]
Access Type	RW

TO_VAL	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: These bits define the number of cycles after which a timeout event occurs. When **GTM_CTRL.TO_VAL** equals zero (0) the AEI timeout functionality is disabled.

AEIM_CLUSTER	
Description	AEIM cluster number
Loop	–
Bit Range	$[15 : 12]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: These bits show the number of the AEI master port cluster which throws the interrupts *AEIM_USP_ADDR_IRQ* , *AEIM_IM_ADDR_IRQ* and *AEIM_USP_BE_IRQ* depending on the AEI master port access status. **Note:** If one of the corresponding irq notify bits [6:4] is set, this bit field will be frozen until the interrupt notify bits [6:4] are cleared.

3.16.4 GTM_CFG

Description	GTM configuration register
Loop	
Condition	$\mathrm{NTOM} > 0 \&\& \mathrm{NTIM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_CFG
Address	$0xC$

C-Name	GTM.CLS[0].ARCH.CFG

Interface: MCS[i]

Name	GTM_CFG
Address	$0xC$
C-Name	

SRC_IN_MUX	
Description	Input source selection for signal TIM[i]_AUX_IN (input port AUX_IN at module TIM)
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Use for TIM[i] channel j the output of TOM[k] channel j+h; k = floor(i / 2); h = mod(i , 2) *8 (j∈{0, 1,..., 7}) 1 : Use for TIM[i] channel j the output of TOM[i] channel j (TIM and TOM are in the same cluster) (j∈{0, 1,..., 7})
	See figure *5 "TIM Auxiliary Input Multiplexing"* for details.

3.16.5 GTM_AEI_ADDR_XPT

Description	GTM AEI timeout exception address register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_AEI_ADDR_XPT
Address	$0x10$
C-Name	GTM.CLS[0].ARCH.AEI_ADDR_XPT

Interface: MCS[i]

Name	GTM_AEI_ADDR_XPT
Address	$0x10$
C-Name	

TO_ADDR	
Description	AEI timeout address
Loop	–
Bit Range	[20 : 0]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async GTM_AEI_RESET: sync
	Note: This bit field defines the AEI address for which the AEI timeout event occurred.

TO_W1R0	
Description	AEI timeout Read/Write flag
Loop	–
Bit Range	[24 : 24]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async GTM_AEI_RESET: sync
R-Coding	0 : timeout occurred for read transaction 1 : timeout occurred for write transaction
	Note: This bit defines the AEI Read/Write flag for which the AEI timeout event occurred.

3.16.6 GTM_AEI_STA_XPT

Description	GTM AEI non zero status register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	ARCH_CLK_ENABLE == 1

Interface: CPU

Name	GTM_AEI_STA_XPT
Address	$0x14$
C-Name	GTM.CLS[0].ARCH.AEI_STA_XPT

Interface: MCS[i]

Name	GTM_AEI_STA_XPT
Address	$0x14$
C-Name	

ADDR	
Description	AEI exception address
Loop	-
Bit Range	$[20:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async GTM_AEI_RESET: sync
	Note: This bit field captures the address of the first AEI access resulting with a non-zero AEI status signal. The bit field can be cleared by clearing the interrupt flags **GTM_IRQ_NOTIFY.AEI_USP_ADDR** , **GTM_IRQ_NOTIFY.AEI_USP_BE** , and **GTM_IRQ_NOTIFY.AEI_IM_ADDR** **Note:** Exception: The address will not be captured in case a protected write (**GTM_RST.BRIDGE_MODE_WRDIS** = 1) to register **BRIDGE_MODE** is performed.

W1R0	
Description	AEI exception Read/Write flag
Loop	-
Bit Range	$[24:24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async GTM_AEI_RESET: sync

W1R0	
R-Coding	0 : Non zero status occurred for read transaction 1 : Non zero status occurred for write transaction
	Note: This bit field captures the address of the first AEI access resulting with a non-zero AEI status signal. The bit field can be cleared by clearing the interrupt flags **GTM_IRQ_NOTIFY.AEI_USP_ADDR** , **GTM_IRQ_NOTIFY.AEI_USP_BE** , and **GTM_IRQ_NOTIFY.AEI_IM_ADDR** . **Note:** Exception: This bit will not be set in case a protected write (**GTM_RST.BRIDGE_MODE_WRDIS** = 1) to register **BRIDGE_MODE** is performed.

3.16.7 GTM_IRQ_NOTIFY

Description	GTM Interrupt notification register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_IRQ_NOTIFY
Address	$0x18$
C-Name	GTM.CLS[0].ARCH.IRQ_NOTIFY

Interface: MCS[i]

Name	GTM_IRQ_NOTIFY
Address	$0x18$
C-Name	

AEI_TO_XPT	
Description	AEI timeout exception occurred
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

AEI_TO_XPT	
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

AEI_USP_ADDR	
Description	AEI unsupported address interrupt
Loop	–
Bit Range	$[1:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

AEI_IM_ADDR	
Description	AEI illegal Module address interrupt
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear Interrupt

AEI_IM_ADDR	
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged. **Note:** Exception: This bit will not be set in case a protected write (**GTM_RST.BRIDGE_MODE_WRDIS** = 1) to register **BRIDGE_MODE** is performed.

AEI_USP_BE	
Description	AEI unsupported byte enable interrupt
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear Interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

AEIM_USP_ADDR	
Description	AEI master port unsupported address interrupt
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear Interrupt

AEIM_USP_ADDR	
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

AEIM_IM_ADDR	
Description	AEI master port illegal Module address interrupt
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear Interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

AEIM_USP_BE	
Description	AEI master port unsupported byte enable interrupt
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear Interrupt

AEIM_USP_BE	
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged. **Note:** This bit cannot be triggered by an MCS bus master address alignment error. It can only be triggered by **GTM_IRQ_FORCINT.TRG_AEI_USP_BE** .

CLK_EN_ERR	
Description	Clock enable error interrupt
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	INT_CLK_EN_GEN == 0 0 : No interrupt raised 1 : Interrupt was raised
W-Coding	INT_CLK_EN_GEN == 0 0 : No action 1 : Clear interrupt
R-Coding	INT_CLK_EN_GEN == 1 0 : No interrupt raised 1 : Interrupt was raised
W-Coding	INT_CLK_EN_GEN == 1 0 : No action 1 : Clear Interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

CLK_PER_ERR	
Description	Clock period error interrupt
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–

CLK_PER_ERR	
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	INT_CLK_EN_GEN == 0 0 : No interrupt raised 1 : Interrupt was raised
W-Coding	INT_CLK_EN_GEN == 0 0 : No action 1 : Clear interrupt
R-Coding	INT_CLK_EN_GEN == 1 0 : No interrupt raised 1 : Interrupt was raised
W-Coding	INT_CLK_EN_GEN == 1 0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

CLK_EN_ERR_STATE	
Description	Erroneous clock enable state
Loop	–
Bit Range	$[25 : 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{INT\_CLK\_EN\_GEN} == 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0b00 : disable state for external clock *CLK* / 2; disable state for external clock *CLK* 0b01 : disable state for external clock CLK / 2; enable state for external clock CLK 0b10 : enable state for external clock CLK / 2; disable state for external clock CLK 0b11 : enable state for external clock CLK / 2; enable state for external clock CLK
	This bit field defines the GTM external clk enable state at occurrence of the **GTM_IRQ_NOTIFY.CLK_EN_ERR** event.

CLK_EN_EXP_STATE	
Description	Expected clock enable state
Loop	–
Bit Range	$[29 : 28]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–

CLK_EN_EXP_STATE	
Condition	$\mathrm{INT\_CLK\_EN\_GEN} == 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0b00 : Disable state expected for external clock CLK / 2; disable state expected for external clock CLK 0b01 : Disable state expected for external clock CLK / 2; enable state expected for external clock CLK 0b10 : Enable state expected for external clock CLK / 2; disable state expected for external clock CLK 0b11 : Enable state expected for external clock CLK / 2; enable state expected for external clock CLK
	This bit field defines the GTM expected clk enable state at occurrence of the **GTM_IRQ_NOTIFY.CLK_EN_ERR** event.

3.16.8 GTM_IRQ_EN

Description	GTM interrupt enable register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_IRQ_EN
Address	$0x1C$
C-Name	GTM.CLS[0].ARCH.IRQ_EN

Interface: MCS[i]

Name	GTM_IRQ_EN
Address	$0x1C$
C-Name	

AEI_TO_XPT_IRQ_EN	
Description	AEI_TO_XPT_IRQ interrupt enable
Loop	–
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

AEI_TO_XPT_IRQ_EN	
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

AEI_USP_ADDR_IRQ_EN	
Description	AEI_USP_ADDR_IRQ interrupt enable
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

AEI_IM_ADDR_IRQ_EN	
Description	AEI_IM_ADDR_IRQ interrupt enable
Loop	-
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

AEI_USP_BE_IRQ_EN	
Description	AEI_USP_BE_IRQ interrupt enable
Loop	-
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

AEI_USP_BE_IRQ_EN	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

AEIM_USP_ADDR_IRQ_EN	
Description	AEI_USP_ADDR_IRQ interrupt enable
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

AEIM_IM_ADDR_IRQ_EN	
Description	AEIM_IM_ADDR_IRQ interrupt enable
Loop	–
Bit Range	$[5:5]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

AEIM_USP_BE_IRQ_EN	
Description	AEIM_USP_BE_IRQ interrupt enable
Loop	–
Bit Range	$[6:6]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

AEIM_USP_BE_IRQ_EN	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

CLK_EN_ERR_IRQ_EN	
Description	CLK_EN_ERR_IRQ interrupt enable
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

CLK_PER_ERR_IRQ_EN	
Description	CLK_PER_ERR_IRQ interrupt enable
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

3.16.9 GTM_EIRQ_EN

Description	GTM error interrupt enable register
Loop	
Condition	
Storage Type	REGISTER

Clock Active Cond	ARCH_CLK_ENABLE == 1

Interface: CPU

Name	GTM_EIRQ_EN
Address	$0x20$
C-Name	GTM.CLS[0].ARCH.EIRQ_EN

Interface: MCS[i]

Name	GTM_EIRQ_EN
Address	$0x20$
C-Name	

AEI_TO_XPT_EIRQ_EN	
Description	AEI_TO_XPT_EIRQ error interrupt enable
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable error interrupt, interrupt is not visible outside GTM-IP 1 : Enable error interrupt, interrupt is visible outside GTM-IP

AEI_USP_ADDR_EIRQ_EN	
Description	AEI_USP_ADDR_EIRQ error interrupt enable
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable error interrupt, interrupt is not visible outside GTM-IP 1 : Enable error interrupt, interrupt is visible outside GTM-IP

AEI_IM_ADDR_EIRQ_EN	
Description	AEI_IM_ADDR_EIRQ error interrupt enable
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable error interrupt, interrupt is not visible outside GTM-IP 1 : Enable error interrupt, interrupt is visible outside GTM-IP

AEI_USP_BE_EIRQ_EN	
Description	AEI_USP_BE_EIRQ error interrupt enable
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable error interrupt, interrupt is not visible outside GTM-IP 1 : Enable error interrupt, interrupt is visible outside GTM-IP

AEIM_USP_ADDR_EIRQ_EN	
Description	AEIM_USP_ADDR_EIRQ error interrupt enable
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

AEIM_USP_ADDR_EIRQ_EN	
RW-Coding	0 : Disable error interrupt, interrupt is not visible outside GTM-IP 1 : Enable error interrupt, interrupt is visible outside GTM-IP

AEIM_IM_ADDR_EIRQ_EN	
Description	AEIM_IM_ADDR_EIRQ error interrupt enable
Loop	-
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable error interrupt, interrupt is not visible outside GTM-IP 1 : Enable error interrupt, interrupt is visible outside GTM-IP

AEIM_USP_BE_EIRQ_EN	
Description	AEIM_USP_BE_EIRQ error interrupt enable
Loop	-
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable error interrupt, interrupt is not visible outside GTM-IP 1 : Enable error interrupt, interrupt is visible outside GTM-IP

CLK_EN_ERR_EIRQ_EN	
Description	CLK_EN_ERR_EIRQ interrupt enable
Loop	-
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

CLK_EN_ERR_EIRQ_EN	
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable interrupt, interrupt is not visible outside GTM–IP 1 : Enable interrupt, interrupt is visible outside GTM–IP

CLK_PER_ERR_EIRQ_EN	
Description	CLK_PER_ERR_EIRQ interrupt enable
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable interrupt, interrupt is not visible outside GTM–IP 1 : Enable interrupt, interrupt is visible outside GTM–IP

3.16.10 GTM_IRQ_FORCINT

Description	GTM Software interrupt generation register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	ARCH_CLK_ENABLE == 1

Interface: CPU

Name	GTM_IRQ_FORCINT
Address	$0x24$
C–Name	GTM.CLS[0].ARCH.IRQ_FORCINT

Interface: MCS[i]

Name	GTM_IRQ_FORCINT
Address	$0x24$
C–Name	

TRG_AEI_TO_XPT	
Description	Trigger the bit GTM_IRQ_NOTIFY.AEI_TO_XPT by software.
Loop	–

TRG_AEI_TO_XPT	
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **GTM_TRQ_NOTIFY.AEI_TO_XPT**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 && *AEI_W1R0* ==1).

TRG_AEI_USP_ADDR	
Description	Trigger the bit GTM_IRQ_NOTIFY.AEI_USP_ADDR by software.
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **GTM_IRQ_NOTIFY.AEI_USP_ADDR**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 AND *AEI_W1R0* ==1).

TRG_AEI_IM_ADDR	
Description	Trigger the bit GTM_IRQ_NOTIFY.AEI_IM_ADDR by software.
Loop	–
Bit Range	[2 : 2]
Access Type	RW

TRG_AEI_IM_ADDR	
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No status
W–Coding	0 : No action 1 : Force setting of **GTM_IRQ_NOTIFY.AEI_IM_ADDR**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 && *AEI_W1R0* ==1).

TRG_AEI_USP_BE	
Description	Trigger the bit GTM_IRQ_NOTIFY.AEI_USP_BE by software.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No status
W–Coding	0 : No action 1 : Force setting of **GTM_IRQ_NOTIFY.AEI_USP_BE**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 AND *AEI_W1R0* ==1).

TRG_AEIM_USP_ADDR	
Description	Trigger the bit GTM_IRQ_NOTIFY.AEIM_USP_ADDR by software.
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear

TRG_AEIM_USP_ADDR	
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **GTM_IRQ_NOTIFY.AEIM_USP_ADDR**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 AND *AEI_W1R0* ==1).

TRG_AEIM_IM_ADDR	
Description	Trigger the bit GTM_IRQ_NOTIFY.AEIM_IM_ADDR by software.
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **GTM_IRQ_NOTIFY.AEIM_IM_ADDR**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 AND *AEI_W1R0* ==1).

TRG_AEIM_USP_BE	
Description	Trigger the bit GTM_IRQ_NOTIFY.AEIM_USP_BE by software.
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0

TRG_AEIM_USP_BE	
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **GTM_IRQ_NOTIFY.AEIM_USP_BE**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 AND *AEI_W1R0* ==1).

TRG_CLK_EN_ERR	
Description	Trigger the bit GTM_IRQ_NOTIFY.CLK_EN_ERR by software.
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **GTM_IRQ_NOTIFY.CLK_EN_ERR**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 && *AEI_W1R0* ==1).

TRG_CLK_PER_ERR	
Description	Trigger the bit GTM_IRQ_NOTIFY.CLK_PER_ERR by software.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 8, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async

TRG_CLK_PER_ERR	
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **GTM_IRQ_NOTIFY.CLK_PER_ERR**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by (**GTM_CTRL.RF_PROT** ==1 && *AEI_WDATA* [8:0] != 0 && *AEI_W1R0* ==1).

3.16.11 GTM_IRQ_MODE

Description	GTM top level interrupts mode selection
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_IRQ_MODE
Address	$0x28$
C-Name	GTM.CLS[0].ARCH.IRQ_MODE

Interface: MCS[i]

Name	GTM_IRQ_MODE
Address	$0x28$
C-Name	

IRQ_MODE	
Description	Interrupt strategy mode selection for the AEI timeout and address monitoring interrupts.
Loop	-
Bit Range	$[1:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$\mathrm{IRQ\_MODE\_RST\_VAL}$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode

IRQ_MODE	
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* . **Note:** This mode selection is only valid for the interrupts described in register **GTM_IRQ_NOTIFY** .

3.16.12 GTM_CLS_CLK_CFG

Description	GTM Cluster Clock Configuration
Loop	
Condition	$\mathrm{NCCM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_CLS_CLK_CFG
Address	$0x2C$
C-Name	GTM.CLS[0].ARCH.CLK_CFG

Interface: MCS[i]

Name	GTM_CLS_CLK_CFG
Address	$0x2C$
C-Name	

CLS[j]_CLK_DIV	
Description	Cluster [j] Clock Divider
Loop	$j = \{n : 0 <= n <= 11\}$
Bit Range	$[2 * j + 1 : 2 * j]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NCCM} > j$
Initial value	$1 + \mathrm{CFG\_CLOCK\_RATE}$
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CFG_CLOCK_RATE == 0 0b00 : Cluster is disabled. 0b01 : Cluster is enabled without clock divider.
RW-Coding	CFG_CLOCK_RATE == 1 0b00 : Cluster is disabled. 0b01 : Cluster is enabled without clock divider. 0b10 : Cluster is enabled with clock divider 2.

CLS[j]_CLK_DIV	
	Note: These bits are only writable if bit field **GTM_CTRL.RF_PROT** is cleared. **Note:** Writing a reserved value e.g.: 0b11 will not change the bit field value. **Note:** The configuration value **GTM_CLS_CLK_CFG.CLS[0]_CLK_DIV** defines the primary input clock period for CMU. **Note:** If **GTM_CLS_CLK_CFG.CLS[0]_CLK_DIV** is configured to a value 0b10 (i.e. clock divider 2), the maximum CMU resolution signal frequency for all other cluster k=1..n is also limited to configured CMU resolution frequency of cluster 0.

Note:
If **CCM[0]_HW_CONF.CFG_CLOCK_RATE** =1, there might be limitation for some cluster (FAST_CLK_CLS[j]=0, with j = 0 to NCCM−1) to enable high frequency clock usage. In this case only values 0b00 and 0b10 are valid for bit fields **GTM_CLS_CLK_CFG.CLS[j]_CLK_DIV** (with j = 0 to NCCM−1).

3.16.13 GTM_ARU_COM_DIS

Description	GTM ARU communication disable
Loop	
Condition	$\mathrm{NCCM} > 0 \&\& \mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	GTM_ARU_COM_DIS
Address	$0x30$
C−Name	GTM.CLS[0].ARCH.ARU_COM_DIS

Interface: MCS[i]

Name	GTM_ARU_COM_DIS
Address	$0x30$
C−Name	

CLS[j]_DIS	
Description	Disable cluster [j] ARU communication
Loop	$j = \{n : 0 <= n <= (\mathrm{NCCM} - 1)\}$
Bit Range	$[j : j]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	−
Condition	−
Initial value	0
Protect Enable Cond	RF_PROT == 1

CLS[j]_DIS	
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : ARU communication to all clusters enabled 1 : ARU communication disabled to all clusters, except the own
	Note: These bits are only writable if bit field **GTM_CTRL.RF_PROT** is cleared.

3.17 AEI Configuration Registers Description

3.17.1 BRIDGE_MODE

Description	GTM AEI bridge mode register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	1

Interface: CPU

Name	BRIDGE_MODE
Address	$0x40$
C-Name	GTM.CLS[0].AEI.BRIDGE_MODE

BRG_MODE	
Description	Defines the operation mode for the AEI bridge
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	(GTM_RST.BRIDGE_MODE_WRDIS == 1)
Reset group	HW_CPU_IF_RESET: async
RW-Coding	0 : AEI bridge operates in sync_bridge mode 1 : AEI bridge operates in async_bridge mode

MSK_WR_RSP	
Description	Mask write response
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

MSK_WR_RSP	
Condition	–
Initial value	0
Protect Enable Cond	(GTM_RST.BRIDGE_MODE_WRDIS == 1)
Reset group	HW_CPU_IF_RESET: async
RW–Coding	0 : Do not mask the write response. Depending on the selected address the latency for execution can vary due to GTM internal arbitration. After this time the status of the access will be signaled by the signal AEI_STATUS to the bus interface. 1 : Mask write response. The write buffer of the bridge is activated, the actual access will be stored to the write buffer and without latency on the bus interface the acceptance of the access is signaled. AEI_STATUS=0b00 will be signaled. In case of a full write buffer the actual access will be postponed until the next write buffer entry becomes free.
	Note: The status of the executed write accesses can be observed by using the notify bits **GTM_IRQ_NOTIFY.AEI_USP_ADDR** , **GTM_IRQ_NOTIFY.AEI_IM_ADDR** , **GTM_IRQ_NOTIFY.AEI_USP_BE** . **Note:** The status of the executed write accesses can be observed by using the notify bits **GTM_IRQ_NOTIFY.AEI_USP_ADDR** , **GTM_IRQ_NOTIFY.AEI_IM_ADDR** , **GTM_IRQ_NOTIFY.AEI_USP_BE** . **Note:** With active write buffer **BRIDGE_MODE.MSK_WR_RSP** =1, execution of actions can be delayed due to previous inserted write actions in the transaction buffer which wait to be serviced. This can lead to the fact that an access on the bus to a different peripheral than the GTM might be executed earlier in time than the write access buffered in the GTM. Applications must be setup up with this in mind otherwise unexpected operation can happen.

BYPASS_SYNC	
Description	Bypass synchronizer storage elements
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(GTM_RST.BRIDGE_MODE_WRDIS == 1)
Reset group	HW_CPU_IF_RESET: async
RW–Coding	BRIDGE_MODE.BRG_MODE == 1 0 : Synchronizer storage elements in use, latency increase due to synchronization (aei_clk to CLK and back CLK to aei_clk). This setting must be used if aei_clk and CLK operate fully asynchronous by independent clock sources. 1 : Synchronizer storage elements are bypassed. No additional latency due to synchronization. This setting can be used if aei_clk and CLK are generated by clock gating or clock division out of a common clock source. Clock edges on aei_clk and CLK generated out of the same clock edge of the common clock source must have zero skew.
RW–Coding	BRIDGE_MODE.BRG_MODE == 0 0 : In synchronous mode (**BRIDGE_MODE.BRG_MODE** = 0), this bit field is not relevant and has no impact. 1 : In synchronous mode (**BRIDGE_MODE.BRG_MODE** = 0), this bit field is not relevant and has no impact.

MODE_UP_PGR	
Description	Mode update in progress.
Loop	–

MODE_UP_PGR	
Bit Range	[8 : 8]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_CPU_IF_RESET: async
R–Coding	0 : No update in progress. 1 : Update in progress.

BUFF_OVL	
Description	Buffer overflow register
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	(GTM_RST.BRIDGE_MODE_WRDIS == 1)
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
R–Coding	0 : No buffer overflow occurred. 1 : Buffer overflow occurred.
W–Coding	0 : No action. 1 : Clear buffer overflow status.
	Note: A buffer overflow can occur while multiple aborts are issued by the external bus or a pipelined instruction is started while FBC = 0 (see **BRIDGE_PTR1** register).

SYNC_INPUT_REG	
Description	Additional pipelined stage in synchronous bridge mode
Loop	–
Bit Range	[12 : 12]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	SYNC_INPUT_REG
Protect Enable Cond	–

SYNC_INPUT_REG	
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
R–Coding	0 : No additional pipelined stage implemented. 1 : Additional pipelined stage implemented. All accesses in synchronous mode will be increased by one clock cycle.
	Note: Reset value depends on the hardware configuration chosen by silicon vendor.

BRG_RST	
Description	Bridge software reset
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(GTM_RST.BRIDGE_MODE_WRDIS == 1)
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
W–Coding	0 : No bridge reset request. 1 : Bridge reset request.
R–Coding	0 : No status
	Note: This bit is cleared while bridge reset is executed.

BUFF_DPT	
Description	Buffer depth of AEI bridge
Loop	–
Bit Range	[31 : 24]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	BRIDGE_BUFF_DPT
Protect Enable Cond	–
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
	Signals the buffer depth of the GTM AEI bridge implementation. **Note:** Reset value depends on the hardware configuration chosen by silicon vendor.

Note:
All writable bits are write protected by bit **GTM_RST.BRIDGE_MODE_WRDIS** .

3.17.2 BRIDGE_PTR1

Description	GTM AEI bridge pointer 1 register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	1

Interface: CPU

Name	BRIDGE_PTR1
Address	$0x44$
C-Name	GTM.CLS[0].AEI.BRIDGE_PTR1

NEW_TRAN_PTR	
Description	New transaction pointer
Loop	-
Bit Range	$[4:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
	Signals the actual value of the new transaction pointer.

FIRST_RSP_PTR	
Description	First response pointer
Loop	-
Bit Range	$[9:5]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync

FIRST_RSP_PTR	
	Signals the actual value of first response pointer.

TRAN_IN_PGR	
Description	Transaction in progress pointer (acquire)
Loop	–
Bit Range	[14 : 10]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
	Transaction in progress pointer.

ABT_TRAN_PGR	
Description	Aborted transaction in progress pointer
Loop	–
Bit Range	[19 : 15]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
	Aborted transaction in progress pointer.

FBC	
Description	Free buffer count
Loop	–
Bit Range	[25 : 20]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	BRIDGE_BUFF_DPT

FBC	
Protect Enable Cond	–
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
	Number of free buffer entries. **Note:** Initial value depends on the hardware configuration chosen by silicon vendor. (see bit field **BRIDGE_MODE.BUFF_DPT**).

RSP_TRAN_RDY	
Description	Response transactions ready
Loop	–
Bit Range	$[31 : 26]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
	Number of ready response transactions.

Note:
This register operates on the AEI_CLK domain.

Note:
This register holds diagnosis information about the AEI bus bridge. Each access to the GTM-IP will update the defined pointer bit fields. Depending on which values of **BRIDGE_MODE.BRG_MODE** , **BRIDGE_MODE.MSK_WR_RSP** , the AEI protocol, and operating frequency are used, the 4 pointer bit fields change at different clock cycles relative to the start of the transaction. As a result of this, reading the register can show values not equal to the defined initial value, even directly after a write to **BRIDGE_MODE.BRG_RST** =1.

3.17.3 BRIDGE_PTR2

Description	GTM AEI bridge pointer 2 register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	BRIDGE_PTR2
Address	$0x48$
C-Name	GTM.CLS[0].AEI.BRIDGE_PTR2

TRAN_IN_PGR2	
Description	Transaction in progress pointer (aquire2)

TRAN_IN_PGR2	
Loop	–
Bit Range	$[4:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_CPU_IF_RESET: async GTM_AEI_RESET: sync
	Transaction in progress pointer 2.

Note:
This register operates on the GTM_CLK domain.

3.17.4 MCS_AEM_DIS

Description	GTM MCS master port disable register
Loop	
Condition	$\mathrm{NMCS} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARCH\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS_AEM_DIS
Address	$0x4C$
C-Name	GTM.CLS[0].AEI.MCS_AEM_DIS

DIS_CLS[j]	
Description	Disable MCS AEIM access in cluster [j]
Loop	$j = \{n : 0 <= n <= (\mathrm{NMCS} - 1)\}$
Bit Range	$[2 * j + 1 : 2 * j]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

DIS_CLS[j]	
W-Coding	0b00 : Don't care, bits will not be changed 0b01 : Enable MCS AEIM access in cluster j 0b10 : Disable MCS AEIM access in cluster j 0b11 : Don't care, bits will not be changed
R-Coding	0b00 : MCS AEIM access in cluster j enabled 0b01 : unused 0b10 : unused 0b11 : MCS AEIM access in cluster j disabled
	Multithread encoding in use (**MCS_AEM_DIS.DIS_CLS[j]** (1) defines the state of the signal) **Note:** Any read access to a **MCS_AEM_DIS.DIS_CLS[j]** bit field will always result in a value 00 or 11 indicating current state. A modification of the state is only performed with the values 01 and 10. Writing the values 00 and 11 is always ignored.

3.18 GTM TOP-Level Port Description

3.18.1 GTM clock/reset/infrastructure interface

Loop	-
Condition	-
Format	-

CLK	
Description	GTM main clock
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC
Port Direction	IN
Port Range	-
Operational Clock	-
Special Function	isClock
Operational Reset	-
Reset Value	-

RESET	
Description	GTM reset
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC
Port Direction	IN
Port Range	-
Operational Clock	CLK
Special Function	isReset
Operational Reset	-
Reset Value	-

SCAN_MODE	
Description	Signal to bypass reset_mux in scan mode
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	–
Reset Value	1

TEST_CLK	
Description	Testclock for clusters running on divided clock only; in use with scan_mode=1
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	isClock
Operational Reset	–
Reset Value	–

CGATE_TE	
Description	Enable clock passing in clock gating cells for ATPG test purpose
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	–
Reset Value	–

3.18.2 GTM clock enable interface

Loop	–
Condition	$\mathrm{INT\_CLK\_EN\_GEN} == 0$
Format	–

CLK_EN	
Description	GTM main clock enable
Loop	–

CLK_EN	
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	CLK
Special Function	isClockEnable
Operational Reset	–
Reset Value	–

GTM_WDG_ERR	
Description	Watchdog error indicating an invalid CLK_EN
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	–
Reset Value	–

3.18.3 GTM toplevel AEI interface

Loop	–
Condition	$\mathrm{USE\_AXIS} == 0$
Format	–

AEI_CLK	
Description	AEI interface clock
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	isClock
Operational Reset	–
Reset Value	–

AEI_RESET	
Description	AEI interface reset (deassertion synchronized to AEI_CLK)
Loop	–
Condition	–
Logical Name	–

AEI_RESET	
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AEI_CLK
Special Function	isReset
Operational Reset	–
Reset Value	–

AEI_SEL	
Description	Indicates active AEI transfer
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_PIPE	
Description	Indicates AEI protocol and valid address phase
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_ADDR	
Description	AEI access address
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	20 DOWNTO 0
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET

AEI_ADDR	
Reset Value	0

AEI_W1R0	
Description	AEI access type (read/write)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_WDATA	
Description	AEI write data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	31 DOWNTO 0
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_RDATA	
Description	AEI read data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	31 DOWNTO 0
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_READY	
Description	AEI access ready
Loop	–
Condition	–

AEI_READY	
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_STATUS	
Description	AEI access status
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_SPLIT	
Description	Indicates AEI split transaction protocol type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_RESPONSE_REQ	
Description	Indicates request for AEI response data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET

AEI_RESPONSE_REQ	
Reset Value	0

AEI_RESPONSE_READY	
Description	AEI response data ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

AEI_RESPONSE_TRANSACTION_CNT	
Description	Number of ready response transfers
Loop	–
Condition	BRIDGE_BUFF_DPT == 1
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	0 DOWNTO 0
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	–
Reset Value	–

AEI_FREE_BUFFER_CNT	
Description	Number of free transfer buffer entries
Loop	–
Condition	BRIDGE_BUFF_DPT == 1
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	0 DOWNTO 0
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	–
Reset Value	–

AEI_DEBUG_ACCESS	
Description	Sideband signal controlling debugger access
Loop	–
Condition	–

AEI_DEBUG_ACCESS	
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AEI_CLK
Special Function	–
Operational Reset	AEI_RESET
Reset Value	0

3.18.4 GTM toplevel AXIS interface

Loop	–
Condition	USE_AXIS == 1
Format	–

AXIS_ACLK	
Description	AXI reference clock
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	0

AXIS_ARESETN	
Description	AXI reset signal (low active)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	–
Reset Value	0

AXIS_ARID	
Description	AXI transaction ID, ID width can be selected via device configuration parameter
Loop	–
Condition	–

AXIS_ARID	
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	axis_id_width_g-1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_ARADDR	
Description	AXI read address
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	31 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_ARLEN	
Description	AXI read burst length
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	3 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_ARBURST	
Description	AXI read burst type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN

AXIS_ARBURST	
Reset Value	0

AXIS_ARSIZE	
Description	AXI read size, must always be 0b010, as we only allow word accesses
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	2 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_ARVALID	
Description	AXI read valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_ARREADY	
Description	AXI read ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_AWID	
Description	AXI transaction ID, ID width can be selected via device configuration parameter
Loop	–
Condition	–

AXIS_AWID	
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	axis_id_width_g-1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_AWADDR	
Description	AXI write address
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	31 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_AWLEN	
Description	AXI write burst length
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	3 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_AWBURST	
Description	AXI write burst type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN

AXIS_AWBURST	
Reset Value	0

AXIS_AWSIZE	
Description	AXI write data size (must always be 0b010, as we only support 32bit accesses)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	2 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_AWVALID	
Description	AXI write data valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_AWREADY	
Description	AXI write data ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_RID	
Description	AXI transaction ID, ID width can be selected via device configuration parameter
Loop	–
Condition	–

AXIS_RID	
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	axis_id_width_g-1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_RDATA	
Description	AXI read data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	axis_data_width_g-1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_RLAST	
Description	AXI real last in a burst signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_RRESP	
Description	AXI read response
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN

AXIS_RRESP	
Reset Value	0

AXIS_RVALID	
Description	AXI read data valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_RREADY	
Description	AXI read data ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_WID	
Description	AXI transaction ID, ID width can be selected via device configuration parameter
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	axis_id_width_g-1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_WDATA	
Description	AXI write data
Loop	–
Condition	–

AXIS_WDATA	
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	axis_data_width_g-1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_WLAST	
Description	AXI write last in a burst signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_WSTRB	
Description	AXI byte strobes (must by all high, as we only support full word access)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	axis_data_width_g/8-1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_WVALID	
Description	AXI write data valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN

AXIS_WVALID	
Reset Value	0

AXIS_WREADY	
Description	AXI write data ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_BID	
Description	AXI transaction ID, ID width can be selected via device configuration parameter
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	axis_id_width_g DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_BRESP	
Description	AXI write response
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_BVALID	
Description	AXI write response valid
Loop	–
Condition	–

AXIS_BVALID	
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_BREADY	
Description	AXI wire response ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

AXIS_BUS_ERROR	
Description	ERROR occurred, can only be resolved with AXIS_axis_aresetnN
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	AXIS_ACLK
Special Function	–
Operational Reset	AXIS_ARESETN
Reset Value	0

3.18.5 GTM toplevel AXIM interface

Loop	–
Condition	–
Format	–

AXIM_ACLK	
Description	AXIM reference clock
Loop	–
Condition	–
Logical Name	–

AXIM_ACLK	
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	0

AXIM_ARESETN	
Description	AXIM reset signal (low active)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	AXIM_ACLK
Special Function	–
Operational Reset	–
Reset Value	0

AXIM_ARUSER	
Description	AXI read user data; one hot coded signal, the value AXIM_ARUSER[i:i] = 1 signals that the corresponding cluster i has initiated this read transaction
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	AXIM_ARESETN
Reset Value	0

AXIM_AWUSER	
Description	AXI write user data; one hot coded signal, the value AXIM_AWUSER[i:i]=1 signals that the corresponding cluster i has initiated this write transaction
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	AXIM_ARESETN

AXIM_AWUSER	
Reset Value	0

3.18.6 GTM toplevel TIM[i] signal interface

Loop	$i = \{n : 0 <= n <= \mathrm{NTIM} - 1\}$
Condition	–
Format	–

GTM_TIM[i]_IN	
Description	Input signals for TIM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.7 GTM toplevel TOM[i] signal interface

Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}$
Condition	–
Format	–

GTM_TOM[i]_OUT	
Description	Timer output signals for TOM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	15 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	TOM_OUT_RST

GTM_TOM[i]_OUT_N	
Description	Inverted timer output signals for TOM[i]
Loop	–
Condition	–

GTM_TOM[i]_OUT_N	
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	15 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	TOM_OUT_RST

3.18.8 GTM toplevel ATOM[i] signal interface

Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}$
Condition	–
Format	–

GTM_ATOM[i]_OUT	
Description	Timer output signals for ATOM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	ATOM_OUT_RST

GTM_ATOM[i]_OUT_N	
Description	Inverted timer output signals for ATOM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	ATOM_OUT_RST

3.18.9 GTM toplevel TOM[i] hres output interface

Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= \text{TOM\_HIGH\_RES}[i] - 1\}$

Condition	–
Format	–

GTM_TOM[i]_OUT_HRES-[x]_CLK	
Description	Output signal GTM_TOM[i]_OUT[x:x] high resolution clock
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_TOM[i]_OUT_HRES-[x]	
Description	Output signal GTM_TOM[i]_OUT[x:x] high resolution delay value
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_TOM[i]_OUT_N_H-RES[x]_CLK	
Description	Inverted output signal GTM_TOM[i]_OUT_N[x:x] high resolution clock
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_TOM[i]_OUT_N_H-RES[x]	
Description	Inverted output signal GTM_TOM[i]_OUT_N[x:x] high resolution delay value

GTM_TOM[i]_OUT_N_HRES[x]	
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.10 GTM toplevel ATOM[i] hres output interface

Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= \text{ATOM\_HIGH\_RES}[i] - 1\}$
Condition	–
Format	–

GTM_ATOM[i]_OUT_HRES[x]_CLK	
Description	Output signal GTM_ATOM[i]_OUT[x:x] high resolution clock
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_ATOM[i]_OUT_HRES[x]	
Description	Output signal GTM_ATOM[i]_OUT[x:x] high resolution delay value
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_ATOM[i]_OUT_N_HRES[x]_CLK	
Description	Inverted output signal GTM_ATOM[i]_OUT_N[x:x] high resolution clock
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_ATOM[i]_OUT_N_HRES[x]	
Description	Inverted output signal GTM_ATOM[i]_OUT_N[x:x] high resolution delay value
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.11 GTM toplevel TIO[i] signal interface

Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1$
Format	–

GTM_TIO[i]_G[g]_IN	
Description	Input signals for TIO[i]
Loop	$g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET

GTM_TIO[i]_G[g]_IN	
Reset Value	0

GTM_TIO[j]_G[g]_OUT	
Description	Output signals for TIO[i]
Loop	$g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_TIO[j]_G[g]_OUT_N	
Description	Inverted output signals for TIO[i]
Loop	$g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.12 GTM toplevel TIM[i] interrupt interface

Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}$
Condition	–
Format	–

GTM_TIM[i]_IRQ	
Description	Shared interrupt from module TIM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–

GTM_TIM[i]_IRQ	
Operational Reset	RESET
Reset Value	0

GTM_TIM[i]_IRQ_CLR	
Description	Interrupt clear to module TIM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.13 GTM toplevel TOM[i] interrupt interface

Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}$
Condition	–
Format	–

GTM_TOM[i]_IRQ	
Description	Shared interrupt from module TOM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_TOM[i]_IRQ_CLR	
Description	Interrupt clear to module TOM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–

GTM_TOM[i]_IRQ_CLR	
Operational Reset	RESET
Reset Value	0

3.18.14 GTM toplevel ATOM[i] interrupt interface

Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}$
Condition	–
Format	–

GTM_ATOM[i]_IRQ	
Description	Shared interrupt from module ATOM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_ATOM[i]_IRQ_CLR	
Description	Interrupt clear to module ATOM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	3 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.15 GTM toplevel DTM[i] signal interface

Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	–
Format	–

GTM_CDTM[i]_DTM[d]_AUX_IN	
Description	Input signals for DTM[d] in cluster [i].
Loop	–

GTM_CDTM[i]_DTM[d]_AUX_IN	
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.16 GTM toplevel MCS[i] interrupt interface

Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	–
Format	–

GTM_MCS[i]_IRQ	
Description	Interrupt from module MCS[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_MCS[i]_IRQ_CLR	
Description	Interrupt clear to module MCS[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_MCS[i]_S_IRQ	
Description	Shared interrupt from module MCS[i]
Loop	–

GTM_MCS[i]_S_IRQ	
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_MCS[i]_S_IRQ_CLR	
Description	Shared interrupt clear to module MCS[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_MCS[i]_S_IRQ_SET	
Description	Shared interrupt set to module MCS[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_MCS[i]_HBP_IRQ	
Description	Breakpoint interrupt from module MCS[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	CLK
Special Function	–

GTM_MCS[i]_HBP_IRQ	
Operational Reset	RESET
Reset Value	0

GTM_MCS[i]_HBP_IRQ_CLR	
Description	Breakpoint clear to module MCS[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.17 GTM toplevel PSM[i] interrupt interface

Loop	$i = \{n : 0 <= n <= \mathrm{NPSM} - 1\}$
Condition	–
Format	–

GTM_PSM[i]_IRQ	
Description	Shared interrupt from module PSM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_PSM[i]_IRQ_CLR	
Description	Interrupt clear to module PSM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–

GTM_PSM[i]_IRQ_CLR	
Operational Reset	RESET
Reset Value	0

3.18.18 GTM_AEI interrupt interface

Loop	–
Condition	–
Format	–

GTM_AEI_IRQ	
Description	Shared interrupt from module GTM_CTL
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_AEI_IRQ_CLR	
Description	Interrupt clear to module GTM_CTL
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.19 GTM toplevel ARU interrupt interface

Loop	$i = \{n : 0 <= n <= \mathrm{NARU} - 1\}$
Condition	–
Format	–

GTM_ARU_IRQ	
Description	Shared interrupt from module ARU[0:0]: ARU_NEW_DATA0_I Interrupt[1:1]: ARU_NEW_DATA1_I Interrupt[2:2]: ARU_ACC_ACK_I Interrupt
Loop	–

GTM_ARU_IRQ	
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_ARU_IRQ_CLR	
Description	Interrupt clear to module ARU
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	2 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.20 GTM toplevel BRC interrupt interface

Loop	$i = \{n : 0 <= n <= \mathrm{NBRC} - 1\}$
Condition	–
Format	–

GTM_BRC_IRQ	
Description	Shared interrupt from module BRC
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_BRC_IRQ_CLR	
Description	Interrupt clear to module BRC
Loop	–

GTM_BRC_IRQ_CLR	
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.21 GTM toplevel CMP interrupt interface

Loop	$i = \{n : 0 <= n <= \mathrm{NCMP} - 1\}$
Condition	–
Format	–

GTM_CMP_IRQ	
Description	Shared interrupt from module CMP
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_CMP_IRQ_CLR	
Description	Interrupt clear to module CMP
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.22 GTM toplevel DPLL interrupt interface

Loop	$i = \{n : 0 <= n <= \text{NDPLL} - 1\}$
Condition	–
Format	–

GTM_DPLL_IRQ	
Description	Shared interrupt from module DPLL[0:0]: DPLL_DCGI: DPLL direction change interrupt[1:1]: DPLL_PDI or DPLL_PEI; DPLL enable or disable interrupt[2:2]: DPLL_TINI: DPLL TRIG. min. hold time (THMI) viol. detected[3:3]: DPLL_TAXI: DPLL TRIG. max. hold time (THMA) viol. detected[4:4]: DPLL_SISI: DPLL STATE inactive slope detected[5:5]: DPLL_TISI: DPLL TRIGGER inactive slope detected[6:6]: DPLL_MSI: DPLL Missing STATE interrupt[7:7]: DPLL_MTI: DPLL Missing TRIGGER interrupt[8:8]: DPLL_SASI: DPLL STATE active slope detected[9:9]: DPLL_TASI: DPLL TRIG. active slope det. while DPLL_NTI_CNT.NTI_CNT is 0[10:10]: DPLL_PWI: DPLL Plausibility window (PVT) viol. int. of TRIG.[11:11]: DPLL_W2I: DPLL Write access to RAM region 2 interrupt[12:12]: DPLL_W1I: DPLL Write access to RAM region 1b or 1c int.[13:13]: DPLL_GL1I: DPLL Get of lock interrupt for SUB_INC1[14:14]: DPLL_LL1I: DPLL Lost of lock interrupt for SUB_INC1[15:15]: DPLL_EI: DPLL Error interrupt[16:16]: DPLL_GL2I: DPLL Get of lock interrupt for SUB_INC2[17:17]: DPLL_LL2I: DPLL Lost of lock interrupt for SUB_INC2[18:18]: DPLL_TE0I: DPLL TRIGGER event interrupt 0[19:19]: DPLL_TE1I: DPLL TRIGGER event interrupt 1[20:20]: DPLL_TE2I: DPLL TRIGGER event interrupt 2[21:21]: DPLL_TE3I: DPLL TRIGGER event interrupt 3[22:22]: DPLL_TE4I: DPLL TRIGGER event interrupt 4[23:23]: DPLL_CDTI; DPLL calculated duration interrupt for TRIGGER[24:24]: DPLL_CDSI; DPLL calculated duration interrupt for STATE[25:25]: DPLL_TORI; TRIGGER out of range interrupt[26:26]: DPLL_SORI; STATE out of range interrupt
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	26 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_DPLL_IRQ_CLR	
Description	Interrupt clear to module DPLL
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	26 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.23 GTM toplevel SPE[i] interrupt interface

Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	–
Format	–

GTM_SPE[i]_IRQ	
Description	Shared interrupt from module SPE[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_SPE[i]_IRQ_CLR	
Description	Interrupt clear to module SPE[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.24 GTM toplevel TIO[i] interrupt interface

Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1$
Format	–

GTM_TIO[i]_G[g]_IRQ	
Description	Shared interrupt from module TIO[i]
Loop	$g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_TIO[j]_G[g]_IRQ_CLR	
Description	Interrupt clear to module TIO[i]
Loop	$g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.25 GTM toplevel ERR interrupt interface

Loop	–
Condition	–
Format	–

GTM_ERR_IRQ	
Description	Error interrupt from module ICM
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

GTM_ERR_IRQ_CLR	
Description	Error interrupt clear to module ICM
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.26 GTM toplevel CMU signal interface

Loop	–
Condition	–
Format	–

GTM_CMU_ECLK	
Description	Clock output signal from CMU
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0
Operational Clock	CLK
Special Function	–
Operational Reset	RESET
Reset Value	0

3.18.27 GTM Halt Interface

Loop	–
Condition	–
Format	–

GTM_HALT_REQ	
Description	Control signal which requests that GTM-IP operation has to be halted
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	–
Reset Value	0

GTM_HALT_ACTIVE	
Description	Signal indicates that GTM-IP operation is halted
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–

GTM_HALT_ACTIVE	
Operational Reset	–
Reset Value	–

GTM_RESTORE	
Description	Indicates restore phase of functional memory accesses when leaving GTM–IP halt state and continue with regular operation
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLK
Special Function	–
Operational Reset	–
Reset Value	–

3.19 GTM TOP–Level Signal Description

3.19.1 GTM TOP–Level Interrupt Signals

Loop	–
Condition	–
Format	–

AEIM_IM_ADDR_IRQ	
Description	Interrupt signal MCS AEI master illegal Module address access
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEIM_IM_ADDR_EIRQ	
Description	Error interrupt signal MCS AEI master illegal Module address access
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEIM_USP_ADDR_IRQ	
Description	Interrupt signal MCS AEI master unsupported address
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEIM_USP_ADDR_EIRQ	
Description	Error interrupt signal MCS AEI master unsupported address

AEIM_USP_ADDR_EIRQ	
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEIM_USP_BE_IRQ	
Description	Interrupt signal MCS AEI master unsupported byte enable
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEIM_USP_BE_EIRQ	
Description	Error interrupt signal MCS AEI master unsupported byte enable
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEI_IM_ADDR_IRQ	
Description	Interrupt signal AEI illegal Module address access
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEI_IM_ADDR_EIRQ	
Description	Error interrupt signal AEI illegal Module address access
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEI_TO_XPT_IRQ	
Description	Interrupt signal AEI timeout exception occurred.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEI_TO_XPT_EIRQ	
Description	Error interrupt signal AEI timeout exception occurred.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEI_USP_ADDR_IRQ	
Description	Interrupt signal AEI unsupported address
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEI_USP_ADDR_EIRQ	
Description	Error interrupt signal AEI unsupported address
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEI_USP_BE_IRQ	
Description	Interrupt signal AEI unsupported byte enable
Loop	–
Condition	–
Signal Type	INT
Assignment	–

AEI_USP_BE_EIRQ	
Description	Error interrupt signal AEI unsupported byte enable
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CLK_EN_ERR_IRQ	
Description	Interrupt signal clock enable error
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CLK_EN_ERR_EIRQ	
Description	Error interrupt signal clock enable error
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CLK_PER_ERR_IRQ	
Description	Interrupt signal clock period error
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CLK_PER_ERR_EIRQ	
Description	Error interrupt signal clock period error
Loop	–
Condition	–
Signal Type	INT
Assignment	–

3.19.2 CLS[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	–
Format	–

CLS[j]_CLK_ENABLE	
Description	Cluster [j] clocks are enabled; physical netgtm_core_i/gtm_cls[j]_i/en_clk_zz
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	GTM_CLS_CLK_CFG.CLS[j]_CLK_DIV != 0 && (INT_CLK_EN == 1) && (CGATE_TE == 0)

CLS[j]_CLK	
Description	Cluster [j] clock signal
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	–

3.19.3 CLK_EN

Loop	–
Condition	–
Format	–

INT_CLK_EN	
Description	Internal clock enable signal
Loop	–
Condition	–
Signal Type	INT
Assignment	((INT_CLK_EN_GEN == 0) && CLK_EN) || (INT_CLK_EN_GEN == 1)

3.19.4 RF_PROT AND RESETS

Loop	–
Condition	–
Format	–

RF_PROT	
Description	rf_prot signal; physical netgtm_core_i/rf_prot_zz

RF_PROT	
Loop	–
Condition	–
Signal Type	INT
Assignment	GTM_CTRL.RF_PROT

HW_RESET	
Description	GTM global reset active signal for clk clock domain; physical netgtm_core_i/reset_int_zz
Loop	–
Condition	–
Signal Type	INT
Assignment	(RESET == RESET_ACTIVE)

GTM_RESET	
Description	GTM global reset active signal; physical netgtm_core_i/reset_int_zz
Loop	–
Condition	–
Signal Type	INT
Assignment	(GTM_RST.RST == 1 && GRSTEN == 1 && SCAN_MODE == 0)

HW_CPU_IF_RESET	
Description	Reset active signal for CPU interface clock domain; physical net
Loop	–
Condition	–
Signal Type	INT
Assignment	((USE_AXIS == 0) && (AEI_RESET == RESET_ACTIVE)) || ((USE_AXIS == 1) && (AXIS_ARESETN == RESET_ACTIVE))

GTM_AEI_RESET	
Description	Software reset of gtm_aei bridge; physical net
Loop	–
Condition	–
Signal Type	INT
Assignment	(BRIDGE_MODE.BRG_RST == 1)

3.19.5 CLS[j]_AEI_ARB

Loop	$j = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	–
Format	–

CLS[j]_AEI_ARB_STATE	
Description	Cluster [j] AEI bus arbiter state: 0=MCS; 1=CPU
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	CLS_CPU_AEI_SEL[j] && !CLS_MCS_AEIM_SEL[j]

CLS[j]_AEI_ARB_WDATA	
Description	Cluster [j] AEI bus write data of arbiter; physical netgtm_core_i/gtm_cls[j]_i/arb_aei_wdata_zz
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	((CLS[j]_AEI_ARB_STATE == 0) * CLS[j]_MCS_AEIM_WDATA) + ((CLS[j]_AEI_ARB_STATE == 1) * CLS[j]_CPU_AEI_WDATA)

CLS[j]_CPU_AEI_WDATA	
Description	Cluster [j] AEI CPU bus write data; physical netgtm_core_i/gtm_cls[j]_i/aei_wdata_cls
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	0

CLS[j]_MCS_AEIM_WDATA	
Description	Cluster [j] AEI MCS bus write data; physical netgtm_core_i/gtm_cls[j]_i/mcs_wrapper_i/aeim_wdata_zz
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	0

CLS_CPU_AEI_SEL[j]	
Description	Cluster [j] CPU bus read/write access active; physical net
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	0

CLS_MCS_AEIM_SEL[j]	
Description	Cluster [j] MCS bus read/write access active; physical net
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	0

3.19.6 MCS_AEIM Signals

Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	–
Format	–

CCM[i]_AEI_STATUS	
Description	AEI access status driven by MCS_AEIM in Cluster [i]
Loop	–
Condition	–
Signal Type	ARRAY[2]

CCM[i]_AEI_STATUS	
Assignment	–

3.19.7 Top Level Internal Signals

Loop	–
Condition	–
Format	–

TIM[i]_AUX_IN	
Description	Selected source for module TIM[i] AUX_IN
Loop	$i = \{n : 0 <= n <= \mathrm{NTIM} - 1\}$
Condition	–
Signal Type	ARRAY[NTIM]
Assignment	–

TIM[i]_EXT_CAPTURE	
Description	Signal vector: driven by output port TIM_EXT_CAPTURE of TIM instance i channel x:={0, 1,..., 7}
Loop	$i = \{n : 0 <= n <= \mathrm{NTIM} - 1\}$
Condition	–
Signal Type	ARRAY[NTIM]
Assignment	–

CLS_DIS	
Description	Each bit i signals if corresponding cluster i is disabled from ARU communication
Loop	–
Condition	–
Signal Type	INT
Assignment	–

3.19.8 Interrupt unit internal signals

Loop	–
Condition	–
Format	–

HW_CLEAR	
Description	Interrupt unit: Hardware request to clear any registered interrupt
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SW_CLEAR	
Description	Interrupt unit: Software request to clear any registered interrupt
Loop	–
Condition	–

SW_CLEAR	
Signal Type	INT
Assignment	–

IRQ_BIT	
Description	Interrupt unit: Interrupt output signal, behaving as defined by corresponding IRQ mode
Loop	–
Condition	–
Signal Type	INT
Assignment	–

IRQ_OCCURRED_BIT	
Description	Interrupt unit: Interrupt output signal, 1 is signaling that an interrupt occurred
Loop	–
Condition	–
Signal Type	INT
Assignment	–

IRQ_LINE	
Description	Interrupt unit: Bundled output signal which combines multiple IRQ_BIT signals
Loop	–
Condition	–
Signal Type	INT
Assignment	–

IRQ_OCCURRED	
Description	Interrupt unit: Bundled output signal which combines multiple IRQ_OCCURRED_BIT signals
Loop	–
Condition	–
Signal Type	INT
Assignment	–

INT_IN	
Description	Interrupt priority decoding: Interrupt event input
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CLEAR_IN	
Description	Interrupt priority decoding: Interrupt clear event input
Loop	–
Condition	–
Signal Type	INT
Assignment	–

INT_OUT	
Description	Interrupt priority decoding: Interrupt event output

INT_OUT	
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CLEAR_OUT	
Description	Interrupt priority decoding: Interrupt clear event output
Loop	–
Condition	–
Signal Type	INT
Assignment	–

3.19.9 Error Interrupt Unit Internal Signals

Loop	–
Condition	–
Format	–

EIRQ_BIT	
Description	Error interrupt unit: Interrupt output signal, behaving as defined by corresponding IRQ mode
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EIRQ_OCCURRED_BIT	
Description	Error interrupt unit: Interrupt output signal, 1 is signaling that an interrupt occurred
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EIRQ_LINE	
Description	Error interrupt unit: Bundled output signal which combines multiple EIRQ_BIT signals
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EIRQ_OCCURRED	
Description	Error interrupt unit: Bundled output signal which combines multiple EIRQ_OCCURRED_BIT signals
Loop	–
Condition	–
Signal Type	INT
Assignment	–

4 AXI Master

4.1 Functional Characteristics

The AXI transaction generator implements multiple AEI slave interfaces and an AXI compliant master interface. It includes the necessary registers, arbitration and bus state machines.

- AEI standard protocol compliant.
- 32–bit AEI interface.
- Zero wait state register access.
- Max 10 AEI slave interfaces.
- Max 16 AXI transaction slots per AEI slave (configurable via device configuration parameter AXIM).
- One common register set per AEI interface.
- Full AMBA AXI 3.0 compliant.
- 32–bit AXI address range support.
- 32 or 64–bit AXI data width (configurable via device configuration variable AXIM_DATA_WIDTH; chapter *4.2.3.5.1 "64bit AXI data width support"*).
- Single accesses only (length is fixed to 1).
- All other AXI signals are programmable.
- ID width configurable (configurable via device configuration variable AXIM_ID_WIDTH; chapter *85 "GTM Device Configuration Variables, Defined by SOC Integration"*).
- Support of AXI *ARUSER* and *AWUSER* signals to indicate, which cluster of the GTM is initiating the transaction.
- One "interrupt" line per slot.

Indices as used inside this chapter are:

- NAXIM: number of AXIM instances in the device
- i:={0, 1,..., (NAXIM–1)} instance index of MCS module, which requests AXI transfers
- s:={0, 1,..., (AXIM–1)} index for slots

4.2 AXI Transaction Generator Block

4.2.1 Functional View

Figure 27 Block Diagram

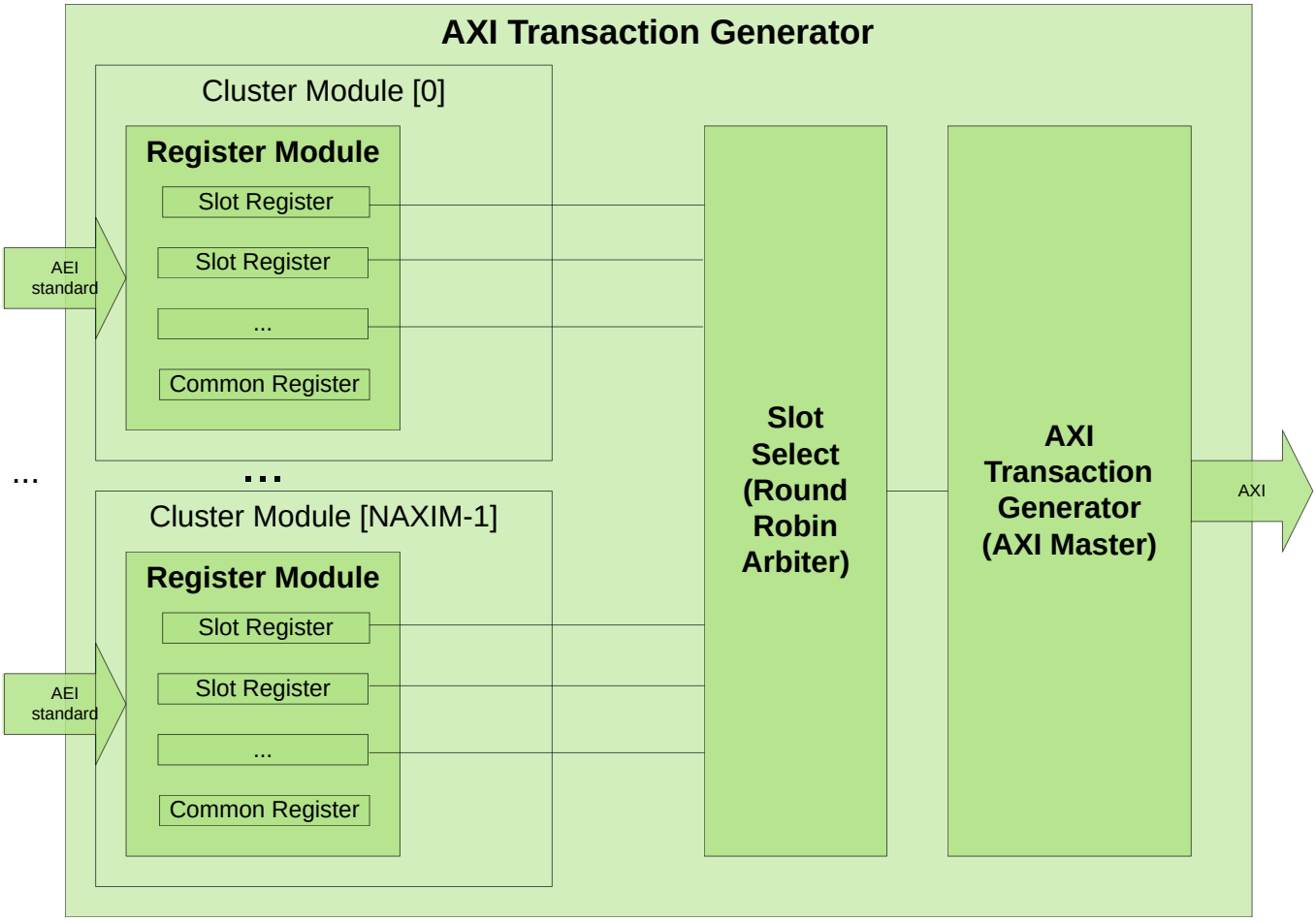


Figure *27 "Block Diagram"* shows the basic modules of the AXI Transaction Generator block. For each AEI interface one cluster module is implemented. This cluster module includes the register module. The transactions are then arbitrated and finally used to create the bus transactions.

4.2.2 Block Entity Table

The GTM-IP has a configurable number of AEI standard slave interfaces on the "left" side and an AXI master interface on the "right" side.

4.2.3 Detailed Architecture

4.2.3.1 Cluster Module

The cluster module holds all components which belong to one cluster of the GTM. It is intended to ease the layout separation of the different components.
Each cluster implements a separate clock input, but clocks of all clusters and AXI port clock must fulfill synchronous timing relationships. Cluster clocks must be generated from the same base clock as AXI port clock (*ACLK*), but some clock cycles may be gated. Therefore, the clock rate for each cluster is different:
AXIM_CLK (cluster) = *ACLK* /n (n∈{1, 2,..., x} (typical x=2))

Signal synchronization between *AXIM_CLK* and *ACLK* domains is supported by the *ENCLK* inputs. *ENCLK* (cluster)="1" will identify active cluster clock edges to *ACLK* domain logic.

Cluster clocks may be disabled completely for longer time frames (e.g.: debugging). A disabled cluster clock is signaled by *CLKOFF* (cluster) input signal. This signal overrules *ENCLK* (cluster) for *AXIM_CLK* / *ACLK* synchronization logic. Otherwise, synchronization logic may hang if *AXIM_CLK* is disabled.
After a clock off phase, cluster state is reset. Therefore, *CLKOFF* must be active for at least one active clock cycle when clock off state is released.
When one cluster leaves clock off state, system software must ensure that no active AXI cycle is pending for any slot of this cluster. Otherwise,

AXI responses from requests issued before entering clock off state may collide with newer slot requests issued after release of clock off state.

4.2.3.2 Register Module

For each AEI slave interface one register module is instantiated. The register module is built from two building blocks, the slot registers, where one set is implemented for each available slot and a common register area where the slot allocation takes place.
All registers provide 1 cycle accesses; this means *AEIM_AXIM_READY* must be asserted in the same cycle as *AEIM_AXIM_SEL* is asserted. This is required for correct MCS operation.
In debug mode (*AEIM_AXIM_DEBUG_ACCESS* =1), read accesses to registers must not have any side effects:

- A debug read to the data register doesn't start a new AXI read cycle in auto-increment mode.
- A debug read to the **AXIM[i]_REQUEST** register doesn't allocate a new slot.

4.2.3.2.1 Slot Registers

All information needed for the AXI transaction is stored in the slot registers. The goal is to use as less registers as possible to reduce the number of required accesses by the controlling program. The slot register block also includes a state machine to communicate with the following arbiter stage and the AXI transaction generator.

The register **AXIM[i]_SLOT[s]_ADDR_LOW** supports addressing of a 32bit AXI address range. The data register **AXIM[i]_SLOT[s]_DATA_LOW** holds the write data for a write transaction and the read data for a read transaction after the transaction has been completed.
Two configuration registers exist. The first holds the AXI transaction information, the second the ID and the write strobes.
If configured, the block supports posted writes. This means a write transaction is successfully executed after the write data have been accepted by the slave (the NOC).
The corresponding write response is ignored. In the module the selection posted or not posted is done by the LSB of the transaction ID. A one (1) means the transaction is posted, a transaction that is not posted, must have a zero (0) LSB.
The status register shows the state of the slot:

- **AXIM[i]_SLOT[s]_STATUS.ALLOC** The slot is allocated
- **AXIM[i]_SLOT[s]_STATUS.QUEUED** A transaction has been queued for execution
- **AXIM[i]_SLOT[s]_STATUS.STARTED** The transaction is accepted by the arbiter
- **AXIM[i]_SLOT[s]_STATUS.READY** The AXI transaction has been executed

In the **AXIM[i]_SLOT[s]_STATUS.RESP** field, the response from the bus is stored. In case of an invalid AXI request or response (see section *4.2.3.4 "Transaction Generator"*), the response from the AXI bus is overruled and set to "11" by the AXI Transaction Generator. Invalid AXI requests are never sent out to the AXI bus by the AXI Transaction Generator.

Via a write operation to the release register the slot can be de-allocated.

Figure 28 Slot Registers

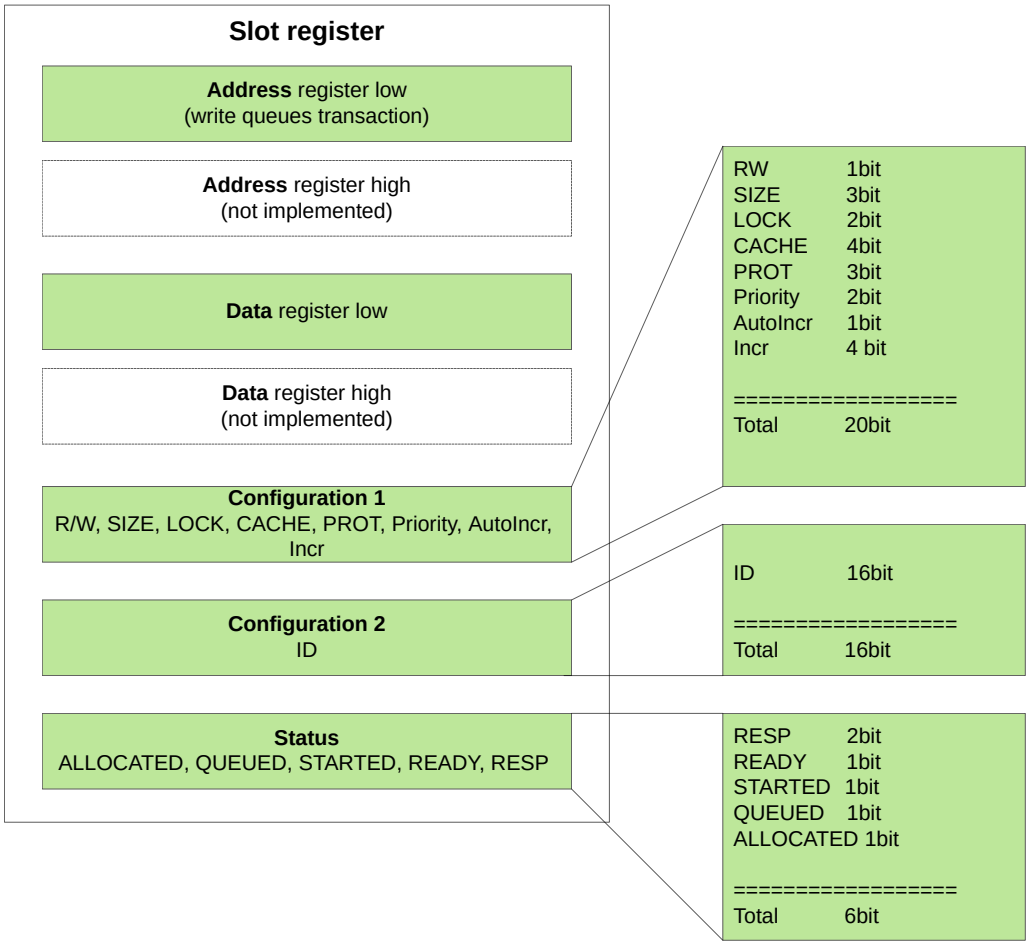
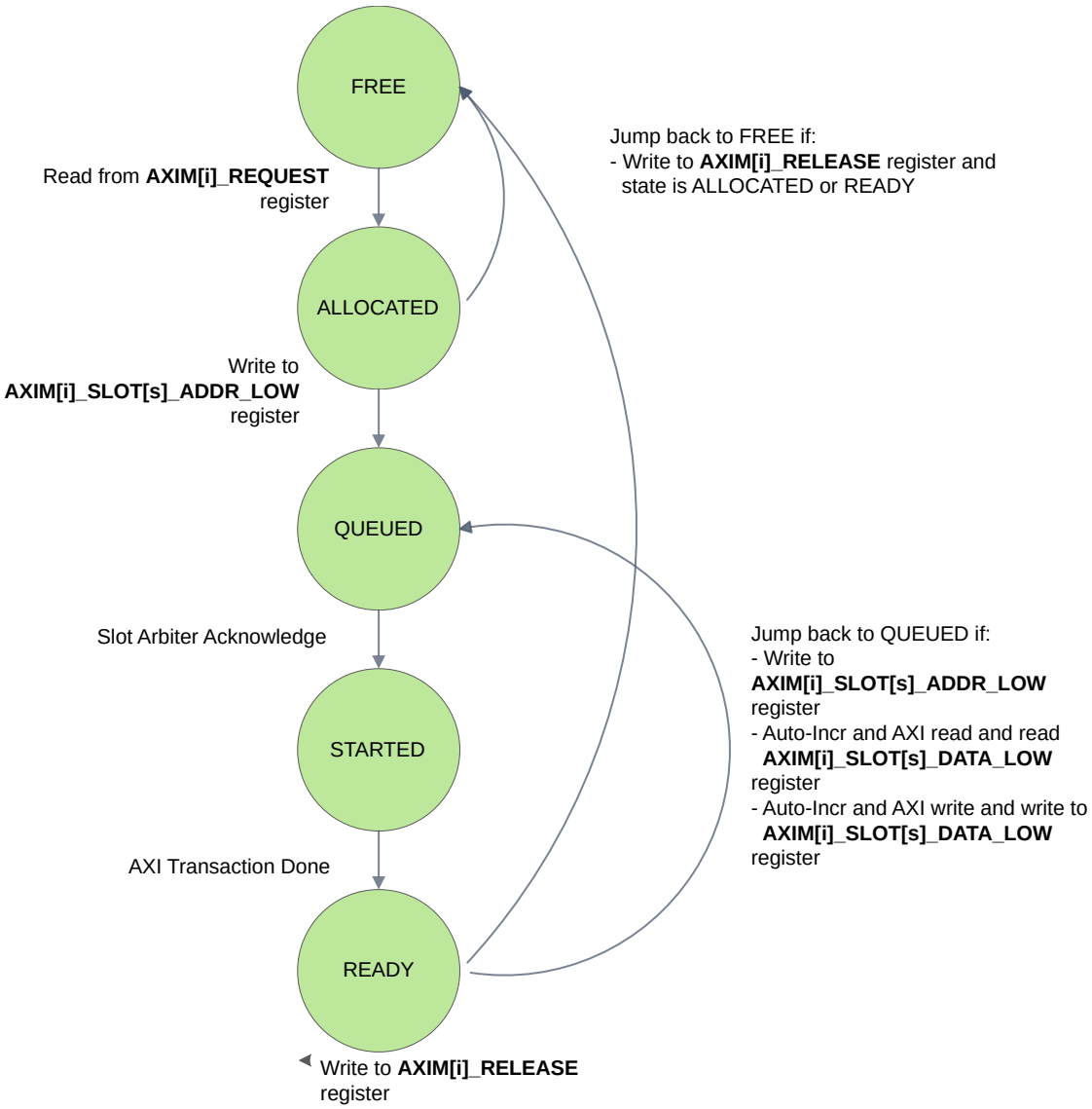


Figure 29 Slot State Machine



Example for a read transaction

1. Allocate a slot.
2. Write the configuration registers.
3. Write the address register, this transfers the slot into the **AXIM[i]_SLOT[s]_STATUS.QUEUED** state.
4. Wait for **AXIM[i]_SLOT[s]_STATUS.READY** interrupt or poll status register.
5. Check **AXIM[i]_SLOT[s]_STATUS.RESP** value (if required).
6. Clear **AXIM[i]_SLOT[s]_STATUS.READY** interrupt request by asserting the external input line *AEIM_AXIM_IRQ_CLEAR* for this slot.
7. Fetch read data from the Data Register: (a) If auto-increment is enabled this already triggers the next transaction. Continue with step 4. (b) If auto-increment is disabled, state doesn't change. (c) If this is the last loop of an auto-increment procedure, software should clear the "auto_incr" bit inside configuration 1 register, before reading the data register.
8. Continue with: (a) Release slot by writing to **AXIM[i]_RELEASE** register; Continue with step 1. (b) Start a new read cycle by writing a new address to the **AXIM[i]_SLOT[s]_ADDR_LOW** register; Continue with step 4.

Example for a write transaction

1. Allocate a slot.
2. Write the configuration registers.
3. Write the data into the data register.
4. Write the address register, this transfers the slot into the **AXIM[i]_SLOT[s]_STATUS.QUEUED** state.
5. Wait for **AXIM[i]_SLOT[s]_STATUS.READY** interrupt or poll status register.

6. Check **AXIM[i]_SLOT[s]_STATUS.RESP** value (if required).
7. Clear **AXIM[i]_SLOT[s]_STATUS.READY** interrupt request by asserting the external input line *AEIM_AXIM_IRQ_CLEAR* for this slot.
8. Continue with: (a) If auto-increment is enabled, write of new data to the data register starts a new AXI cycle with incremented address; continue with step 5. (b) Start a new write cycle by writing a new address to the **AXIM[i]_SLOT[s]_ADDR_LOW** Register; continue with step 5. (c) Release the slot by writing to the **AXIM[i]_RELEASE** register; continue with step1.

4.2.3.2.2 Interrupt Handling

Each slot provides one interrupt line to the corresponding MCS (*AEIM_AXIM_IRQ*). This interrupt line is set when the slot state changes from **AXIM[i]_SLOT[s]_STATUS.STARTED** to **AXIM[i]_SLOT[s]_STATUS.READY** .

The interrupt is cleared by asserting the external input line *AEIM_AXIM_IRQ_CLEAR* for this slot.

4.2.3.2.3 Auto Increment

The module offers an address auto increment feature. If it is enabled, the address (inside the **AXIM[i]_SLOT[s]_ADDR_LOW** register) is incremented automatically after each transaction according to the **AXIM[i]_SLOT[s]_CFG1.INCR** bit field:
New_ADDR <= **AXIM[i]_SLOT[s]_ADDR_LOW** + **AXIM[i]_SLOT[s]_CFG1.INCR**

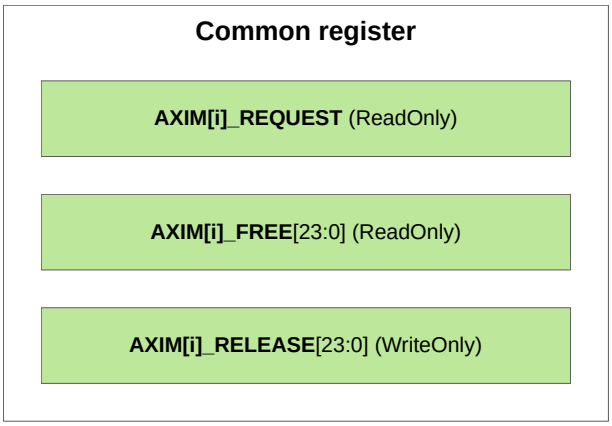
To disable auto-increment, the auto-increment bit **AXIM[i]_SLOT[s]_CFG1.AUTO_INCR** has to be cleared.

There is one auto-increment adder per cluster. Each slot requests auto-increment processing from this shared resource whenever the last transaction has finished. (Slot state changed from **AXIM[i]_SLOT[s]_STATUS.STARTED** to **AXIM[i]_SLOT[s]_STATUS.READY**). The new address is available inside the address register two clock cycles later.

4.2.3.2.4 Common Registers

The common registers are used for slot management. If an MCS thread wants to allocate a transaction slot, it reads the **AXIM[i]_REQUEST** register. If a slot is available, the return value is the number of the allocated slot in two coding's. The lower 24 bits are one-hot coded, each bit representing one slot. The upper 8 bits are the slot numbers allocated in binary format. If no slot was available the maximum value of 0xFF is returned, the lower bytes are then 0x00 0x00 0x00.
To release a slot, MCS must write a "1" to the corresponding bit in the **AXIM[i]_RELEASE** register.
Via the **AXIM[i]_FREE** register it is possible to check which slots are still free. One bit exists for each implemented slot.

Figure 30 Common Registers



4.2.3.3 Arbitration Scheme

The slot arbiter should balance all slots in an equal manner. To achieve an equal load balancing over all slots, the arbiter must process all slots in parallel (flat). Also, each slot request can be assigned to a priority level between 0 and 3 and the arbiter must select slots depending on this priority level.
The slot arbiter only accepts new slot requests while input signal *AXIM_GTM_HALT* ="0". Slot requests already accepted when *AXIM_GTM_HALT* changes from "1" to "0" are still forwarded to the AXI transaction generator. Therefore, after asserting *AXIM_GTM_HALT* up to two pending AXI transactions may still be issued to the AXI port.

Figure 31 Arbiter implementation

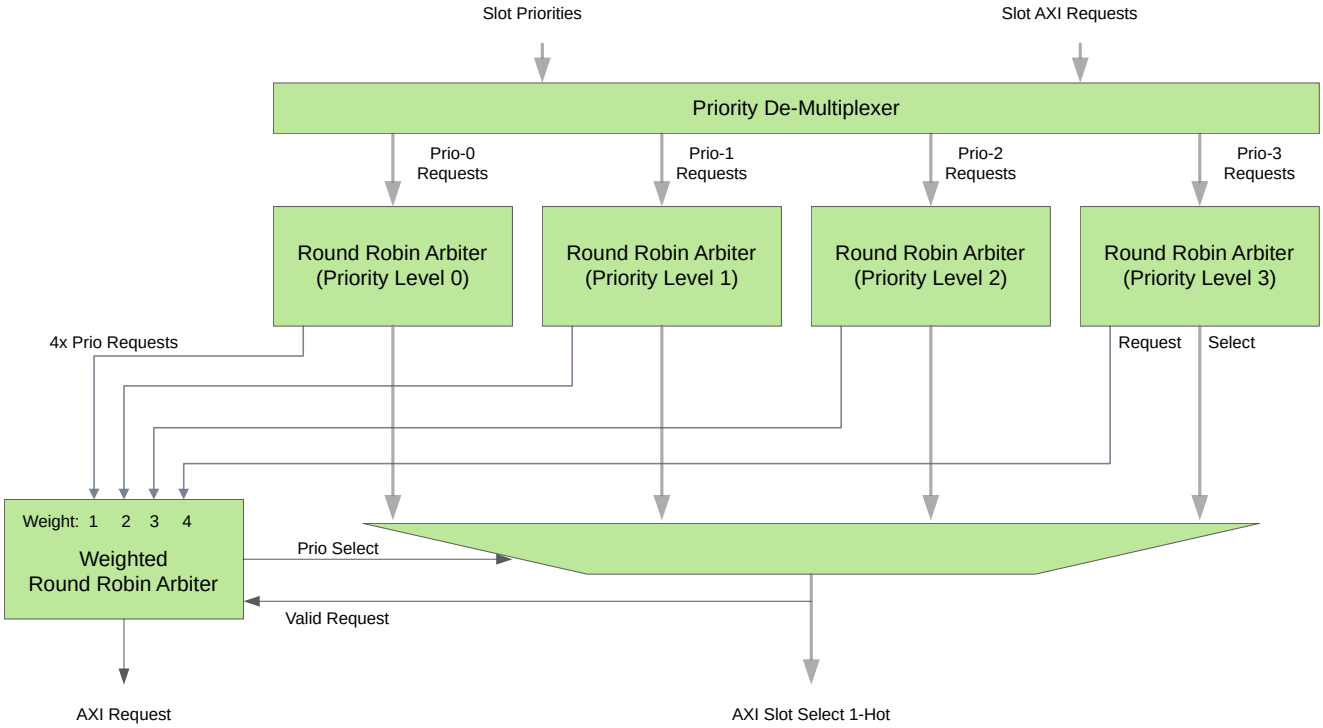
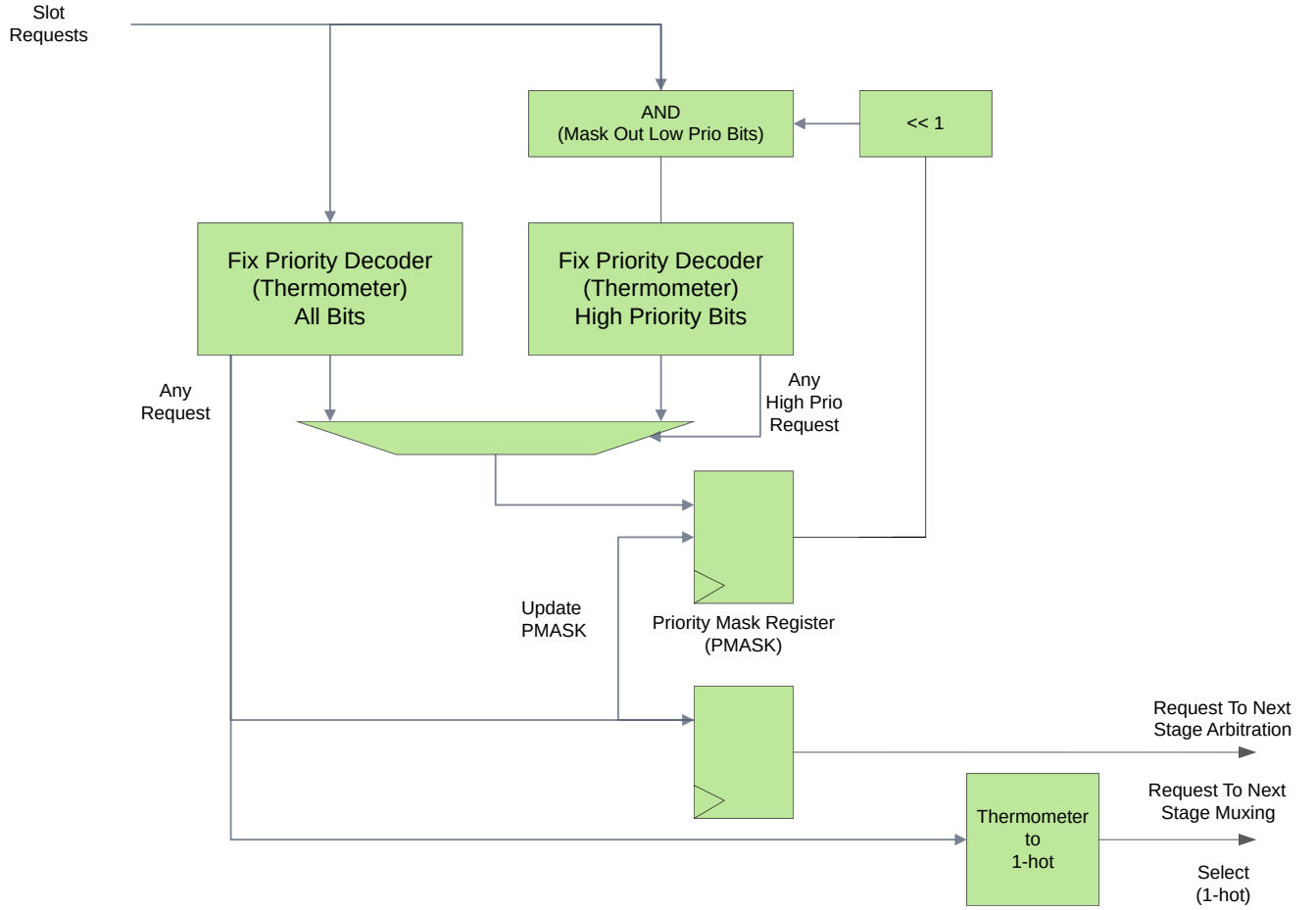


Figure *31 "Arbiter implementation"* shows the structure of the arbiter implementation. First, slot requests are separated into request groups for each priority class. Each priority class implements a unique flat Round Robin Arbiter (RRA). The common requests from these four arbiters are routed to a weighted Round Robin Arbiter. The weight for each input of the weighted RRA is fixed and represents the priority of the priority class RRA.
The working scheme for the weighted RRA is based on credits. Whenever the weighted RRA chose a new input port, it assigns a number of credits to this port. The number of initial credits represents the weight of the port. With each acknowledged request one credit is removed. If no more credits are left, the weighted RRA selects the next input port. This new input port is selected by a standard Round Robin scheme. If the currently selected port has still credits left, but no more pending requests, the weighted RRA also switches to the next input port. If none of the four first stage RRAs has any pending request, the weighted RRA keeps the currently selected port.
For the sake of timing, the first stage RRAs must register their request signals to the second stage RRA. This results in an extra clock cycle for arbitration, whenever the weighted RRA must select a new input port. But as long as the weighted RRA keeps the selected port, it issues a new request to the AXI Transaction Generator at every second clock cycle. Furthermore, as the select signal from the currently arbitrated first stage, RRA is visible to the second stage RRA one cycle before the request signals, even a new transfer of a currently arbitrated priority level that still has credits available can be accepted and prevent the re-arbitration if it occurs one cycle before the second stage RRA otherwise would select the next first stage RRA.

4.2.3.3.1 Round Robin Arbiter

The Round Robin Arbiter for each priority class must be configurable for the actual number of slots. For maximum GTM configurations this is up to 80 inputs (10 x 8).

Figure 32 Round Robin Arbiter



The chosen architecture for the RRA is shown in figure *32 "Round Robin Arbiter"* and is based on thermometer encoded fixed priority decoders. Thermometer encoded means that all bits starting from the first active request input up to the most significant bit of the encoder are set to "1". This is achieved by a simple OR daisy chain over all request inputs (from LSB to MSB).
The RRA implements two of these fixed priority decoders: one for all bits and one for high priority bits only. The high priority bits are the bits starting after the last arbitration winner up to the MSB. Due to thermometer encoding, these bits are simply selected by masking the input request signals with the left shifted result of last saved decoder output (PMASK). The final decoder result is the result of the high bits priority decoder – if this one has any match. Otherwise, the result of the all bit fixed priority decoder.
Every time a request is accepted, the result of last arbitration is saved in thermometer encoded format inside the PMASK register. This register serves two purposes: first it masks the high priority input requests (see above) and second it shows the last arbitration winner. To decode the arbitration winner, the content of the PMASK register is transformed from thermometer to 1–hot encoding. This transformation is simple. It is achieved by an AND operation of the PMASK register with the left shifted and afterwards inverted content of the same PMASK register (also known as the first 1 from LSB in the direction of MSB).
Public available analysis shows that thermometer encoded Round Robin arbiters give reasonably good results even for high numbers of request inputs. Nevertheless, this arbiter operates with up to nearly 80 inputs. For this high number of inputs, it is necessary to pipeline the fixed priority decoders to achieve timing constraints. In principle this is possible, but not implemented for the configurable VHDL block.

4.2.3.4 Transaction Generator

The Transaction Generator creates the actual AXI bus transaction. It applies the transaction values to the right AXI lines and observes the bus until the transaction has been completed. If available, the next transaction is immediately started.
The Transaction Generator has output as well as input storage elements to the AXI bus to ease the timing towards the NOC. Due to the pipelining of AXI handshake signals, the Transaction Generator only initiates new transaction every second clock cycle. This is no limitation, because also the arbiter delivers new requests with a maximum rate of every second clock cycle.
The Transaction Generator checks the parameters for every requested AXI transaction for validity. If an invalid request is detected, the AXI Transaction Generator doesn't send a request to the AXI bus, instead it directly responds to the requesting slot with an error response code (0b11).

Invalid transactions are:

1. Secure AXI cycle requested (**AXIM[i]_SLOT[s]_CFG1.AXI_PROT** [1:1] == 0) and VHDL constant "AXIM_SEC_ACC" is set to false (default). In systems supporting ARM Trustzone technology, secure AXI bus cycles are reserved for the so called "Secure Monitor" software. Other bus master like GTM doesn't issue secure bus cycles.
2. Privileged AXI cycle requested (**AXIM[i]_SLOT[s]_CFG1.AXI_PROT** [0:0] == 1) and VHDL constant "axim_priv_acc_c" is set to false. If privileged bus cycles are allowed, depends on application. By default "axim_priv_acc_c" is set to true.

3. LOCK= 0b11 (reserved value) or LOCK= 0b10 (Hard Lock): An AXI cycle with LOCK= 0b10 is a hard lock. It locks all other bus cycles on the NOC interconnect until the bus lock is removed with an unlock bus cycle. This is dangerous. To avoid this, GTM must only use "soft locks" or "exclusive access locks" (LOCK= 0b01).
4. AXI cycles with **CCM[i]_ARP[a]_CTRL.SIZE** greater 2 (greater 4 byte): AXI Transaction Generator only supports transactions not larger that the external data bus width, which is currently fixed to 32-bit.
5. Misaligned 2-byte bus read cycles: If *ARSIZE* is configured for 2 bytes (0b001), the address must be aligned to 2 bytes (*ARADDR* [0:0]=0).
6. Misaligned 2-byte bus write cycles: If *AWSIZE* is configured for 2 bytes (0b001), the address must be aligned to 2 bytes (*AWADDR* [0:0]=0).
7. Misaligned 4-byte bus read cycles: If *ARSIZE* is configured for 4 bytes (0b010), the address must be aligned to 4 bytes (*ARADDR* [1:0]= 0b00).
8. Misaligned 4-byte bus write cycles: If *AWSIZE* is configured for 4 bytes (0b010), the address must be aligned to 4 bytes (*AWADDR* [1:0]= 0b00).

4.2.3.4.1 Transaction initiator indication

The AXIM module supports up to NAXIM clusters, each cluster i ∈ {0, ..., NAXIM-1} initiates AXIM transactions. The AXI sideband signals *ARUSER* / *AWUSER* indicate the cluster i which initiated the transaction.

These AXI sideband signals are in operation in the read / write address phase of a transaction and fulfill the same timing requirements as defined in the AXI protocol specification for other signals (e.g. AXIM_ARID / AXIM_AWID).

Each transaction request is setup in the Cluster Module Slot registers (e.g. **AXIM[i]_SLOT[s]_CFG1**) (see *27 "Block Diagram"*) and arbitrated in the Slot Select module based on rules defined in *4.2.3.3 "Arbitration Scheme"* . The information, which cluster_id i has initiated the arbitrated slot, is preserved and handed over to AXI Transaction Generator (AXI Master).

This cluster_id i is used to generate the one hot coded signals *ARUSER* / *AWUSER* dependent on the transaction type read or write in the following way:
Read transaction: FOR j IN {0,..., NAXIM-1} IF j==i THEN *ARUSER* [j:j]=1 ELSE *ARUSER* [j:j]=0
Write transaction: FOR j IN {0,..., NAXIM-1} IF j==i THEN *AWUSER* [j:j]=1 ELSE *AWUSER* [j:j]=0

The GTM top level ports *AXIM_ARUSER* / *AXIM_AWUSER* , which support transaction initiator indication, are connected to the AXIM module level signals as follows:
AXIM module signal *ARUSER* is connected to top level port *AXIM_ARUSER*
AXIM module signal *AWUSER* is connected to top level port *AXIM_AWUSER*

4.2.3.5 Data Alignment

AXI Transaction Generator shifts the data from the slot **AXIM[i]_SLOT[s]_DATA_LOW** register to the correct AXI data bus lines, according to **AXIM[i]_SLOT[s]_ADDR_LOW** . Software must expect the data inside the data register always aligned to bit 0. For AXI operations with a size of less than 32-bits, the Transaction Generator shifts the data bytes to the respective positions on the AXI data bus. For AXI read operations, the Transaction Generator masks unused bytes. E.g.: If software wants to initiate a one byte AXI write transaction to address 3, it programs the data byte to bit **AXIM[i]_SLOT[s]_DATA_LOW** [7:0]. AXI Transaction Generator shifts this byte to bits [31:24] of the AXI data bus.
For AXI write operations, the Transaction Generator calculates the correct AXI write strobe signals (*WSTRB*) depending on **AXIM[i]_SLOT[s]_CFG1.AXI_SIZE** and **AXIM[i]_SLOT[s]_ADDR_LOW** .

4.2.3.5.1 64bit AXI data width support

The GTM supports 32-bit or 64-bit AXI data bus width configurations (selected via device configuration variable AXIM_DATA_WIDTH). Still, even with a 64bit AXI data bus width only 32bit accesses are initiated. If the AXI data bus width is 64-bit the gtm_axim "only" reads the 32-bit data word from the corresponding data lines and provides in case of a write access the write data duplicated on the upper and the lower 32-bit data lines and selects the AXI write strobe signals (*WSTRB*) accordingly.

4.3 AXIM Configuration Registers description

4.3.1 AXIM[i]_FREE

Description	AXIM[i] slot allocation status.
Loop	$i = \{n : 0 <= n <= \mathrm{NAXIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{AXIM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	AXIM[i]_FREE
Address	$0x20000 * i + 0x5000$
C-Name	GTM.CLS[i].AXIM.FREE

Interface: MCS[i]

Name	AXIM[i]_FREE
Address	$0x5000$
C-Name	

FREE[t]	
Description	This bit represents the allocation status of the slot [t].
Loop	$t = \{n : 0 <= n <= \text{AXIM} - 1\}$
Bit Range	$[t : t]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Slot is allocated, or not available 1 : Slot is free

4.3.2 AXIM[i]_REQUEST

Description	AXIM[i] slot request (allocation).
Loop	$i = \{n : 0 <= n <= \text{NAXIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{AXIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	AXIM[i]_REQUEST
Address	$0x20000 * i + 0x5004$
C-Name	GTM.CLS[i].AXIM.REQUEST

Interface: MCS[i]

Name	AXIM[i]_REQUEST
Address	$0x5004$
C-Name	

REQ1HOT[t]	
Description	A read to the AXIM[i]_REQUEST register will allocate a new slot if any slot is available. This bit field shows the new allocated slot as 1-hot encoded vector. If no slot could be allocated, this bit field is read as all 0.
Loop	$t = \{n : 0 <= n <= \text{AXIM} - 1\}$
Bit Range	$[t : t]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Slot t not allocated 1 : Slot t was allocated by current request

REQID	
Description	This bit field shows the new allocated slot as binary encoded index. If no slot could be allocated, this bit field is read as all 1 (0xff).
Loop	-
Bit Range	$[31 : 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

4.3.3 AXIM[i]_RELEASE

Description	AXIM[i] slot release (de-allocation).
Loop	$i = \{n : 0 <= n <= \text{NAXIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{AXIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	AXIM[i]_RELEASE
Address	$0x20000 * i + 0x5008$
C-Name	GTM.CLS[i].AXIM.RELEASE

Interface: MCS[i]

Name	AXIM[i]_RELEASE
Address	$0x5008$
C-Name	

RELREQ[t]	
Description	Slot [t] release request: A write to AXIM[i]_RELEASE.RELREQ[t] de-allocates one or more slots. Each bit in AXIM[i]_RELEASE.RELREQ[t] represents one slot.
Loop	$t = \{n : 0 <= n <= \text{AXIM} - 1\}$
Bit Range	$[t : t]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0 : Don't change state of Slot(t) 1 : De-allocate Slot(t)

Note:
This bit field is self-clearing and always read as zero.

Note:
Software checks, if a release request really results in slot de-allocation. Slot only changes the state at release request if the slot has no pending request to the AXI Transaction Generator (Slot states **AXIM[i]_SLOT[s]_STATUS.ALLOC** or **AXIM[i]_SLOT[s]_STATUS.READY**).

4.3.4 AXIM[i]_SLOT[s]_ADDR_LOW

Description	AXIM[i] slot[s] address bits 31:0 of AXI transaction.
Loop	$i = \{n : 0 <= n <= \text{NAXIM} - 1\}; s = \{n : 0 <= n <= \text{AXIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{AXIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	AXIM[i]_SLOT[s]_ADDR_LOW
Address	$0x20000 * i + 0x20 * s + 0x5020$
C-Name	GTM.CLS[i].AXIM.SLOT[s].ADDR_LOW

Interface: MCS[i]

Name	AXIM[i]_SLOT[s]_ADDR_LOW
Address	$0x20 * s + 0x5020$
C-Name	

AXI_ADDR	
Description	Address for the AXI transaction.
Loop	–
Bit Range	$[31:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: If the auto-increment mode is enabled (**AXIM[i]_SLOT[s]_CFG1.AUTO_INCR** =1), this register is updated after each AXI transaction: **AXIM[i]_SLOT[s]_ADDR_LOW.AXI_ADDR** <= **AXIM[i]_SLOT[s]_ADDR_LOW.AXI_ADDR** + **AXIM[i]_SLOT[s]_CFG1.INCR** If the slot is in **AXIM[i]_SLOT[s]_STATUS.ALLOC** or **AXIM[i]_SLOT[s]_STATUS.READY** state. A write to this register starts a new AXI request (Slot state changes to **AXIM[i]_SLOT[s]_STATUS.QUEUED**).

4.3.5 AXIM[i]_SLOT[s]_DATA_LOW

Description	AXIM[i] slot[s] data bits 31:0 of AXI transaction.
Loop	$i = \{n : 0 <= n <= \text{NAXIM} - 1\}; s = \{n : 0 <= n <= \text{AXIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{AXIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	AXIM[i]_SLOT[s]_DATA_LOW
Address	$0x20000 * i + 0x20 * s + 0x5028$
C-Name	GTM.CLS[i].AXIM.SLOT[s].DATA_LOW

Interface: MCS[i]

Name	AXIM[i]_SLOT[s]_DATA_LOW
Address	$0x20 * s + 0x5028$
C-Name	

AXI_DATA_LOW	
Description	Read or write data.
Loop	–
Bit Range	$[31:0]$
Access Type	RW
Volatile	False

AXI_DATA_LOW	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: AXI write operation (**AXIM[i]_SLOT[s]_CFG1.AXI_RW** ==0): Data that is sent out via AXI transaction. Software must configure this register before the AXI write operation is started. Data in **AXIM[i]_SLOT[s]_DATA_LOW.AXI_DATA_LOW** is aligned to bit 0. AXI Transaction Generator applies the data to the correct AXI data lines depending on **AXIM[i]_SLOT[s]_ADDR_LOW.AXI_ADDR** . **Note:** If auto-increment is enabled (**AXIM[i]_SLOT[s]_CFG1.AUTO_INCR** ==1) and the slot is in **AXIM[i]_SLOT[s]_STATUS.READY** state. A write to this register starts a new AXI transaction. (Slot state changes to **AXIM[i]_SLOT[s]_STATUS.QUEUED**). **Note:** AXI read operation (**AXIM[i]_SLOT[s]_CFG1.AXI_RW** ==1): Data read by last AXI transaction. Software reads the received data after the AXI transaction has finished (Slot state == **AXIM[i]_SLOT[s]_STATUS.READY**). Data in **AXIM[i]_SLOT[s]_DATA_LOW.AXI_DATA_LOW** is always aligned to bit 0. AXI Transaction Generator shifts the AXI data bus lines according to **AXIM[i]_SLOT[s]_ADDR_LOW.AXI_ADDR** . Unused bytes (identified by **AXIM[i]_SLOT[s]_CFG1.AXI_SIZE**) are masked and are read as all 0. **Note:** If auto-increment is enabled (**AXIM[i]_SLOT[s]_CFG1.AUTO_INCR** ==1) and the slot is in **AXIM[i]_SLOT[s]_STATUS.READY** state. A read to this register starts a new AXI transaction. (Slot state changes to **AXIM[i]_SLOT[s]_STATUS.QUEUED**).

4.3.6 AXIM[i]_SLOT[s]_CFG1

Description	AXIM[i] slot [s] configuration 1
Loop	$i = \{n : 0 <= n <= \text{NAXIM} - 1\}; s = \{n : 0 <= n <= \text{AXIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{AXIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	AXIM[i]_SLOT[s]_CFG1
Address	$0x20000 * i + 0x20 * s + 0x5030$
C-Name	GTM.CLS[i].AXIM.SLOT[s].CFG1

Interface: MCS[i]

Name	AXIM[i]_SLOT[s]_CFG1
Address	$0x20 * s + 0x5030$
C-Name	

INCR	
Description	Address increment for auto-increment mode

INCR	
Loop	–
Bit Range	[3 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

AUTO_INCR	
Description	Enable or disable auto-increment mode
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : disable auto-increment 1 : enable auto-increment

PRIO	
Description	Slot priority. Priority used for slot arbitration. Lowest priority is 0, highest priority is 3.
Loop	–
Bit Range	[6 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

AXI_PROT	
Description	AXI priority mode
Loop	–

AXI_PROT	
Bit Range	[13 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	AXIM[i]_SLOT[s]_CFG1.AXI_PROT [1:1]: 0 = Secure mode access; 1 = none-secure mode access **AXIM[i]_SLOT[s]_CFG1.AXI_PROT** [0:0]: 1 = privileged mode access; 0 = user mode access **AXIM[i]_SLOT[s]_CFG1.AXI_PROT** [2:2]: 0 = Data mode access; 1 = instruction mode access

AXI_CACHE	
Description	AxCACHE bit field (see ARM AXI-3 spec)
Loop	–
Bit Range	[17 : 14]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

AXI_LOCK	
Description	AXI Lock indication
Loop	–
Bit Range	[19 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

AXI_LOCK	
RW-Coding	0b00 : no lock 0b01 : exclusive access ("soft lock") 0b10 : access with bus lock (not allowed for GTM) 0b11 : reserved (not allowed for GTM)

AXI_SIZE	
Description	AXI data size
Loop	-
Bit Range	[24 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : transmit 1 data byte 0b001 : transmit 2 data bytes 0b010 : transmit 4 data bytes 0b011 : not allowed for GTM 0b100 : not allowed for GTM 0b101 : not allowed for GTM 0b110 : not allowed for GTM 0b111 : not allowed for GTM

AXI_RW	
Description	AXI Read Write indication
Loop	-
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : AXI write transaction 1 : AXI read transaction

4.3.7 AXIM[i]_SLOT[s]_CFG2

Description	AXIM[i] slot[s] configuration 2

Loop	$i = \{n : 0 <= n <= \text{NAXIM} - 1\}; s = \{n : 0 <= n <= \text{AXIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{AXIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	AXIM[i]_SLOT[s]_CFG2
Address	$0x20000 * i + 0x20 * s + 0x5034$
C-Name	GTM.CLS[i].AXIM.SLOT[s].CFG2

Interface: MCS[i]

Name	AXIM[i]_SLOT[s]_CFG2
Address	$0x20 * s + 0x5034$
C-Name	

AXI_ID	
Description	AXI ID for transaction. If posted writes are enabled (device configuration variable), the lower bit of the AXIM[i]_SLOT[s]_CFG2.AXI_ID selects between posted and none-posted writes.
Loop	-
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	1 : on **AXIM[i]_SLOT[s]_CFG2.AXI_ID** [0:0] = 1 use posted write (if enabled by device configuration variable AXIM_POSTED_WRITE=1; value can be read via AXIM_POSTED_WRITE) 0 : on **AXIM[i]_SLOT[s]_CFG2.AXI_ID** [0:0] = 0 always use none-posted write
	Note: Only lower bits of **AXIM[i]_SLOT[s]_CFG2.AXI_ID** are send to the bus, if device configuration variable AXIM_ID_WIDTH is less than 16.

4.3.8 AXIM[i]_SLOT[s]_STATUS

Description	AXIM[i] slot[s] status
Loop	$i = \{n : 0 <= n <= \text{NAXIM} - 1\}; s = \{n : 0 <= n <= \text{AXIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{AXIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	AXIM[i]_SLOT[s]_STATUS
Address	$0x20000 * i + 0x20 * s + 0x5038$
C-Name	GTM.CLS[i].AXIM.SLOT[s].STATUS

Interface: MCS[i]

Name	AXIM[i]_SLOT[s]_STATUS
Address	$0x20 * s + 0x5038$
C-Name	

ALLOC	
Description	AXI Slot occupation indication
Loop	-
Bit Range	$[0:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Slot is free 1 : Slot is allocated
	Note: This bit is asserted as long as the slot is allocated, also during states **AXIM[i]_SLOT[s]_STATUS.QUEUED** , **AXIM[i]_SLOT[s]_STATUS.STARTED** and **AXIM[i]_SLOT[s]_STATUS.READY** . The slot state Allocated is represented by **AXIM[i]_SLOT[s]_STATUS.ALLOC** =1 and **AXIM[i]_SLOT[s]_STATUS.QUEUED** = **AXIM[i]_SLOT[s]_STATUS.STARTED** = **AXIM[i]_SLOT[s]_STATUS.READY** =0

QUEUED	
Description	This bit represents the slot AXIM[i]_SLOT[s]_STATUS.QUEUED state.
Loop	-
Bit Range	$[1:1]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : AXI request is not queued for arbitration 1 : AXI request is queued for arbitration

STARTED	
Description	This bit represents the slot AXIM[i]_SLOT[s]_STATUS.STARTED state.
Loop	–
Bit Range	$[2:2]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : AXI request is not accepted by arbiter 1 : AXI request is accepted by arbiter

READY	
Description	This bit represents the slot AXIM[i]_SLOT[s]_STATUS.READY state.
Loop	–
Bit Range	$[3:3]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : AXI Transaction Generator has not finished request 1 : AXI Transaction Generator has finished request

RESP	
Description	AXI response from last AXI transaction
Loop	–
Bit Range	$[5:4]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

RESP	
R-Coding	0b00 : OK 0b01 : EXOKAY 0b10 : SLVERR 0b11 : DECERR; If AXI Transaction Generator detects any invalid requests or responses (unexpected response ID), it forces response to this value.

4.4 AXIM Port Description

4.4.1 AXIM General Ports

Loop	–
Condition	–
Format	–

ACLK	
Description	AXI port clock
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARESETN	
Description	AXI port reset (always low active). Release of ARESETN must be synchronized to ACLK.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AXIM_CLK	
Description	AXIM port clocks. One clock per AEIM interface. ACLK /AXIM_CLK must all fulfill synchronous timing relationships, but AXIM_CLK may be gated.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–

AXIM_CLK	
Special Function	–
Operational Reset	–
Reset Value	–

ENCLK	
Description	Gate enable for AXIM clocks. Used to synchronize data between AXIM_CLK /ACLK domains.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

CLKOFF	
Description	AXIM interface [i] of cluster [i] is in clock off state. This signal works as a synchronous reset for AEIM interface and overrules ENCLK to avoid hang-ups in ACLK /AXIM_CLK synchronization.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AXIM_GTM_HALT	
Description	Debug signal. While AXIM_GTM_HALT is asserted, no new requests are accepted by the AXI port. Pending AXI cycles are still completed.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

4.4.2 AXIM AEI Ports

Loop	–
Condition	–

Format	–

AEIM_AXIM_IRQ	
Description	AEI interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_STATUS	
Description	AEI transaction status
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_RDATA	
Description	AEI read data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	31 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_READY	
Description	AEI ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	–

AEIM_AXIM_READY	
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_WDATA	
Description	AEI write data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	31 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_ADDR_LSB	
Description	AEI address LSBs
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_ADDR	
Description	AEI address MCBs
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_W1R0	
Description	AEI write/read selection
Loop	–
Condition	–

AEIM_AXIM_W1R0	
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_SEL	
Description	AEI select
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_DEBUG_ACCESS	
Description	AEI Debug Access
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AEIM_AXIM_IRQ_CLEAR	
Description	AEI interrupt clear
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

4.4.3 AXIM Read Address Ports

Loop	–
Condition	–
Format	–

ARID	
Description	AXI read address ID, ID width can be selected via device configuration variable.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	AXIM_ID_WIDTH-1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARADDR	
Description	AXI read address
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	31 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARLEN	
Description	AXI read burst length (fixed to 0x0, as we only support single transactions).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARBURST	
Description	AXI read burst type (fixed to 0x1 (AXIM[i]_SLOT[s]_CFG1.INCR))
Loop	–
Condition	–
Logical Name	–

ARBURST	
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARSIZE	
Description	AXI read burst size
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARLOCK	
Description	AXI lock type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARCACHE	
Description	AXI cache type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARPROT	
Description	AXI protection type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARVALID	
Description	AXI read address valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARREADY	
Description	AXI read address ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

ARUSER	
Description	AXI read user data; one hot coded signal, the value ARUSER[i:i]=1 signals that the corresponding cluster i has initiated this read transaction.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	NAXIM-1 DOWNTO 0
Operational Clock	–
Special Function	–

ARUSER	
Operational Reset	–
Reset Value	–

4.4.4 AXIM Write Address Ports

Loop	–
Condition	–
Format	–

AWID	
Description	AXI write address ID, ID width is selected via device configuration variable.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	AXIM_ID_WIDTH-1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWADDR	
Description	AXI write address
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	31 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWLEN	
Description	AXI write burst length (fixed to 0x0, as we only support single transactions).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWBURST	
Description	AXI write burst type (fixed to 0x1 (AXIM[i]_SLOT[s]_CFG1.INCR)).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWSIZE	
Description	AXI write burst size
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWLOCK	
Description	AXI lock type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWCACHE	
Description	AXI cache type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	–
Special Function	–

AWCACHE	
Operational Reset	–
Reset Value	–

AWPROT	
Description	AXI protection type
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWVALID	
Description	AXI write address valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWREADY	
Description	AXI write address ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

AWUSER	
Description	AXI write user data; one hot coded signal, the value AWUSER[i:i]=1 signals that the corresponding cluster i has initiated this write transaction.
Loop	–
Condition	–
Logical Name	–

AWUSER	
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	NAXIM-1 DOWNTO 0
Operational Clock	-
Special Function	-
Operational Reset	-
Reset Value	-

4.4.5 AXIM Read Data Ports

Loop	-
Condition	-
Format	-

RID	
Description	AXI read data ID, ID width can be selected via device configuration variable.
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	AXIM_ID_WIDTH-1 DOWNTO 0
Operational Clock	-
Special Function	-
Operational Reset	-
Reset Value	-

RDATA	
Description	AXI read data, width can be selected via device configuration variable AXIM_DATA_WIDTH.
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	31 DOWNTO 0
Operational Clock	-
Special Function	-
Operational Reset	-
Reset Value	-

RLAST	
Description	AXI read last in a burst signal
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC
Port Direction	IN
Port Range	-

RLAST	
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

RRESP	
Description	AXI read response
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

RVALID	
Description	AXI read data valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

RREADY	
Description	AXI read data ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

4.4.6 AXIM Write Data Ports

Loop	–
Condition	–

Format	–

WID	
Description	AXI write data ID, ID width can be selected via device configuration variable.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	AXIM_ID_WIDTH-1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

WDATA	
Description	AXI write data, width can be selected via device configuration variable AXIM_DATA_WIDTH.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	31 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

WLAST	
Description	AXI write last in a burst signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

WSTRB	
Description	AXI byte strobes, width can be selected via device configuration variable AXIM_DATA_WIDTH.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	AXIM_DATA_WIDTH/8-1 DOWNTO 0

WSTRB	
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

WVALID	
Description	AXI write data valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

WREADY	
Description	AXI write data ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

4.4.7 AXIM Response Ports

Loop	–
Condition	–
Format	–

BID	
Description	AXI response ID, ID width can be selected via device configuration variable.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	AXIM_ID_WIDTH-1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–

BID	
Reset Value	–

BRESP	
Description	AXI write response
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

BVALID	
Description	AXI write response valid
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

BREADY	
Description	AXI write response ready
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	–
Special Function	–
Operational Reset	–
Reset Value	–

5 Advanced Routing Unit (ARU)

5.1 Overview

The Advanced Routing Unit (ARU) is a flexible infrastructure component for transferring 53-bit wide data (five control bits and two 24-bit values) between several submodules of the GTM core in a configurable manner.

Indices and their range as used inside this chapter are:

- g:={0, 1} index of ARU port
- k:={0, 1} index of ARU sub-channels

Since the concept of the ARU has already been described in section *3.9 "ARU Routing Concept"* , this section only describes additional ARU features that are used by the software for configuring and debugging ARU related data streams.
Also, the definitions of 'streams' and 'channels' in the context of ARU are specified in section 3.4 "ARU Routing Concept" *3.9 "ARU Routing Concept"* .

The principle of ARU data routing is described in chapter *3.9 "ARU Routing Concept"* .
In the real GTM implementation, the ARU serves per clock period two individual data destinations in parallel, one at interface ARU_0 and another one at interface ARU_1. Both ARU interfaces ARU_0 and ARU_1 run in parallel by default but can be configured in dynamic routing mode (see below) to run in an individual mode.

Inside the ARU a 2-cycle pipeline stage is implemented. Therefore, one read access by the ARU takes two clock cycles or two ARU slots respectively.
Due to this implemented pipeline stage, the fastest data rate that can be achieved in the dynamic routing mode with just one dedicated data sink (ARU master ID) is one data transfer every third ARU clock cycles.
This means that if one ARU master ID is addressed back-to-back by holding the same ARU master ID constant for one or several ARU clock cycles, or addressed alternatively with another ARU master ID, each second ARU master ID will not be served (on ARU slave interface) by connected data source.
Consecutive different ARU master IDs round trip is not affected and the transfer is executed one after the other because the ARU data sources connected to ARU data destinations with different ARU master IDs are different.

Each data source has its specific write address where the data destination can read via the ARU interface ARU_0 or ARU_1. Each data destination module should be configured with the specific write address of the required data source.
When the counter **ARU_CADDR** of ARU interfaces ARU_0 or ARU_1 is equal to the ARU master ID of a GTM submodule, this module will be served as the data destination and the specific write address of its configured data source will be selected to provide data to serve the data destination.
Via the ARU interfaces ARU_0 and ARU_1 with two different write addresses, two independent GTM submodules (data destinations) can be served.
The combination of ARU interfaces (ARU_0 or ARU_1) and each unique ARU write address only points to one ARU data source (i.e. the ARU write port of a GTM submodule).

The assignment of GTM submodules (possible data destinations) to the ARU interfaces ARU_0 and ARU_1 and their ARU master IDs are device-dependent and can be found in *94 "ARU Master ID"* . The write addresses of GTM submodules (possible data sources) can be found in *93 "ARU Write Address Overview"* .

If cluster 0 clock signal is disabled, i.e. *CLS[0]_CLK_ENABLE* = 0 and *ARU_CLK_ENABLE* = 0, the ARU functionalities will be still running, but the ARU registers access via AEI configuration interface will be not possible.

5.2 Special Data Sources

Besides the addresses of the submodule related data sources as described in table *93 "ARU Write Address Overview"* , the ARU provides two special data sources that can be used for the configuration of data streams. These data sources are defined as follows:

Address 0x1FF: Data source that always provides a 53-bit data word with zeros. A read access to this memory location never blocks a requesting data destination.

Address 0x1FE: Data source that never provides a data word. A read access to this memory location always blocks a requesting data destination. This is the reset value of the read address registers inside the data destinations.

Address 0x000: This address is reserved and is used to bring data through the ARU registers **ARU_DATA_H** and **ARU_DATA_L** into the system by writing the write address 0x000 into the **ARU_ACCESS** register. This means that software test data can be brought into the GTM-IP by the CPU.

5.3 ARU Access via AEI

Besides the data transfer between the connected submodules, there are two possibilities to access ARU data via the AEI.

5.3.1 Default ARU Access

The default ARU access incorporates the register **ARU_ACCESS** , which is used for initiation of a read or write request and the registers **ARU_DATA_H** and **ARU_DATA_L** that provide the ARU data word to be transferred.

The status of a read or write transfer can be determined by polling specific bits in register **ARU_ACCESS** . Furthermore, the **ARU_IRQ_NOTIFY.ACC_ACK** bit in the interrupt notify register is set after the read or write access is performed to avoid data loss e.g. on access cancelation.

A pending read or write request is also canceled by clearing the associated bit.

In the case of a read request, the AEI access behaves as a read request initiated by a data destination of a module. The read request is served by the ARU immediately, when no other destination has a pending read request. This means, that an AEI read access does not take part in the scheduling of the destination channels and that the time between two consecutive read accesses is not limited by the round trip time.

On the other hand, the AEI access has the lowest priority behind the ARU scheduler that serves the destination channels. Thus, in worst case, the read request is served after one round trip of the ARU, when all destination channels request data at the same point in time.

In the case of the write request, the ARU provides the write data at the address defined by the **ARU_ACCESS.ADDR** .

To avoid data loss, the reserved ARU address 0x0 has to be used to bring data into the system. Otherwise, in case the address specified inside the **ARU_ACCESS.ADDR** bit field is defined for another submodule that acts as a source at the ARU, data loss may occur and no deterministic behavior is guaranteed.

This is because the regular source submodule is not aware that its address is used by the ARU itself to provide data to a destination.

It is guaranteed that the ARU write data is sent to the destination in case both modules want to provide data at the same time.

Configuring both read and write request bits results in a read request, if the write request bit inside the register isn't already set. The read request bit is set but not the write request bit. The following table describes the important cases of the bit **ARU_ACCESS.RREQ** and bit **ARU_ACCESS.WREQ** :

Table 11 ARU_ACCESS.WREQ and ARU_ACCESS.RREQ

AEI write access : *AEI_WDATA* [13:12]	current value of ARU_ACCESS [13:12]	next value of ARU_ACCESS [13:12]	comment
0 0	0 1	0 0	cancel read request
0 0	1 0	0 0	cancel write request
0 1	1 0	1 0	unchanged register
1 0	0 1	0 1	unchanged register
1 1	0 0	0 1	both read and write request results in a read request
1 1	1 0	1 0	as before but **ARU_ACCESS.WREQ** bit is already set –> unchanged register

5.3.2 Debug Access

The debug access mode enables to inspect routed data of configured data streams during runtime.

The ARU provides two independent debug channels, whereas each is configured by a dedicated ARU read address in register **ARU_DBG_ACCESS0** and **ARU_DBG_ACCESS1** respectively.

The registers **ARU_DBG_DATA0_H** and **ARU_DBG_DATA0_L** (**ARU_DBG_DATA1_H** and **ARU_DBG_DATA1_L**) provide read access to the latest data word that the corresponding data source sent through the ARU.

Any time when data is transferred through the ARU from a data source to the destination requesting the data, the interrupt signal *ARU_NEW_DATA0_IRQ* (*ARU_NEW_DATA1_IRQ*) is raised.

For advanced debugging purposes, the interrupt signal is also triggered by software using the register **ARU_IRQ_FORCINT** .

The debug mechanism must not be used by the application, when a HW–Debugger is used to trace the ARU communication. In that case, the debug registers are used by the HW–Debugger to specify the ARU streams that must be traced.

5.4 ARU dynamic routing

A dynamic routing feature of the ARU is implemented and configured using the additional AEI registers:

- **ARU_CTRL**
- **ARU_[g]_DYN_CTRL**
- **ARU_[g]_DYN_RDADDR**
- **ARU_[g]_DYN_ROUTE_LOW**

- **ARU_[g]_DYN_ROUTE_HIGH**
- **ARU_[g]_DYN_ROUTE_SR_LOW**
- **ARU_[g]_DYN_ROUTE_SR_HIGH**

For further information see the register part of this chapter.

5.4.1 Dynamic routing – CPU controlled

The dynamic routing feature is enabled separately for ARU_0 and ARU_1 by setting the corresponding bit fields of the register **ARU_CTRL** .

After **ARU_[g]_DYN_ROUTE_LOW** is configured and the dynamic routing featured is enabled by configuring **ARU_CTRL** , the starting of the configured dynamic routing will be synchronous to the next earliest time point when the first ARU master ID is addressed in the running normal routing scheme, i.e. ARU master ID of 0 is addressed. The dynamic route will be started with additional ARU master ID specified in **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID0** .

With the dynamic routing feature, it is possible to insert additional ARU master IDs, **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID0** , **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID1** , **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID2** , **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID3** , **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID4** , and **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID5** , in a defined manner into the normal ARU routing scheme.

While inserting additional ARU master IDs and enabling the dynamic routing, the normal ARU routing scheme is paused. Therefore, please consider that the dynamic routing can prolong the normal routing scheme.

It is possible to configure 6 additional ARU master ID's in the **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** registers for both ARU_0 and ARU_1.

In the bit field **ARU_[g]_DYN_ROUTE_HIGH.DYN_CLK_WAIT** the number of clock cycles between 2 dynamic master IDs has to be configured.

After each configured number of clock cycles, the defined ARU master ID's will be inserted cyclically one after each other in the following manner:
... -> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID0** -> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID1** -> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID2** -> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID3** -> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID4** -> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID5** -> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID0** -> ...

In the shadow registers **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** , 6 ARU master IDs from **ARU_[g]_DYN_ROUTE_SR_LOW.DYN_READ_ID6** to **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_READ_ID11** can be further defined.

If **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_UPDATE_EN** is set to 1, the **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** registers can be updated from its shadow registers except **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_UPDATE_EN** . The update is executed and the updated master IDs are inserted one by one after the original 6 master IDs in **ARU_[g]_DYN_ROUTE_LOW** have been inserted. When update is performed, **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_UPDATE_EN** is reset. That means setting **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_UPDATE_EN** can trigger the update only once.

With the **ARU_[g]_DYN_CTRL.DYN_ROUTE_SWAP** option it is possible to swap the registers **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** with its shadow registers **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** . The swapping is executed always after the 6 ARU master IDs are inserted. So it is possible to insert a maximum of 12 ARU master IDs cyclically after every configured number of clock cycles. If swap started, **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_UPDATE_EN** is reset.

ARU_[g]_DYN_CTRL.DYN_ROUTE_SWAP option is not supported, if ARU update for dynamic routing mode is enabled. Therefore, the configuration bit **ARU_[g]_DYN_CTRL.DYN_ROUTE_SWAP** must be reset, if configuration bit **ARU_[g]_DYN_CTRL.DYN_ARU_UPDATE_EN** is enabled.

By setting the bit field **ARU_[g]_DYN_ROUTE_HIGH.DYN_CLK_WAIT** to zero, only the defined ARU master read IDs are executed. The normal ARU routing scheme is stopped.

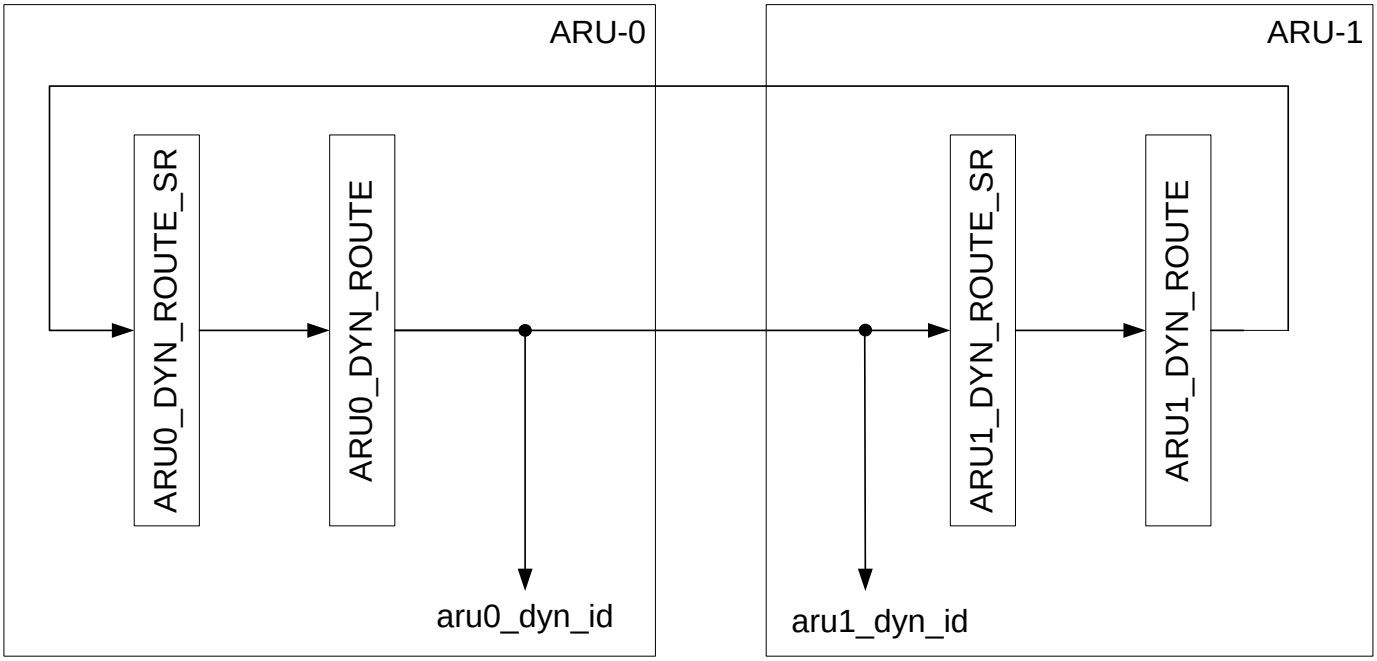
By setting the bit field **ARU_[g]_DYN_ROUTE_HIGH.DYN_CLK_WAIT** to 15, only the normal ARU routing scheme is executed. Inserting of additional IDs is stopped.

To reset the ARU caddr counter and ARU dynamic route counter, set the bit **CMU_GLB_CTRL.ARU_ADDR_RSTGLB** followed by a write access to register **CMU_CLK_EN** .

5.4.1.1 Dynamic routing ring mode

In dynamic routing ring mode, it is possible to use all 24 read ID's from both ARU_0 and ARU_1 by setting bit field **ARU_CTRL.ARU_DYN_RING_MODE** to 1. In this mode all 4 registers **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** and **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** are connected as a ring, so all 24 read ID's can be used from both ARU's. The ring structure is shown in figure *33 "ARU dynamic routing – ring mode"* . The data register shift direction is shown by the arrows in the ring.

Figure 33 ARU dynamic routing – ring mode



Enabling the dynamic routing ring mode automatically resets the caddr counter of both ARU_0 and ARU_1. This is necessary to synchronize both ARU's in this mode.

Enabling the dynamic routing ring mode will ignore **ARU_[g]_DYN_CTRL.DYN_ROUTE_SWAP** and **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_UPDATE_EN** .

ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_UPDATE_EN must be disabled in dynamic routing ring mode.

It is possible to enable the dynamic routing ring mode for both ARU_0 and ARU_1 or only for one of the ARU's by setting the corresponding bit field **ARU_CTRL.ARU_[k]_DYN_EN** .
Because of the fact that two different GTM submodules are served to each ARU interface ARU_0 and ARU_1 with the same ARU master ID, it also makes sense to enable ARU dynamic routing only for one interface, either ARU_0 or ARU_1 if configured to ring mode. The other port is then served in the default Round Robin manner.

In dynamic routing ring mode **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** and **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** are not write-protected.

It is recommended to avoid modification of **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** and **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** in active dynamic routing ring mode.

5.4.2 Dynamic routing – ARU controlled

Furthermore, it is possible to reload the **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** registers by ARU itself.

Therefore, the ARU has its own master port which is served in the normal ARU routing scheme. The ARU read address for this master port has to be configured in the register **ARU_[g]_DYN_RDADDR** .

This feature is enabled by setting **ARU_[g]_DYN_CTRL.DYN_ARU_UPDATE_EN** register.

The following mapping of the ARU word to the **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** registers is implemented:
ARU_[g]_DYN_ROUTE_SR_LOW = *ARU_DATA* [23:0]
ARU_[g]_DYN_ROUTE_SR_HIGH = *ARU_DATA* [52:24]

The bit field **ARU_DATA** [51:48] controls the configuration bits **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_CLK_WAIT** and the bit **ARU_DATA** [52:52] controls the configuration bit **ARU_[g]_DYN_ROUTE_SR_HIGH.DYN_UPDATE_EN** . Both functions are described in the chapter above *5.4.1 "Dynamic routing – CPU controlled"* .

Opposite to the dynamic routing scheme controlled from CPU/AEI (only the 6 additional ARU master IDs are inserted) two additional IDs are served. One is the ARU master ID itself for reloading and the other is the default ID_0. The ID_0 is only added to the inserted routing scheme if bit field **ARU_[g]_DYN_ROUTE_HIGH.DYN_CLK_WAIT** is set to zero (only the inserted routing scheme is executed). This ensures that a debug access takes place even if only the inserted routing scheme is executed.

The following dynamic routing scheme is executed for 15 > **ARU_[g]_DYN_ROUTE_HIGH.DYN_CLK_WAIT** > 0 :
... –> ARU_master_ID –> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID0** –> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID1** –> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID2** –> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID3** –> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID4** –> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID5** –> ARU_master_ID –> ...

The following dynamic routing scheme is executed for **ARU_[g]_DYN_ROUTE_HIGH.DYN_CLK_WAIT** == 0 :
... –> ARU_master_ID –> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID0** –> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID1** –> **ARU_[g]_DYN_ROUTE_LOW.DYN_READ_ID2** –> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID3** –> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID4** –> **ARU_[g]_DYN_ROUTE_HIGH.DYN_READ_ID5** –> default_ID0 –> ARU_master_ID –> ...

With the possibility of reloading the dynamic routing scheme over ARU, a FIFO or MCS is able to deliver the dynamic routing scheme data.

As long as this feature is enabled by setting of bit **ARU_[g]_DYN_CTRL.DYN_ARU_UPDATE_EN** , the dynamic routing registers **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** and **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** must not be written over configuration interface to avoid unexpected behavior.

5.5 ARU Configuration Registers Description

5.5.1 ARU_ACCESS

Description	ARU access register
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_ACCESS
Address	$0x180$
C–Name	GTM.CLS[0].ARU.ACCESS

Interface: MCS[i]

Name	ARU_ACCESS
Address	$0x180$
C–Name	

ADDR	
Description	ARU address
Loop	–
Bit Range	$[8:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0x1FE$
Protect Enable Cond	(ARU_ACCESS.RREQ == 1) || (ARU_ACCESS.WREQ == 1)
Reset group	HW_RESET: async GTM_RESET: async

ADDR	
	Define the ARU address used for transferring data. **Note:** For an ARU write request, the preferred address 0x0 have to be used. **Note:** A write request to the address 0x1FF (always full address) or 0x1FE (always empty address) are ignored and doesn't have any effect. **Note:** ARU address bits **ARU_ACCESS.ADDR** are only writable, if **ARU_ACCESS.RREQ** and **ARU_ACCESS.WREQ** bits are zero.

RREQ	
Description	Initiate read request
Loop	–
Bit Range	[12 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	ARU_ACCESS.WREQ == 1
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : No read request is pending 1 : Set read request to source channel addressed by **ARU_ACCESS.ADDR**
	Note: This bit is cleared automatically after transaction. Moreover, it is cleared by the software to cancel a read request. **Note:** **ARU_ACCESS.RREQ** bit is only writable if **ARU_ACCESS.WREQ** bit is zero, so to switch from **ARU_ACCESS.RREQ** to **ARU_ACCESS.WREQ** , a cancel request has to be performed in advance. **Note:** Configuring both **ARU_ACCESS.RREQ** and **ARU_ACCESS.WREQ** bits results in a read request, so **ARU_ACCESS.RREQ** bit is set if the **ARU_ACCESS.WREQ** bit of the register isn't already set. **Note:** The ARU read request on address **ARU_ACCESS.ADDR** is served immediately when no other destination has actually a read request when the **ARU_ACCESS.RREQ** bit is set by CPU. In a worst case scenario, the read request is served after one round trip of the ARU, but this is only the case when every destination channel issues a read request at consecutive points in time.

WREQ	
Description	Initiate write request
Loop	–
Bit Range	[13 : 13]
Access Type	RW
Volatile	True
Multithread	False

WREQ	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(ARU_ACCESS.RREQ == 1) || (bitrange(CLS[0]_AEI_ARB_WDATA, 12, 12) == 1)
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : No write request is pending 1 : Mark data in registers **ARU_DATA_H** and **ARU_DATA_L** as valid
	Note: This bit is cleared automatically after transaction. Moreover, it is cleared by the software to cancel a write request. **Note:** **ARU_ACCESS.WREQ** bit is only writable if **ARU_ACCESS.RREQ** bit is zero, so to switch from **ARU_ACCESS.WREQ** to **ARU_ACCESS.RREQ** a cancel request has to be performed in advance. **Note:** Configuring both **ARU_ACCESS.RREQ** and **ARU_ACCESS.WREQ** bits results in a read request, so **ARU_ACCESS.WREQ** bit is not set. **Note:** The data is provided at address **ARU_ACCESS.ADDR** . This address has to be programmed as the source address in the destination submodule channel. In worst case, the data is provided after one full ARU round trip.

Note:
The register **ARU_ACCESS** is used either for reading or for writing, but not both at the same point in time.

5.5.2 ARU_DATA_H

Description	ARU access register upper data word
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_DATA_H
Address	$0x184$
C–Name	GTM.CLS[0].ARU.DATA_H

Interface: MCS[i]

Name	ARU_DATA_H
Address	$0x184$
C–Name	

DATA	
Description	Upper ARU data word
Loop	–
Bit Range	$[28 : 0]$

DATA	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: Transfer upper ARU data word addressed by **ARU_ACCESS.ADDR** . The data bits [52:24] of an ARU word are mapped to the data bits [28:0] of this register.

5.5.3 ARU_DATA_L

Description	ARU access register lower data word
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_DATA_L
Address	$0x188$
C–Name	GTM.CLS[0].ARU.DATA_L

Interface: MCS[i]

Name	ARU_DATA_L
Address	$0x188$
C–Name	

DATA	
Description	Lower ARU data word
Loop	–
Bit Range	$[28 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

DATA	
Reset group	HW_RESET: async GTM_RESET: async
	Note: Transfer lower ARU data word addressed by **ARU_ACCESS.ADDR** . The data bits [23:0] of an ARU word are mapped to the data bits [23:0] of this register and the data bits [52:48] of an ARU word are mapped to the data bits [28:24] of this register when data is read by the CPU. **Note:** For writing data into the ARU by the CPU, the bits [28:24] are not transferred to bit [52:48] of the ARU word. Only bits [23:0] are written to bits [23:0] of the ARU word.

5.5.4 ARU_DBG_ACCESS0

Description	ARU debug access channel 0
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_DBG_ACCESS0
Address	$0x18C$
C-Name	GTM.CLS[0].ARU.DBG_ACCESS0

Interface: MCS[i]

Name	ARU_DBG_ACCESS0
Address	$0x18C$
C-Name	

ADDR	
Description	ARU debugging address
Loop	-
Bit Range	$[8:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x1FE$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: Define address of ARU debugging channel 0.

5.5.5 ARU_DBG_DATA0_H

Description	ARU debug access 0 transfer register upper data word
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_DBG_DATA0_H
Address	$0x190$
C-Name	GTM.CLS[0].ARU.DBG_DATA0_H

Interface: MCS[i]

Name	ARU_DBG_DATA0_H
Address	$0x190$
C-Name	

DATA	
Description	Upper debug data word
Loop	-
Bit Range	$[28 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: Transfer upper ARU data word addressed by register **ARU_DBG_ACCESS0** . The data bits [52:24] of an ARU word are mapped to the data bits [28:0] of this register. **Note:** The interrupt *ARU_NEW_DATA0_IRQ* is raised if a new data word is available.

5.5.6 ARU_DBG_DATA0_L

Description	ARU debug access 0 transfer register lower data word
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER

Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_DBG_DATA0_L
Address	$0x194$
C-Name	GTM.CLS[0].ARU.DBG_DATA0_L

Interface: MCS[i]

Name	ARU_DBG_DATA0_L
Address	$0x194$
C-Name	

DATA	
Description	Lower debug data word
Loop	-
Bit Range	$[28:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: Transfer lower ARU data word addressed by register **ARU_DBG_ACCESS0** . The data bits [23:0] of an ARU word are mapped to the data bits [23:0] of this register and the data bits [52:48] of an ARU word is mapped to the data bits [28:24] of this register. **Note:** The interrupt *ARU_NEW_DATA0_IRQ* is raised if a new data word is available.

5.5.7 ARU_DBG_ACCESS1

Description	ARU debug access channel 0
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_DBG_ACCESS1
Address	$0x198$

C-Name	GTM.CLS[0].ARU.DBG_ACCESS1

Interface: MCS[i]

Name	ARU_DBG_ACCESS1
Address	$0x198$
C-Name	

ADDR	
Description	ARU debugging address
Loop	-
Bit Range	$[8:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x1FE$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: Define address of ARU debugging channel 1.

5.5.8 ARU_DBG_DATA1_H

Description	ARU debug access 1 transfer register upper data word
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_DBG_DATA1_H
Address	$0x19C$
C-Name	GTM.CLS[0].ARU.DBG_DATA1_H

Interface: MCS[i]

Name	ARU_DBG_DATA1_H
Address	$0x19C$
C-Name	

DATA	
Description	Upper debug data word
Loop	-

DATA	
Bit Range	$[28:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: Transfer upper ARU data word addressed by register **ARU_DBG_ACCESS1** . The data bits [52:24] of an ARU word are mapped to the data bits [28:0] of this register. **Note:** The interrupt *ARU_NEW_DATA1_IRQ* is raised if a new data word is available.

5.5.9 ARU_DBG_DATA1_L

Description	ARU debug access 1 transfer register lower data word
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_DBG_DATA1_L
Address	$0x1A0$
C–Name	GTM.CLS[0].ARU.DBG_DATA1_L

Interface: MCS[i]

Name	ARU_DBG_DATA1_L
Address	$0x1A0$
C–Name	

DATA	
Description	Lower debug data word
Loop	–
Bit Range	$[28:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

DATA	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: Transfer lower ARU data word addressed by register **ARU_DBG_ACCESS1** . The data bits [23:0] of an ARU word are mapped to the data bits [23:0] of this register and the data bits [52:48] of an ARU word is mapped to the data bits [28:24] of this register. **Note:** The interrupt *ARU_NEW_DATA1_IRQ* is raised if a new data word is available.

5.5.10 ARU_IRQ_NOTIFY

Description	ARU interrupt notification register
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_IRQ_NOTIFY
Address	$0x1A4$
C-Name	GTM.CLS[0].ARU.IRQ_NOTIFY

Interface: MCS[i]

Name	ARU_IRQ_NOTIFY
Address	$0x1A4$
C-Name	

NEW_DATA0	
Description	Data was transferred for addr ARU_DBG_ACCESS0
Loop	–
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised

NEW_DATA0	
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit is cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

NEW_DATA1	
Description	Data was transferred for addr ARU_DBG_ACCESS1
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit is cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

ACC_ACK	
Description	AEI to ARU access finished, on read access data are valid
Loop	-
Bit Range	[2 : 2]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit is cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

5.5.11 ARU_IRQ_EN

Description	ARU interrupt enable register
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_IRQ_EN
Address	$0x1A8$
C-Name	GTM.CLS[0].ARU.IRQ_EN

Interface: MCS[i]

Name	ARU_IRQ_EN
Address	$0x1A8$
C-Name	

NEW_DATA0_IRQ_EN	
Description	ARU_NEW_DATA0_IRQ interrupt enable
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

NEW_DATA1_IRQ_EN	
Description	ARU_NEW_DATA1_IRQ interrupt enable
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

NEW_DATA1_IRQ_EN	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

ACC_ACK_IRQ_EN	
Description	ARU_ACC_ACK_IRQ interrupt enable
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

5.5.12 ARU_IRQ_FORCINT

Description	ARU force interrupt register
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_IRQ_FORCINT
Address	$0x1AC$
C-Name	GTM.CLS[0].ARU.IRQ_FORCINT

Interface: MCS[i]

Name	ARU_IRQ_FORCINT
Address	$0x1AC$
C-Name	

TRG_NEW_DATA0	
Description	Trigger the bit ARU_IRQ_NOTIFY.NEW_DATA0 by software
Loop	–
Bit Range	$[0:0]$

TRG_NEW_DATA0	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 2, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **ARU_IRQ_NOTIFY.NEW_DATA0**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

TRG_NEW_DATA1	
Description	Trigger the bit ARU_IRQ_NOTIFY.NEW_DATA1 by software
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 2, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **ARU_IRQ_NOTIFY.NEW_DATA1**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

TRG_ACC_ACK	
Description	Trigger the bit ARU_IRQ_NOTIFY.ACC_ACK by software
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False

TRG_ACC_ACK	
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 2, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **ARU_IRQ_NOTIFY.ACC_ACK**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT**

5.5.13 ARU_IRQ_MODE

Description	ARU interrupt mode register
Loop	
Condition	$NARU > 0$
Storage Type	REGISTER
Clock Active Cond	$ARU\_CLK\_ENABLE == 1$

Interface: CPU

Name	ARU_IRQ_MODE
Address	$0x1B0$
C-Name	GTM.CLS[0].ARU.IRQ_MODE

Interface: MCS[i]

Name	ARU_IRQ_MODE
Address	$0x1B0$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	–
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	–

IRQ_MODE	
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

5.5.14 ARU_CADDR_END

Description	ARU caddr counter end value
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_CADDR_END
Address	$0x1B4$
C-Name	GTM.CLS[0].ARU.CADDR_END

Interface: MCS[i]

Name	ARU_CADDR_END
Address	$0x1B4$
C-Name	

CADDR_END	
Description	Set end value of ARU caddr counter
Loop	-
Bit Range	$[6:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$\mathrm{CADDR\_END\_RST\_VAL}$
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async

CADDR_END	
	Note: The ARU roundtrip counter aru_caddr runs from zero to caddr_end value. **Note:** Shortening the ARU roundtrip cycle by setting a smaller number than the defined reset value does not serve all ARU_connected modules. **Note:** Making the roundtrip cycle longer than the reset value causes longer ARU roundtrip time and as a result some ARU_connected modules are not served as fast as possible for this device. **Note:** The reset value is device-specific. For more information, please refer to Appendix B. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

5.5.15 ARU_CADDR

Description	ARU caddr counter value
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_CADDR
Address	$0x1FC$
C-Name	GTM.CLS[0].ARU.CADDR

Interface: MCS[i]

Name	ARU_CADDR
Address	$0x1FC$
C-Name	

CADDR_0	
Description	Value of ARU_0 caddr counter
Loop	-
Bit Range	$[6:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

CADDR_0	
Reset group	HW_RESET: async GTM_RESET: async

CADDR_1	
Description	Value of ARU_1 caddr counter
Loop	–
Bit Range	$[22 : 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

Note:
The registers **ARU_CADDR.CADDR_0** and **ARU_CADDR.CADDR_1** start incrementing with each clock cycle just after reset. Due to this the initial reset value are not read back.

5.5.16 ARU_CTRL

Description	ARU enable dynamic routing
Loop	
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_CTRL
Address	$0x1BC$
C-Name	GTM.CLS[0].ARU.CTRL

Interface: MCS[i]

Name	ARU_CTRL
Address	$0x1BC$
C-Name	

ARU_[k]_DYN_EN	
Description	Enable dynamic routing for ARU of sub-channel k
Loop	$k = \{n : 0 <= n <= 1\}$
Bit Range	$[2 * k + 1 : 2 * k]$
Access Type	RW

ARU_[k]_DYN_EN	
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W–Coding	0b00 : don't care, bits will not be changed 0b01 : disable dynamic routing 0b10 : enable dynamic routing 0b11 : don't care, bits will not be changed
R–Coding	0b00 : disabled dynamic routing 0b01 : unused 0b10 : unused 0b11 : enabled dynamic routing
	Dynamic routing enable of ARU of sub–channel k. **Note:** If dynamic routing is disabled, the normal ARU routing scheme for ARU of sub–channel k is executed.

ARU_DYN_RING_MODE	
Description	Enable dynamic routing ring mode
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : different dynamic routing scheme for ARU_0 and ARU_1 1 : same dynamic routing scheme for ARU_0 and ARU_1 with 24 possible read–ID's (dynamic routing ring mode)
	Dynamic routing ring mode for both ARU_0 and ARU_1

5.5.17 ARU_[g]_DYN_CTRL

Description	ARU [g] dynamic routing control register
Loop	$g = \{n : 0 <= n <= 1\}$
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER

Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_[g]_DYN_CTRL
Address	$0x4 * g + 0x1C0$
C-Name	GTM.CLS[0].ARU.DYN_CTRL[g]

Interface: MCS[i]

Name	ARU_[g]_DYN_CTRL
Address	$0x4 * g + 0x1C0$
C-Name	

DYN_ARU_UPDATE_EN	
Description	Enable reload of DYN_ROUTE register from ARU itself
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Enable reload of **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** register from ARU itself.

DYN_ROUTE_SWAP	
Description	Enable swapping DYN_ROUTE_SR with DYN_ROUTE register
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Enable swapping of register **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** with register **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** .

5.5.18 ARU_[g]_DYN_RDADDR

Description	ARU [g] master ID for dynamic routing
Loop	$g = \{n : 0 <= n <= 1\}$
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_[g]_DYN_RDADDR
Address	$0x4 * g + 0x1E8$
C-Name	GTM.CLS[0].ARU.DYN_RDADDR[g]

Interface: MCS[i]

Name	ARU_[g]_DYN_RDADDR
Address	$0x4 * g + 0x1E8$
C-Name	

DYN_ARU_RDADDR	
Description	ARU read address ID to reload the dynamic routing register
Loop	-
Bit Range	$[8 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	ARU read address ID to reload the **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** register from ARU itself.

5.5.19 ARU_[g]_DYN_ROUTE_LOW

Description	ARU [g] lower bits of DYN_ROUTE register
Loop	$g = \{n : 0 <= n <= 1\}$
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_[g]_DYN_ROUTE_LOW
Address	$0x4 * g + 0x1C8$
C-Name	GTM.CLS[0].ARU.DYN_ROUTE_LOW[g]

Interface: MCS[i]

Name	ARU_[g]_DYN_ROUTE_LOW
Address	$0x4 * g + 0x1C8$
C-Name	

DYN_READ_ID0	
Description	ARU master ID_0
Loop	-
Bit Range	$[7:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 0 for dynamic routing

DYN_READ_ID1	
Description	ARU master ID_1
Loop	-
Bit Range	$[15:8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 1 for dynamic routing

DYN_READ_ID2	
Description	ARU master ID_2
Loop	-

DYN_READ_ID2	
Bit Range	$[23 : 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 2 for dynamic routing

5.5.20 ARU_[g]_DYN_ROUTE_HIGH

Description	ARU [g] higher bits of DYN_ROUTE register
Loop	$g = \{n : 0 <= n <= 1\}$
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_[g]_DYN_ROUTE_HIGH
Address	$0x4 * g + 0x1D0$
C-Name	GTM.CLS[0].ARU.DYN_ROUTE_HIGH[g]

Interface: MCS[i]

Name	ARU_[g]_DYN_ROUTE_HIGH
Address	$0x4 * g + 0x1D0$
C-Name	

DYN_READ_ID3	
Description	ARU master ID_3
Loop	–
Bit Range	$[7 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

DYN_READ_ID3	
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 3 for dynamic routing

DYN_READ_ID4	
Description	ARU master ID_4
Loop	–
Bit Range	[15 : 8]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 4 for dynamic routing

DYN_READ_ID5	
Description	ARU master ID_5
Loop	–
Bit Range	[23 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 5 for dynamic routing

DYN_CLK_WAIT	
Description	Number of clk cycles for dynamic routing
Loop	–
Bit Range	[27 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

DYN_CLK_WAIT	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Defines the number of clk cycles between each dynamic routing ID.

5.5.21 ARU_[g]_DYN_ROUTE_SR_LOW

Description	ARU [g] shadow register of DYN_ROUTE register lower bits
Loop	$g = \{n : 0 <= n <= 1\}$
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_[g]_DYN_ROUTE_SR_LOW
Address	$0x4 * g + 0x1D8$
C–Name	GTM.CLS[0].ARU.DYN_ROUTE_SR_LOW[g]

Interface: MCS[i]

Name	ARU_[g]_DYN_ROUTE_SR_LOW
Address	$0x4 * g + 0x1D8$
C–Name	

DYN_READ_ID6	
Description	ARU master ID_6
Loop	–
Bit Range	$[7 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 6 for dynamic routing **Note:** These bits are mapped to ARU data bits *ARU_DATA* [7:0].

DYN_READ_ID7	
Description	ARU master ID_7
Loop	–
Bit Range	$[15 : 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 7 for dynamic routing **Note:** These bits are mapped to ARU data bits *ARU_DATA* [15:8].

DYN_READ_ID8	
Description	ARU master ID_8
Loop	–
Bit Range	$[23 : 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 8 for dynamic routing **Note:** These bits are mapped to ARU data bits *ARU_DATA* [23:16].

Note:
This is the shadow register for register **ARU_[g]_DYN_ROUTE_LOW** .

5.5.22 ARU_[g]_DYN_ROUTE_SR_HIGH

Description	ARU [g] shadow register of DYN_ROUTE register higher bits
Loop	$g = \{n : 0 <= n <= 1\}$
Condition	$\mathrm{NARU} > 0$
Storage Type	REGISTER

Clock Active Cond	$\text{ARU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ARU_[g]_DYN_ROUTE_SR_HIGH
Address	$0x4 * g + 0x1E0$
C-Name	GTM.CLS[0].ARU.DYN_ROUTE_SR_HIGH[g]

Interface: MCS[i]

Name	ARU_[g]_DYN_ROUTE_SR_HIGH
Address	$0x4 * g + 0x1E0$
C-Name	

DYN_READ_ID9	
Description	ARU master ID_9
Loop	-
Bit Range	$[7:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 9 for dynamic routing **Note:** These bits are mapped to ARU data bits *ARU_DATA* [31:24].

DYN_READ_ID10	
Description	ARU master ID_10
Loop	-
Bit Range	$[15:8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

DYN_READ_ID10	
	ARU master ID 10 for dynamic routing **Note:** These bits are mapped to ARU data bits *ARU_DATA* [39:32].

DYN_READ_ID11	
Description	ARU master ID_11
Loop	–
Bit Range	[23 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	ARU master ID 11 for dynamic routing **Note:** These bits are mapped to ARU data bits *ARU_DATA* [47:40].

DYN_CLK_WAIT	
Description	Number of clk cycles for dynamic routing
Loop	–
Bit Range	[27 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Defines the number of clk cycles between each dynamic routing ID **Note:** These bits are mapped to ARU data bits *ARU_DATA* [51:48].

DYN_UPDATE_EN	
Description	Update enable from shadow register
Loop	–
Bit Range	[28 : 28]

DYN_UPDATE_EN	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Enable update **ARU_[g]_DYN_ROUTE_LOW** / **ARU_[g]_DYN_ROUTE_HIGH** registers from shadow registers **ARU_[g]_DYN_ROUTE_SR_LOW** / **ARU_[g]_DYN_ROUTE_SR_HIGH** . **Note:** This bit is mapped to ARU data bits *ARU_DATA* [52:52].

Note:
This is the shadow register for register **ARU_[g]_DYN_ROUTE_HIGH** .

5.6 ARU Signal Description

5.6.1 ARU Interrupt Signals

Loop	–
Condition	–
Format	–

ARU_ACC_ACK_IRQ	
Description	ARU access acknowledge IRQ.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ARU_NEW_DATA0_IRQ	
Description	Indicates that data is transferred through the ARU using debug channel ARU_DBG_ACCESS0.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ARU_NEW_DATA1_IRQ	
Description	Indicates that data is transferred through the ARU using debug channel ARU_DBG_ACCESS1.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

5.6.2 ARU Internal Signals

Loop	–
Condition	–
Format	–

ARU_DATA	
Description	Received ARU DATA word, to be processed by the dynamic routing.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

6 Broadcast Module (BRC)

6.1 Overview

Since each write address for the submodule channels of the GTM-IP that can write to the ARU can only be read by a single module, it is not possible to make a data stream available to different modules in parallel. (This statement does not apply to sources that do not invalidate their data after the data has been read by any consumer, e.g. DPLL).

To overcome this issue for regular modules, the submodule Broadcast (BRC) enables duplication of data streams multiple times.

The BRC submodule provides 12 input channels as well as 22 output channels.

Indices and their range as used inside this chapter are:

- i:={0, 1,..., NBRC} index of cluster / module
- x:={0, 1,..., 11} index for the 12 BRC submodule input channels
- z:={0, 1,..., 21} index for the 22 BRC submodule output channels

In order to clone an incoming data stream, the corresponding input channel can be mapped to zero or more output channels.

When mapped to zero no channel is read.

To destroy an incoming data stream, the bit **BRC_SRC_[x]_DEST.EN_TRASHBIN** has to be set.

The total number of output channels that are assigned to a single input channel is variable. However, the total number of assigned output channels must be less than or equal to 22.

6.2 BRC Configuration

As it is the case with all other submodules connected to the ARU, the input channels can read arbitrary ARU address locations and the output channels provide the broadcast data to fixed ARU write address locations.

The associated write addresses for the BRC submodule are fixed and can be obtained from chapter *93 "ARU Write Address Overview"* .

The read address for each input channel is defined by the corresponding register **BRC_SRC_[x]_ADDR** .

The mapping of an input channel to several output channels is defined by setting the appropriate bits in the register **BRC_SRC_[x]_DEST** . Each output channel is represented by a single bit in the register **BRC_SRC_[x]_DEST** . The address of the output channel is defined in chapter *93 "ARU Write Address Overview"* .

If no output channel bit is set within a register **BRC_SRC_[x]_DEST** , no data is provided to the corresponding ARU write address location from the defined read input specified by **BRC_SRC_[x]_ADDR** . This means that the channel does not broadcast any data and is disabled (reset state).

Besides the possibility of mapping an input channel to several output channels, the bit **BRC_SRC_[x]_DEST.EN_TRASHBIN** is set, which results in dropping an incoming data stream. In this case, the data of an input channel defined by **BRC_SRC_[x]_ADDR** is consumed by the BRC module and not routed to any succeeding submodule. Consequently, the output channels defined in the register **BRC_SRC_[x]_DEST** are ignored. Therefore, the bit **BRC_SRC_[x]_DEST.EN_DEST[z]** is set to zero (0) when trash bin functionality is enabled.

In general, the BRC submodule can work in two independent operation modes. In the first operation mode, the data consistency is guaranteed since a BRC channel requests only new data from a source when all destination channels for the BRC have consumed the old data value. This mode is called Data Consistency Mode (DCM).

In a second operation mode the BRC channel always requests data from a source and distributes this data to the destination regardless of whether all destinations have already consumed the old data. This mode is called Maximum Throughput Mode (MTM).

MTM ensures that always the newest available data is routed through the system, while this does not guarantee data consistency since some of the destination channels can be provided with the old data while some other destination channels are provided with the new data. If this is the case, the Data Inconsistency Detected Interrupt *BRC_DID_IRQ* [x:x] is raised but the channel continues to work.

Furthermore, in MTM mode it is guaranteed that it is not possible to read data twice by a read channel. This is blocked.

The channel mode can be configured inside the **BRC_SRC_[x]_ADDR** register.

To avoid invalid configurations of the registers **BRC_SRC_[x]_DEST** , the BRC also implements a plausibility check for these configurations. If the software assigns an already used output channel to a second input channel, BRC performs an auto correction of the lastly configured register **BRC_SRC_[x]_DEST** and it triggers the interrupt *BRC_DEST_ERR_IRQ* .

Consider the following example for clarification of the auto correction mechanism. Assume that the following configuration of the 22 lower significant bits for the registers **BRC_SRC_[x]_DEST** :

- **BRC_SRC_[0]_DEST** : 00 0000 0000 1000 1000 0000 (binary)
- **BRC_SRC_[1]_DEST** : 00 0000 0000 0100 0000 0100 (binary)

- **BRC_SRC_[2]_DEST** : 00 0000 0000 0001 0100 0010 (binary)
- **BRC_SRC_[3]_DEST** : 00 0000 0000 0010 0001 1001 (binary)

If the software overwrites the value for register **BRC_SRC_[2]_DEST** with

- **BRC_SRC_[2]_DEST** : 00 0000 0000 1001 0010 0010 (binary)

(changed bits are underlined), then the BRC releases an *BRC_DEST_ERR_IRQ* interrupt since bit [11:11] is already assigned in register **BRC_SRC_[0]_DEST** . The auto correction forces bit [11:11] to be cleared. The modifications of the bits [6:5] are accepted, since there is no violation with previous configurations. So, the result of the write access mentioned above is the following modified register configuration:

- **BRC_SRC_[2]_DEST** : 00 0000 0000 0001 0010 0010 (binary)

For debug purposes, the interrupt *BRC_DEST_ERR_IRQ* can also be released by writing to register **BRC_IRQ_FORCINT** . Nevertheless, the interrupt has to be enabled to be visible outside of the GTM-IP.

6.3 BRC Interrupt Signals

Table 12 BRC Interrupt Signals Table

Signal	Description
BRC_DEST_ERR_IRQ	Indicating configuration errors for BRC module
BRC_DID_IRQ[x]	Data inconsistency occurred in MTM mode

6.4 BRC Software Reset

The configuration register **BRC_RST** is used for software reset of the BRC module.

Writing the value 0b1 to the bitfield **BRC_RST.RST** via the configuration interface immediately resets the content of the following registers to the initial hardware reset state x.

- **BRC_SRC_[x]_ADDR**
- **BRC_SRC_[x]_DEST**
- **BRC_IRQ_NOTIFY**
- **BRC_IRQ_EN**
- **BRC_IRQ_MODE**
- **BRC_EIRQ_EN**

6.5 BRC Configuration Registers Description

6.5.1 BRC_SRC_[x]_ADDR

Description	BRC read address for input channel [x]
Loop	$x = \{n : 0 <= n <= 11\}$
Condition	$\mathrm{NBRC} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{BRC\_CLK\_ENABLE} == 1$

Interface: CPU

Name	BRC_SRC_[x]_ADDR
Address	$0x8 * x + 0x200$
C-Name	GTM.CLS[0].BRC.SRC[x].ADDR

Interface: MCS[i]

Name	BRC_SRC_[x]_ADDR
Address	$0x8 * x + 0x200$
C-Name	

ADDR	
Description	Source ARU address. Define an ARU read address used as data source for input channel[x].
Loop	-
Bit Range	$[8 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x1FE$
Protect Enable Cond	BRC_SRC_[x]_DEST.EN_DEST[0] || BRC_SRC_[x]_DEST.EN_DEST[1] || BRC_SRC_[x]_DEST.EN_DEST[2] || BRC_SRC_[x]_DEST.EN_DEST[3] || BRC_SRC_[x]_DEST.EN_DEST[4] || BRC_SRC_[x]_DEST.EN_DEST[5] || BRC_SRC_[x]_DEST.EN_DEST[6] || BRC_SRC_[x]_DEST.EN_DEST[7] || BRC_SRC_[x]_DEST.EN_DEST[8] || BRC_SRC_[x]_DEST.EN_DEST[9] || BRC_SRC_[x]_DEST.EN_DEST[10] || BRC_SRC_[x]_DEST.EN_DEST[11] || BRC_SRC_[x]_DEST.EN_DEST[12] || BRC_SRC_[x]_DEST.EN_DEST[13] || BRC_SRC_[x]_DEST.EN_DEST[14] || BRC_SRC_[x]_DEST.EN_DEST[15] || BRC_SRC_[x]_DEST.EN_DEST[16] || BRC_SRC_[x]_DEST.EN_DEST[17] || BRC_SRC_[x]_DEST.EN_DEST[18] || BRC_SRC_[x]_DEST.EN_DEST[19] || BRC_SRC_[x]_DEST.EN_DEST[20] || BRC_SRC_[x]_DEST.EN_DEST[21]
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
	Note: This bit field is only writable if channel is disabled.

BRC_MODE	
Description	BRC Operation mode select
Loop	-
Bit Range	$[12 : 12]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	BRC_SRC_[x]_DEST.EN_DEST[0] || BRC_SRC_[x]_DEST.EN_DEST[1] || BRC_SRC_[x]_DEST.EN_DEST[2] || BRC_SRC_[x]_DEST.EN_DEST[3] || BRC_SRC_[x]_DEST.EN_DEST[4] || BRC_SRC_[x]_DEST.EN_DEST[5] || BRC_SRC_[x]_DEST.EN_DEST[6] || BRC_SRC_[x]_DEST.EN_DEST[7] || BRC_SRC_[x]_DEST.EN_DEST[8] || BRC_SRC_[x]_DEST.EN_DEST[9] || BRC_SRC_[x]_DEST.EN_DEST[10] || BRC_SRC_[x]_DEST.EN_DEST[11] || BRC_SRC_[x]_DEST.EN_DEST[12] || BRC_SRC_[x]_DEST.EN_DEST[13] || BRC_SRC_[x]_DEST.EN_DEST[14] || BRC_SRC_[x]_DEST.EN_DEST[15] || BRC_SRC_[x]_DEST.EN_DEST[16] || BRC_SRC_[x]_DEST.EN_DEST[17] || BRC_SRC_[x]_DEST.EN_DEST[18] || BRC_SRC_[x]_DEST.EN_DEST[19] || BRC_SRC_[x]_DEST.EN_DEST[20] || BRC_SRC_[x]_DEST.EN_DEST[21]
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
RW-Coding	0 : Data Consistency Mode (DCM) selected 1 : Maximum Throughput Mode (MTM) selected

BRC_MODE	
	Note: This bit field is only writable if channel is disabled.

6.5.2 BRC_SRC_[x]_DEST

Description	BRC destination channels for input channel [x]
Loop	$x = \{n : 0 <= n <= 11\}$
Condition	$\mathrm{NBRC} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{BRC\_CLK\_ENABLE} == 1$

Interface: CPU

Name	BRC_SRC_[x]_DEST
Address	$0x8 * x + 0x204$
C-Name	GTM.CLS[0].BRC.SRC[x].DEST

Interface: MCS[i]

Name	BRC_SRC_[x]_DEST
Address	$0x8 * x + 0x204$
C-Name	

EN_DEST[z]	
Description	Enable BRC destination address [z]
Loop	$z = \{n : 0 <= n <= 21\}$
Bit Range	$[z : z]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
RW-Coding	0 : Destination address z not mapped to source **BRC_SRC_[x]_ADDR** 1 : Destination address z mapped to source **BRC_SRC_[x]_ADDR**
	Note: The destination address z for BRC channel is defined in table *93 "ARU Write Address Overview"*

EN_TRASHBIN	
Description	Control trash bin functionality.
Loop	–
Bit Range	$[22 : 22]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
RW-Coding	0 : Trash bin functionality disabled 1 : Trash bin functionality enabled
	Note: When bit **BRC_SRC_[z]_DEST.EN_TRASHBIN** is enabled bits [21:0] are ignored for this input channel. Therefore, the bits [21:0] are set to zero (0) when trash bin functionality is enabled.

Note:
The bits 0 to 21 are cleared by auto correction mechanism if a destination channel is assigned to multiple source channels.

Note:
When a BRC input channel is disabled (all **BRC_SRC_[x]_DEST.EN_DEST[z]** bits are reset to zero) the internal states are reset to their reset value.

6.5.3 BRC_IRQ_NOTIFY

Description	BRC interrupt notification register
Loop	
Condition	$\mathrm{NBRC} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{BRC\_CLK\_ENABLE} == 1$

Interface: CPU

Name	BRC_IRQ_NOTIFY
Address	$0x260$
C-Name	GTM.CLS[0].BRC.IRQ_NOTIFY

Interface: MCS[i]

Name	BRC_IRQ_NOTIFY
Address	$0x260$
C-Name	

DEST_ERR	
Description	Configuration error interrupt for BRC submodule
Loop	–

DEST_ERR	
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

DID[x]	
Description	Data inconsistency occurred in MTM mode for channel [x].
Loop	$x = \{n : 0 <= n <= 11\}$
Bit Range	$[x+1 : x+1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

6.5.4 BRC_IRQ_EN

Description	BRC interrupt enable register
Loop	
Condition	$\mathrm{NBRC} > 0$

Storage Type	REGISTER
Clock Active Cond	$BRC\_CLK\_ENABLE == 1$

Interface: CPU

Name	BRC_IRQ_EN
Address	$0x264$
C-Name	GTM.CLS[0].BRC.IRQ_EN

Interface: MCS[i]

Name	BRC_IRQ_EN
Address	$0x264$
C-Name	

DEST_ERR_IRQ_EN	
Description	BRC_DEST_ERR_IRQ interrupt enable
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

DID_IRQ_EN[x]	
Description	Enable BRC_DID_IRQ for channel [x]
Loop	$x = \{n : 0 <= n <= 11\}$
Bit Range	$[x+1 : x+1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync

DID_IRQ_EN[x]	
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

6.5.5 BRC_IRQ_FORCINT

Description	BRC force interrupt register
Loop	
Condition	$\mathrm{NBRC} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{BRC\_CLK\_ENABLE} == 1$

Interface: CPU

Name	BRC_IRQ_FORCINT
Address	$0x268$
C-Name	GTM.CLS[0].BRC.IRQ_FORCINT

Interface: MCS[i]

Name	BRC_IRQ_FORCINT
Address	$0x268$
C-Name	

TRG_DEST_ERR	
Description	Trigger the bit BRC_IRQ_NOTIFY.DEST_ERR by software
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 12, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **BRC_IRQ_NOTIFY.DEST_ERR**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

TRG_DID[x]	
Description	Trigger the bit BRC_IRQ_NOTIFY.DID[x] by software

TRG_DID[x]	
Loop	$x = \{n : 0 <= n <= 11\}$
Bit Range	$[x+1 : x+1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 12, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No status
W–Coding	0 : No action 1 : Force setting of **BRC_IRQ_NOTIFY.DEST_ERR**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

6.5.6 BRC_IRQ_MODE

Description	BRC interrupt mode configuration register
Loop	
Condition	$\mathrm{NBRC} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{BRC\_CLK\_ENABLE} == 1$

Interface: CPU

Name	BRC_IRQ_MODE
Address	$0x26C$
C–Name	GTM.CLS[0].BRC.IRQ_MODE

Interface: MCS[i]

Name	BRC_IRQ_MODE
Address	$0x26C$
C–Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	–
Bit Range	$[1 : 0]$
Access Type	RW

IRQ_MODE	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

6.5.7 BRC_EIRQ_EN

Description	BRC error interrupt enable register
Loop	
Condition	NBRC > 0
Storage Type	REGISTER
Clock Active Cond	BRC_CLK_ENABLE $== 1$

Interface: CPU

Name	BRC_EIRQ_EN
Address	$0x274$
C-Name	GTM.CLS[0].BRC.EIRQ_EN

Interface: MCS[i]

Name	BRC_EIRQ_EN
Address	$0x274$
C-Name	

DEST_ERR_EIRQ_EN	
Description	BRC_DEST_ERR_EIRQ error interrupt enable
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

DEST_ERR_EIRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP 1 : Enable error interrupt, error interrupt is visible outside GTM–IP

DID_EIRQ_EN[x]	
Description	Enable BRC_DID_EIRQ for channel [x]
Loop	$x = \{n : 0 <= n <= 11\}$
Bit Range	$[x + 1 : x + 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async BRC_RESET: sync
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP 1 : Enable error interrupt, error interrupt is visible outside GTM–IP

6.5.8 BRC_RST

Description	BRC software reset register
Loop	
Condition	$\mathrm{NBRC} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{BRC\_CLK\_ENABLE} == 1$

Interface: CPU

Name	BRC_RST
Address	$0x270$
C–Name	GTM.CLS[0].BRC.RST

Interface: MCS[i]

Name	BRC_RST
Address	$0x270$
C–Name	

RST	
Description	Software reset
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : No action 1 : Reset BRC
	Note: This bit is cleared automatically after write by CPU. The channel registers are set to their reset values and channel operation is stopped immediately.

6.6 BRC Signal Description

6.6.1 BRC Software Reset

Loop	$j = \{n : 0 <= n <= \mathrm{NBRC} - 1\}$
Condition	–
Format	–

BRC_RESET	
Description	BRC_RESET active signal
Loop	–
Condition	–
Signal Type	INT
Assignment	BRC_RST.RST == 1

6.6.2 BRC Signals

Loop	–
Condition	–
Format	–

BRC_DEST_ERR_IRQ	
Description	signal for interrupt of configuration error
Loop	–
Condition	–
Signal Type	INT
Assignment	–

BRC_DID_IRQ	
Description	signal for interrupt of data inconsistency occurred in MTM mode
Loop	–
Condition	–
Signal Type	INT
Assignment	–

BRC_DID_IRQ[x]	
Description	signal for interrupt of data inconsistency occurred in MTM mode
Loop	$x = \{n : 0 <= n <= 11\}$
Condition	–
Signal Type	ARRAY[12]
Assignment	–

BRC_DEST_ERR_EIRQ	
Description	signal for error interrupt of configuration error
Loop	–
Condition	–
Signal Type	INT
Assignment	–

BRC_DID_EIRQ	
Description	signal for error interrupt of data inconsistency occurred in MTM mode
Loop	–
Condition	–
Signal Type	INT
Assignment	–

6.7 BRC Port Description

7 First In First Out Module (FIFO)

7.1 Overview

The FIFO unit is the storage part of the PSM sub-module. The F2A described in chapter *9 "FIFO to ARU Unit (F2A)"* and the AFD described in chapter *8 "AEI to FIFO Data Interface (AFD)"* implement the interface part of the FIFO sub-module to the ARU and the AEI bus. Each FIFO unit embeds eight logical FIFO channels. These logical FIFOs are configurable in the following manner:

- FIFO size (defines start and end address)
- FIFO operation modes (normal mode or ring buffer operation mode)
- Fill level control / memory region read protection

Indices as used inside this chapter are :

- i:={0, 1,..., (NPSM-1)} instance index of cluster / module
- x:={0, 1,..., 7} index of logical FIFO channels

Each logical FIFO represents a data stream between the sub-modules of the GTM and the microcontroller connected to AFD sub-module (see section *8 "AEI to FIFO Data Interface (AFD)"*). The FIFO RAM counts 1K words, where the word size is 29 bits. This gives the freedom to program or receive 24 bits of data together with the five control bits inside an ARU data word.

The FIFO unit provides three ports for accessing its content. One port is connected to the F2A interface, one port is connected to the AFD interface and one port has its own AEI bus interface.

The AFD interface has always the highest priority. Accesses to the FIFO from AFD interface and direct AEI interface in parallel, which means at the same time, is not possible, because both interfaces are driven from the same AEI bus interface of the GTM.

The priority between F2A and direct AEI interface can be defined by software. This can be done by using the register **FIFO[i]_CH[x]_CTRL** for all FIFO channels of the sub-module.

The FIFO is organized as a single RAM that is also accessible through the FIFO AEI interface connected to one of the FIFO ports. To provide the direct RAM access, the RAM is mapped into the address space of the microcontroller. The addresses for accessing the RAM via AEI can be found in [1].

After reset, the FIFO RAM isn't initialized by hardware.

The FIFO channels can be flushed individually. Each of the eight FIFO channels can be used either in normal FIFO operation mode or in ring buffer operation mode.

Besides the possibility of flushing each FIFO channel directly, a write access to **FIFO[i]_CH[x]_END_ADDR** or **FIFO[i]_CH[x]_START_ADDR** will also flush the corresponding channel which means that the read and write pointer and also the fill level of the corresponding channel will be reset. As a result of this, existing data in the concerned FIFO channel are no longer valid, thereafter the channel is empty.

7.2 Operation Modes

7.2.1 FIFO Operation Mode

In normal FIFO operation mode the content of the FIFO is written and read in first-in first-out order, where the data is destroyed after it is delivered to the system bus or the F2A sub-module (see section *9 "FIFO to ARU Unit (F2A)"*).

The upper and lower watermark registers (registers **FIFO[i]_CH[x]_UPPER_WM** and **FIFO[i]_CH[x]_LOWER_WM**) are used for controlling the FIFO's fill level. If the fill level falls below the lower watermark or it exceeds the upper watermark, an interrupt signal is triggered by the FIFO sub-module, if enabled inside **FIFO[i]_CH[x]_IRQ_EN** .

The interrupt signals are sent to the Interrupt Concentrator Module (ICM) (see chapter *24 "Interrupt Concentrator Module (ICM)"*). The ICM can also initiate specific DMA transfers.

7.2.2 Ring Buffer Operation Mode

The ring buffer mode can be used to provide a continuous data or configuration stream to the other GTM sub-modules without CPU interaction. In ring buffer mode the FIFO provides a continuous data stream to the F2A sub-module. The first word of the FIFO is delivered first and after the last word is provided by the FIFO to the ARU, the first word can be obtained again.

If in ring buffer mode the read pointer reaches the write pointer, it will again be set to the configured start address. So the read pointer always rotates cyclically between the configured start address of the corresponding FIFO channel (first written data) and the write pointer which points to the last written data of the channel.

It is possible to add data to the FIFO channel via the AEI to FIFO interface (AFD) using the register **AFD[i]_CH[x]_BUF_ACC** while running in ring buffer mode. The new written data will be added in the next ring buffer cycle. However, the register **AFD[i]_CH[x]_BUF_ACC** should not be read in ring buffer mode.

It is recommended to fill the FIFO channel first before enabling the data stream in the FIFO to ARU interface (F2A).

Modifications of the continuous data stream can be achieved by using direct memory access which is provided by the FIFO AEI interface.

7.2.3 DMA Hysteresis Mode

The DMA hysteresis mode can be enabled by setting bit **FIFO[i]_CH[x]_IRQ_MODE.DMA_HYSTERESIS** = 1.

In the DMA hysteresis mode, the lower and upper watermark will be masked to generate the DMA request in the following manner.

If a DMA is writing data to a FIFO (configured by setting bit **FIFO[i]_CH[x]_IRQ_MODE.DMA_HYST_DIR** = 1), the DMA request will be generated by the lower watermark. The upper watermark does not generate a DMA request. The next DMA request will be generated by the next lower watermark until the upper watermark was reached.

If a DMA is reading data from a FIFO (configured by setting bit **FIFO[i]_CH[x]_IRQ_MODE.DMA_HYST_DIR** =0), the DMA request will be generated by the upper watermark. The lower watermark does not generate a DMA request. The next DMA request will be generated by the next upper watermark until the lower watermark was reached.

The watermarks have to achieve the following condition depending on the IRQ mode.

- Level / Pulse / Pulse–Notify mode : upper watermark > lower watermark
- Single–Pulse mode : upper watermark > lower watermark + 1

For the DMA hysteresis mode to work correctly, both lower and upper watermark interrupts must be enabled by setting **FIFO[i]_CH[x]_IRQ_EN.FIFO_LWM_IRQ_EN** =0b1 and **FIFO[i]_CH[x]_IRQ_EN.FIFO_UWM_IRQ_EN** =0b1.

7.3 FIFO Configuration Registers Description

7.3.1 FIFO[i]_CH[x]_CTRL

Description	FIFO[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \mathrm{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{PSM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_CTRL
Address	$0x20000 * i + 0x40 * x + 0x4A00$
C–Name	GTM.CLS[i].PSM.FIFO.CH[x].CTRL

Interface: MCS[i]

Name	FIFO[i]_CH[x]_CTRL
Address	$0x40 * x + 0x4A00$
C–Name	

RBM	
Description	Ring buffer mode enable
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

RBM	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Normal FIFO operation mode 1 : Ring buffer mode

RAP	
Description	RAM access priority
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : FIFO ports have higher access priority than AEI_IF 1 : AEI–IF has higher access priority than FIFO ports
	Note: The RAP bit is only functional in register **FIFO[i]_CH[0]_CTRL** . The priority is defined for all FIFO channels there

FLUSH	
Description	FIFO flush control
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W–Coding	0 : Normal operation 1 : Execute FIFO flush (bit is automatically cleared after flush).

FLUSH	
	Note: A FIFO Flush operation resets the **FIFO[i]_CH[x]_FILL_LEVEL** , **FIFO[i]_CH[x]_WR_PTR** and **FIFO[i]_CH[x]_RD_PTR** registers to their initial values.

WULOCK	
Description	RAM write unlock. Enable/disable direct RAM write access to the memory mapped FIFO region.
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Direct RAM write access disabled 1 : Direct RAM write access enabled
	Note: Only the register bit **FIFO[i]_CH[0]_CTRL.WULOCK** enables/disables the direct RAM write access for all FIFO channel (whole FIFO RAM). The **FIFO[i]_CH[x]_CTRL.WULOCK** bits of the other channels x (x∈{1, 2,..., 7}) are writeable but have no effect.

7.3.2 FIFO[i]_CH[x]_END_ADDR

Description	FIFO[i] channel [x] end address register
Loop	$i = \{n : 0 <= n <= \mathrm{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{PSM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_END_ADDR
Address	$0x20000 * i + 0x40 * x + 0x4A04$
C–Name	GTM.CLS[i].PSM.FIFO.CH[x].END_ADDR

Interface: MCS[i]

Name	FIFO[i]_CH[x]_END_ADDR
Address	$0x40 * x + 0x4A04$
C–Name	

ADDR	
Description	End address for FIFO channel [x]

ADDR	
Loop	–
Bit Range	$[9:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$128 * (x + 1) - 1$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: Value for **FIFO[i]_CH[x]_END_ADDR.ADDR** is calculated as **FIFO[i]_CH[x]_END_ADDR.ADDR** = 128*(x+1)-1 **Note:** A write access will flush the corresponding channel

7.3.3 FIFO[i]_CH[x]_START_ADDR

Description	FIFO[i] channel [x] start address register
Loop	$i = \{n : 0 <= n <= \mathrm{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{PSM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_START_ADDR
Address	$0x20000 * i + 0x40 * x + 0x4A08$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].START_ADDR

Interface: MCS[i]

Name	FIFO[i]_CH[x]_START_ADDR
Address	$0x40 * x + 0x4A08$
C-Name	

ADDR	
Description	Start address for FIFO channel [x]
Loop	–
Bit Range	$[9:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

ADDR	
Initial value	$128 * x$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: Initial value for **FIFO[i]_CH[x]_START_ADDR.ADDR** is calculated as **FIFO[i]_CH[x]_START_ADDR.ADDR** = 1-28*x **Note:** A write access will flush the corresponding channel

7.3.4 FIFO[i]_CH[x]_UPPER_WM

Description	FIFO[i] channel [x] upper watermark register
Loop	$i = \{n : 0 <= n <= \mathrm{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{PSM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_UPPER_WM
Address	$0x20000 * i + 0x40 * x + 0x4A0C$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].UPPER_WM

Interface: MCS[i]

Name	FIFO[i]_CH[x]_UPPER_WM
Address	$0x40 * x + 0x4A0C$
C-Name	

ADDR	
Description	Upper watermark address
Loop	–
Bit Range	$[9 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0x60$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

ADDR	
	Note: The upper watermark is configured as a relative fill level of the FIFO. **FIFO[i]_CH[x]_UPPER_WM.ADDR** must be in range: 0 <= **FIFO[i]_CH[x]_UPPER_WM.ADDR** <= **FIFO[i]_CH[x]_END_ADDR** - **FIFO[i]_CH[x]_START_ADDR** . Initial value for **FIFO[i]_CH[x]_UPPER_WM.ADDR** is defined as **FIFO[i]_CH[x]_UPPER_WM.ADDR** = 0x60.

7.3.5 FIFO[i]_CH[x]_LOWER_WM

Description	FIFO[i] channel [x] lower watermark register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_LOWER_WM
Address	$0x20000 * i + 0x40 * x + 0x4A10$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].LOWER_WM

Interface: MCS[i]

Name	FIFO[i]_CH[x]_LOWER_WM
Address	$0x40 * x + 0x4A10$
C-Name	

ADDR	
Description	Lower watermark address
Loop	-
Bit Range	$[9 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x20$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: The lower watermark is configured as a relative fill level of the FIFO. **FIFO[i]_CH[x]_LOWER_WM.ADDR** must be in range: 0 <= **FIFO[i]_CH[x]_LOWER_WM.ADDR** <= **FIFO[i]_CH[x]_END_ADDR** - **FIFO[i]_CH[x]_START_ADDR** . Initial value for **FIFO[i]_CH[x]_LOWER_WM.ADDR** is defined as **FIFO[i]_CH[x]_LOWER_WM.ADDR** = 0x20.

7.3.6 FIFO[i]_CH[x]_STATUS

Description	FIFO[i] channel [x] status register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_STATUS
Address	$0x20000 * i + 0x40 * x + 0x4A14$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].STATUS

Interface: MCS[i]

Name	FIFO[i]_CH[x]_STATUS
Address	$0x40 * x + 0x4A14$
C-Name	

EMPTY	
Description	FIFO is empty
Loop	-
Bit Range	$[0 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Fill level > 0 1 : Fill level = 0
	Note: Bit only applicable in normal mode

FULL	
Description	FIFO is full
Loop	-
Bit Range	$[1 : 1]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-

FULL	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Fill level < **FIFO[i]_CH[x]_END_ADDR** – **FIFO[i]_CH[x]_START_ADDR** + 1 1 : Fill level = **FIFO[i]_CH[x]_END_ADDR** – **FIFO[i]_CH[x]_START_ADDR** + 1
	Note: Bit only applicable in normal mode

LOW_WM	
Description	Lower watermark reached
Loop	–
Bit Range	[2 : 2]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Fill level > lower watermark 1 : Fill level <= lower watermark
	Note: Bit only applicable in normal mode

UP_WM	
Description	Upper watermark reached
Loop	–
Bit Range	[3 : 3]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Fill level < upper watermark 1 : Fill level >= upper watermark

UP_WM	
	Note: Bit only applicable in normal mode

7.3.7 FIFO[i]_CH[x]_FILL_LEVEL

Description	FIFO[i] channel [x] fill level register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_FILL_LEVEL
Address	$0x20000 * i + 0x40 * x + 0x4A18$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].FILL_LEVEL

Interface: MCS[i]

Name	FIFO[i]_CH[x]_FILL_LEVEL
Address	$0x40 * x + 0x4A18$
C-Name	

LEVEL	
Description	Fill level of the current FIFO
Loop	-
Bit Range	$[10 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: **FIFO[i]_CH[x]_FILL_LEVEL.LEVEL** is in range: 0 <= **FIFO[i]_CH[x]_FILL_LEVEL.LEVEL** <= **FIFO[i]_CH[x]_END_ADDR** - **FIFO[i]_CH[x]_START_ADDR** + 1. Register content is compared with the upper and lower watermark values for this channel to detect watermark overflow and underflow.

7.3.8 FIFO[i]_CH[x]_WR_PTR

Description	FIFO[i] channel [x] write pointer register

Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_WR_PTR
Address	$0x20000 * i + 0x40 * x + 0x4A1C$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].WR_PTR

Interface: MCS[i]

Name	FIFO[i]_CH[x]_WR_PTR
Address	$0x40 * x + 0x4A1C$
C-Name	

ADDR	
Description	Position of the write pointer
Loop	-
Bit Range	$[9 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$128 * x$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: **FIFO[i]_CH[x]_WR_PTR.ADDR** must be in range 0 <= **FIFO[i]_CH[x]_WR_PTR.ADDR** <= 1023. Initial value for **FIFO[i]_CH[x]_WR_PTR.ADDR** is defined as **FIFO[i]_CH[x]_WR_PTR.ADDR** = **FIFO[i]_CH[x]_START_ADDR**

7.3.9 FIFO[i]_CH[x]_RD_PTR

Description	FIFO[i] channel [x] read pointer register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_RD_PTR
Address	$0x20000 * i + 0x40 * x + 0x4A20$

C-Name	GTM.CLS[i].PSM.FIFO.CH[x].RD_PTR

Interface: MCS[i]

Name	FIFO[i]_CH[x]_RD_PTR
Address	$0x40 * x + 0x4A20$
C-Name	

ADDR	
Description	Position of the read pointer
Loop	-
Bit Range	$[9:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$128 * x$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: **FIFO[i]_CH[x]_RD_PTR.ADDR** must be in range 0 <= **FIFO[i]_CH[x]_RD_PTR.ADDR** <= 1023. Initial value for **FIFO[i]_CH[x]_RD_PTR.ADDR** is defined as **FIFO[i]_CH[x]_RD_PTR.ADDR** = **FIFO[i]_CH[x]_START_ADDR**

7.3.10 FIFO[i]_CH[x]_IRQ_NOTIFY

Description	FIFO[i] channel [x] interrupt notification register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_IRQ_NOTIFY
Address	$0x20000 * i + 0x40 * x + 0x4A24$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].IRQ_NOTIFY

Interface: MCS[i]

Name	FIFO[i]_CH[x]_IRQ_NOTIFY
Address	$0x40 * x + 0x4A24$
C-Name	

FIFO_EMPTY	
Description	FIFO is empty

FIFO_EMPTY	
Loop	–
Bit Range	[0 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

FIFO_FULL	
Description	FIFO is full
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

FIFO_LWM	
Description	FIFO Lower watermark was under-run
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	True

FIFO_LWM	
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

FIFO_UWM	
Description	FIFO Upper watermark was overrun
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

7.3.11 FIFO[i]_CH[x]_IRQ_EN

Description	FIFO[i] channel [x] interrupt enable register
Loop	$i = \{n : 0 <= n <= \mathrm{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{PSM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_IRQ_EN

Address	$0x20000 * i + 0x40 * x + 0x4A28$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].IRQ_EN

Interface: MCS[i]

Name	FIFO[i]_CH[x]_IRQ_EN
Address	$0x40 * x + 0x4A28$
C-Name	

FIFO_EMPTY_IRQ_EN	
Description	Interrupt enable
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP. 1 : Enable interrupt, interrupt is visible outside GTM-IP.

FIFO_FULL_IRQ_EN	
Description	Interrupt enable
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP. 1 : Enable interrupt, interrupt is visible outside GTM-IP.

FIFO_LWM_IRQ_EN	
Description	Interrupt enable
Loop	-
Bit Range	$[2 : 2]$
Access Type	RW

FIFO_LWM_IRQ_EN	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable interrupt, interrupt is not visible outside GTM–IP. 1 : Enable interrupt, interrupt is visible outside GTM–IP.

FIFO_UWM_IRQ_EN	
Description	Interrupt enable
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable interrupt, interrupt is not visible outside GTM–IP. 1 : Enable interrupt, interrupt is visible outside GTM–IP.

7.3.12 FIFO[i]_CH[x]_IRQ_FORCINT

Description	FIFO[i] channel [x] force interrupt register
Loop	$i = \{n : 0 <= n <= \mathrm{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{PSM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_IRQ_FORCINT
Address	$0x20000 * i + 0x40 * x + 0x4A2C$
C–Name	GTM.CLS[i].PSM.FIFO.CH[x].IRQ_FORCINT

Interface: MCS[i]

Name	FIFO[i]_CH[x]_IRQ_FORCINT
Address	$0x40 * x + 0x4A2C$

C-Name	

TRG_FIFO_EMPTY	
Description	Trigger the bit XXX by software FIFO[i]_CH[x]_IRQ_NOTIFY.FIFO_EMPTY by software
Loop	-
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 3, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **FIFO[i]_CH[x]_IRQ_NOTIFY.FIFO_EMPTY**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

TRG_FIFO_FULL	
Description	Trigger the bit XXX by software FIFO[i]_CH[x]_IRQ_NOTIFY.FIFO_FULL by software
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 3, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **FIFO[i]_CH[x]_IRQ_NOTIFY.FIFO_FULL**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

TRG_FIFO_LWM	
Description	Trigger the bit XXX by software FIFO[i]_CH[x]_IRQ_NOTIFY.FIFO_LWM by software
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 3, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **FIFO[i]_CH[x]_IRQ_NOTIFY.FIFO_LWM**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

TRG_FIFO_UWM	
Description	Trigger the bit XXX by software FIFO[i]_CH[x]_IRQ_NOTIFY.FIFO_UWM by software
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 3, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **FIFO[i]_CH[x]_IRQ_NOTIFY.FIFO_UWM**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

7.3.13 FIFO[i]_CH[x]_IRQ_MODE

Description	FIFO[i] channel [x] interrupt mode control register

Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	PSM[i]_CLK_ENABLE == 1

Interface: CPU

Name	FIFO[i]_CH[x]_IRQ_MODE
Address	$0x20000 * i + 0x40 * x + 0x4A30$
C-Name	GTM.CLS[i].PSM.FIFO.CH[x].IRQ_MODE

Interface: MCS[i]

Name	FIFO[i]_CH[x]_IRQ_MODE
Address	$0x40 * x + 0x4A30$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	-
Bit Range	[1 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

DMA_HYSTERESIS	
Description	Enable DMA hysteresis mode
Loop	-
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

DMA_HYSTERESIS	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable FIFO hysteresis for DMA access. 1 : Enable FIFO hysteresis for DMA access.

DMA_HYST_DIR	
Description	DMA direction in hysteresis mode
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : DMA direction read in hysteresis mode. 1 : DMA direction write in hysteresis mode.
	Note: In the case of DMA writing data to a FIFO the DMA requests must be generated by the lower watermark. If the DMA hysteresis is enabled, the FIFO does not generate a new DMA request until the upper watermark is reached. **Note:** In the case of DMA reading data from FIFO, the DMA requests must be generated by the upper watermark. If the DMA hysteresis is enabled, the FIFO does not generate a new DMA request until the lower watermark is reached.

7.3.14 FIFO[i]_CH[x]_EIRQ_EN

Description	FIFO[i] channel [x] error interrupt enable register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	FIFO[i]_CH[x]_EIRQ_EN
Address	$0x20000 * i + 0x40 * x + 0x4A34$
C–Name	GTM.CLS[i].PSM.FIFO.CH[x].EIRQ_EN

Interface: MCS[i]

Name	FIFO[i]_CH[x]_EIRQ_EN
Address	$0x40 * x + 0x4A34$
C-Name	

FIFO_EMPTY_EIRQ_EN	
Description	Error interrupt enable
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM-IP. 1 : Enable error interrupt, error interrupt is visible outside GTM-IP.

FIFO_FULL_EIRQ_EN	
Description	Interrupt enable
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM-IP. 1 : Enable error interrupt, error interrupt is visible outside GTM-IP.

FIFO_LWM_EIRQ_EN	
Description	Interrupt enable
Loop	-
Bit Range	$[2 : 2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

FIFO_LWM_EIRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP. 1 : Enable error interrupt, error interrupt is visible outside GTM–IP.

FIFO_UWM_EIRQ_EN	
Description	Interrupt enable
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP. 1 : Enable error interrupt, error interrupt is visible outside GTM–IP.

7.3.15 FIFO[i]_MEMORY

Description	FIFO data memory
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}$
Condition	
Size	$4 * 1024$
Clock Active Cond	PSM[i]_CLK_ENABLE == 1

Interface: CPU

Name	FIFO[i]_MEMORY
Address	$0x20000 * i + 0x6000$
C–Name	GTM.CLS[i].FIFO_MEMORY

Interface: MCS[i]

Name	FIFO[i]_MEMORY
Address	$0x6000$
C–Name	

DATA	
Description	FIFO memory location

DATA	
Loop	–
Bit Range	$[28 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	FIFO[i]_CH[0]_CTRL.WULOCK == 0
Reset group	–

7.4 FIFO Signal Description

7.4.1 FIFO Interrupt Signals

Loop	–
Condition	–
Format	–

FIFO_EMPTY	
Description	Indicating empty FIFO[i] x was reached
Loop	–
Condition	–
Signal Type	–
Assignment	–

FIFO_FULL	
Description	Indicating full FIFO[i] x was reached
Loop	–
Condition	–
Signal Type	–
Assignment	–

FIFO_LOWER_WM	
Description	Indicating FIFO[i] x reached lower watermark.
Loop	–
Condition	–
Signal Type	–
Assignment	–

FIFO_UPPER_WM	
Description	Indicating FIFO[i] x reached upper watermark.
Loop	–
Condition	–
Signal Type	–
Assignment	–

8 AEI to FIFO Data Interface (AFD)

8.1 Overview

The AFD submodule implements a data interface between the AEI bus and the FIFO submodule, which consists of eight logical FIFO channels.

Register name indices used within this module:

- i:={0, 1,..., (NSPM−1)} index of module / cluster
- x:={0, 1,..., 7} index of FIFO channel

The AFD submodule provides one buffer register that is dedicated to the logical channels of the FIFO. Access to the corresponding FIFO channel is given by reading or writing this buffer register **AFD[i]_CH[x]_BUF_ACC** .

An AEI write access to the buffer register, where the corresponding FIFO channel is full, will be ignored. The data will be lost.

An AEI read access to the buffer register, where the corresponding FIFO channel is empty, will be served with zero data.

8.2 AFD Configuration Register Description

8.2.1 AFD[i]_CH[x]_BUF_ACC

Description	AFD [i] FIFO [x] buffer access register
Loop	$i = \{n : 0 <= n <= \mathrm{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{PSM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	AFD[i]_CH[x]_BUF_ACC
Address	$0x20000 * i + 0x10 * x + 0x4880$
C−Name	GTM.CLS[i].PSM.AFD.CH[x].BUF_ACC

Interface: MCS[i]

Name	AFD[i]_CH[x]_BUF_ACC
Address	$0x10 * x + 0x4880$
C−Name	

DATA	
Description	Read/write data from/to FIFO
Loop	–
Bit Range	$[28 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

DATA	
Reset group	HW_RESET: async GTM_RESET: async

9 FIFO to ARU Unit (F2A)

9.1 Overview

The F2A is the interface between the ARU and the FIFO sub-module. Since the data width of the ARU (ARU word) is 53-bit (two 24-bit values and five control bits) and the data width of the FIFO is only 29-bit, the F2A has to distribute the data from and to the FIFO channels in a configurable manner.

The data transfer between FIFO and ARU is organized with eight different streams that are connected to the eight different channels of the corresponding FIFO module. A stream represents a data flow from/to ARU to/from the FIFO via the F2A.

The general definition of 'channels' and 'streams' in the context of ARU is given in section *3.9 "ARU Routing Concept"* .

Each FIFO channel can act as a write stream (data flow from FIFO to ARU) or as a read stream (data flow from ARU to FIFO).

Within these streams the F2A can transmit/receive the lower, the upper or both 24-bit values of the ARU together with the ARU control bits according to the configured transfer modes as described in section *9.2 "Transfer modes"*

Each stream can be enabled/disabled separately within the register **F2A[i]_ENABLE** . If a stream will be disabled, the stream data which are stored inside the F2A will be deleted. This is necessary to ensure, that no old data are transferred after enabling a stream.

Indices as used in this chapter are:

- i:={0, 1,..., (NPSM-1)} instance index of cluster / module
- x:={0, 1,..., 7} index of logical F2A channels
- y:={0, 1,..., 7} index of FIFO channel

9.2 Transfer modes

The F2A unit provides several transfer modes to map 29-bit data of the FIFO from/to 53-bit data of the ARU. E.g. it is configurable that the 24 bit FIFO data is written to the lower ARU data entry (means bits [23:0]) or to the higher 24-bit ARU data entry (means bits [47:24]). Bits [28:24] of the FIFO data entry (the five control bits) are written/read in both cases to/from bits [52:48] of the ARU entry.

When both values of the ARU have to be stored in the FIFO the values are stored behind each other inside the FIFO if the FIFO is not full.

If there is space only for one 24-bit data word plus the five control bits, the F2A transfers one part of the 53 bits first and then waits for transferring the second part before new data is requested from the ARU.

When two values from the FIFO have to be written to one ARU location the words have to be located behind each other inside the FIFO.

The transfer to ARU is only established when both parts could be read out of the FIFO otherwise if only one 29-bit word was provided by the FIFO the F2A waits until the second part is available before the data is made available at the ARU.

Figure *34 "Data transfer of both ARU words between ARU and FIFO"* shows the data ordering of the FIFO when both ARU values must be transferred between ARU and FIFO.

When reading from the ARU the F2A first writes the lower word to the FIFO.

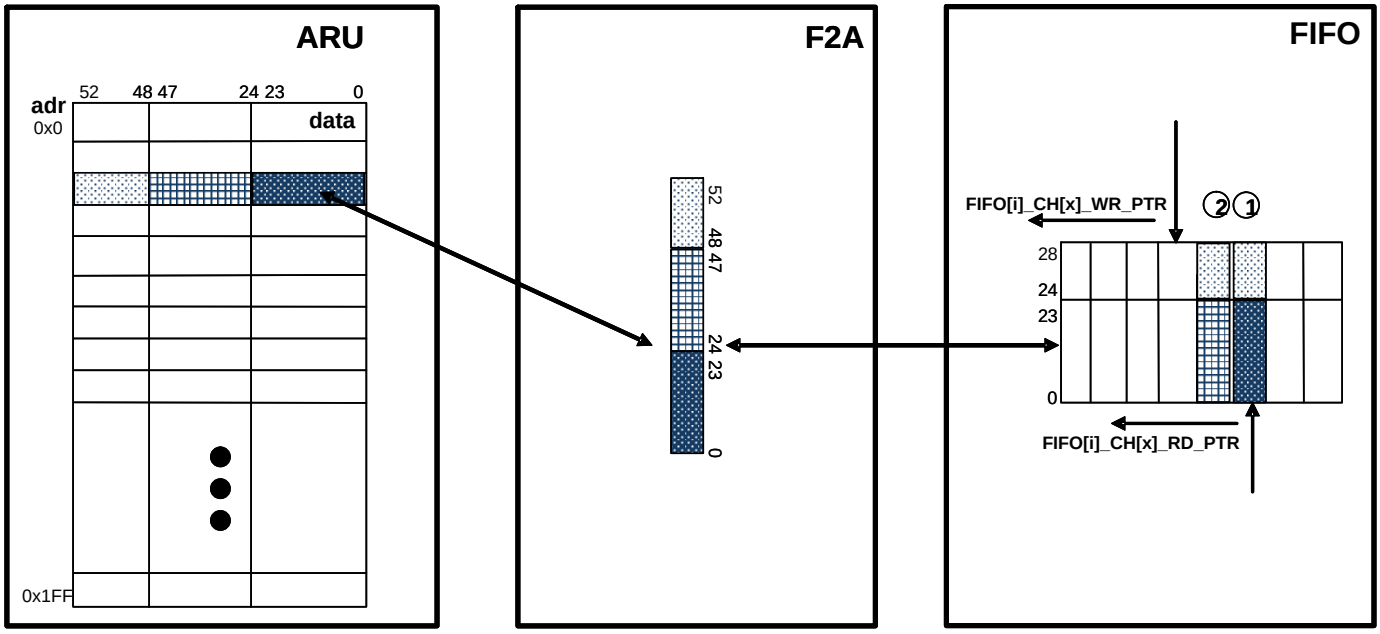
In case of writing to the ARU, the F2A reads the lower word first from the FIFO, thus the lower word must be written first to the FIFO through the AFD interface.

The five control bits (bits [52:48] of the ARU data word) are duplicated as bit [28:24] of both FIFO words in case of reading from ARU.

In the case of writing to the ARU, bits [28:24] of the last written FIFO word (the higher ARU word) are copied to bits [52:48] of the corresponding ARU location.

The transfer modes can be configured with the **F2A[i]_CH[x]_STR_CFG.TMODE** bits.

Figure 34 Data transfer of both ARU words between ARU and FIFO

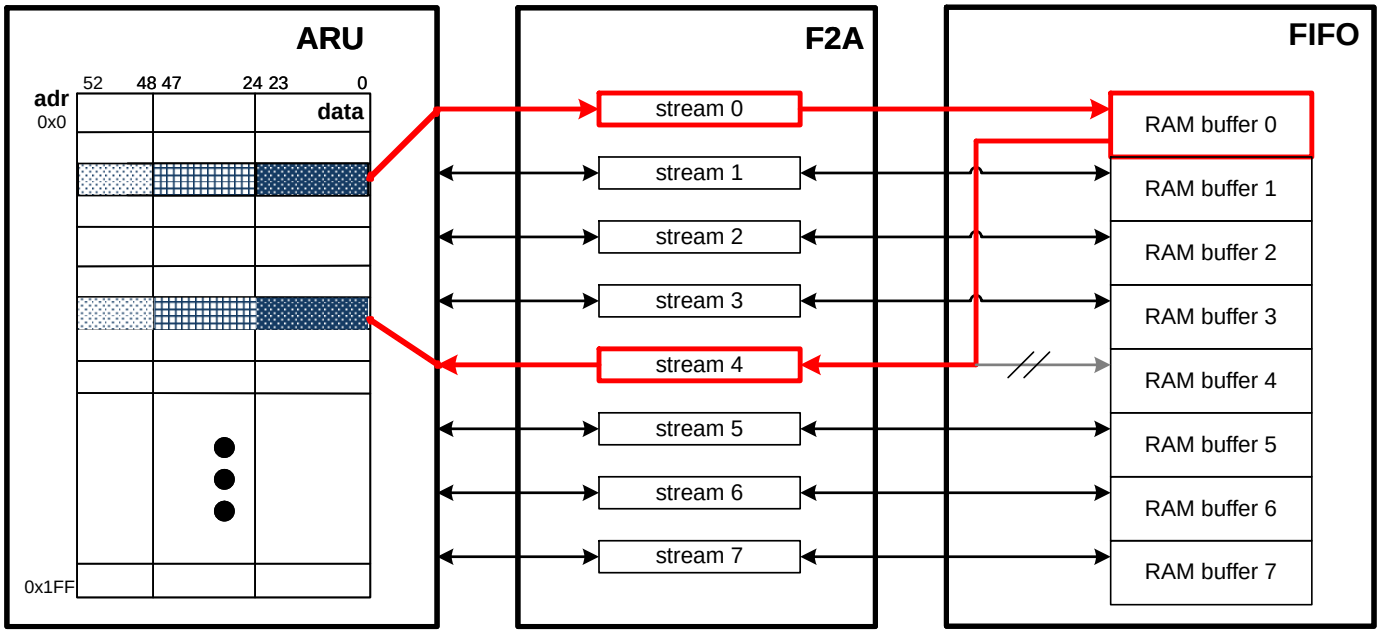


9.3 Internal buffer mode

It is possible to use a FIFO channel as a buffer which is accessed only by ARU internally. To do this, a read and a write stream of the F2A to one FIFO channel is needed. Therefore, it is possible to reconfigure the upper 4 F2A stream 4 to stream 7 to the lower 4 FIFO channels 0 to channel 3 in the following manner:

- F2A stream 4 (+ F2A stream 0) –> FIFO channel 0
- F2A stream 5 (+ F2A stream 1) –> FIFO channel 1
- F2A stream 6 (+ F2A stream 2) –> FIFO channel 2
- F2A stream 7 (+ F2A stream 3) –> FIFO channel 3

Figure 35 Reconfiguration of F2A stream 4 to FIFO channel 0



The configuration of each of the 4 upper F2A streams can be done separately for each stream in the configuration register **F2A[i]_CTRL** .

The corresponding upper FIFO channel 4 to channel 7 cannot be used in this configuration.

9.4 F2A Configuration Registers Description

9.4.1 F2A[i]_ENABLE

Description	F2A[i] stream activation register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	F2A[i]_ENABLE
Address	$0x20000 * i + 0x4840$
C-Name	GTM.CLS[i].PSM.F2A.ENABLE

Interface: MCS[i]

Name	F2A[i]_ENABLE
Address	$0x4840$
C-Name	

STR[y]_EN	
Description	Enable/disable stream [y]
Loop	$y = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * y + 1 : 2 * y]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Don't care, bits will not be changed 0b01 : Stream [y] is disabled and internal states are reset 0b10 : Stream [y] is enabled 0b11 : Don't care, bits will not be changed
R-Coding	0b00 : Stream [y] disabled 0b01 : unused 0b10 : unused 0b11 : Stream [y] enabled
	Note: Stream data inside F2A will be deleted on stream disabling.

9.4.2 F2A[i]_CH[x]_ARU_RD_FIFO

Description	F2A[i] stream [x] ARU read register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	F2A[i]_CH[x]_ARU_RD_FIFO
Address	$0x20000 * i + 0x4 * x + 0x4800$
C-Name	GTM.CLS[i].PSM.F2A.CH_ARU_RD_FIFO[x]

Interface: MCS[i]

Name	F2A[i]_CH[x]_ARU_RD_FIFO
Address	$0x4 * x + 0x4800$
C-Name	

ADDR	
Description	ARU Read address
Loop	-
Bit Range	$[8 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x1FE$
Protect Enable Cond	F2A[i]_ENABLE.STR[x]_EN
Reset group	HW_RESET: async GTM_RESET: async
	Note: This bit field is only writable if channel is disabled.

9.4.3 F2A[i]_CH[x]_STR_CFG

Description	F2A[i] stream [x] configuration register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	F2A[i]_CH[x]_STR_CFG
Address	$0x20000 * i + 0x4 * x + 0x4820$
C-Name	GTM.CLS[i].PSM.F2A.CH_STR_CFG[x]

Interface: MCS[i]

Name	F2A[i]_CH[x]_STR_CFG
Address	$0x4 * x + 0x4820$
C-Name	

TMODE	
Description	Transfer mode for 53 bit ARU data from/to FIFO
Loop	-
Bit Range	$[17 : 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	F2A[i]_ENABLE.STR[x]_EN
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Transfer low word (ARU bits [23:0]) from/to FIFO 0b01 : Transfer high word (ARU bits [47:24]) from/to FIFO 0b10 : Transfer both words from/to FIFO

DIR	
Description	Data transfer direction
Loop	-
Bit Range	$[18 : 18]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	F2A[i]_ENABLE.STR[x]_EN
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Transport from ARU to FIFO 1 : Transport from FIFO to ARU

Note:

The write protected bits of register **F2A[i]_CH[x]_STR_CFG** are only writable if the corresponding enable register bit **F2A[i]_ENABLE.STR[y]_EN** is cleared.

9.4.4 F2A[i]_CTRL

Description	F2A[i] stream control register
Loop	$i = \{n : 0 <= n <= \text{NPSM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{PSM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	F2A[i]_CTRL
Address	$0x20000 * i + 0x4844$
C-Name	GTM.CLS[i].PSM.F2A.CTRL

Interface: MCS[i]

Name	F2A[i]_CTRL
Address	$0x4844$
C-Name	

STR[y]_CONF	
Description	Reconfiguration of stream [y] to FIFO channel [y]-4
Loop	$y = \{n : 4 <= n <= 7\}$
Bit Range	$[2 * (y - 4) + 1 : 2 * (y - 4)]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	F2A[i]_ENABLE.STR[y]_EN || F2A[i]_ENABLE.STR[y - 4]_EN
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Don't care, bits will not be changed 0b01 : Stream [y] is mapped to FIFO buffer [y] 0b10 : Stream [y] is mapped to FIFO buffer [y]-4 0b11 : Don't care, bits will not be changed
R-Coding	0b00 : Stream [y] is mapped to FIFO buffer [y] 0b01 : unused 0b10 : unused 0b11 : Stream [y] is mapped to FIFO buffer [y]-4
	Note: The write protected bits of register **F2A[i]_CTRL** are only writable if the corresponding enable bits **F2A[i]_ENABLE.STR[y-4]_EN** and **F2A[i]_ENABLE.STR[y]_EN** are cleared.

10 Clock Management Unit (CMU)

10.1 Overview

The Clock Management Unit (CMU) only exists in cluster 0 and is tasked with the generation of 14 clock resolution signals and 3 GTM external clocks. The clock resolution signals are routed to all clusters via ICPAs. Within a cluster, they are used as clock enables regulating the resolution (or rate) at which actions are performed. The 3 GTM external clocks are approximately true 50% duty cycle clocks. The primary clock source used in this module is the cluster 0 clock signal *CLS[0]_CLK* . The cluster clock (*CLS[0]_CLK*) can be turned off, or use the GTM clock, or use the GTM clock halved depending on the configuration of **GTM_CLS_CLK_CFG.CLS[0]_CLK_DIV** . The clock resolution signals and the external clocks are generated with reference to it. Figure *36 "CMU Block Diagram "* shows a block diagram of the CMU.

Indices and their range as used inside this chapter are:

- x:={0, 1,..., 7} index for clock resolution channel *CMU_CLK_EN* [x:x]
- y:={0, 1,..., 4} index for fixed clock for *CMU_FXCLK_RES* [y:y]
- z:={0, 1, 2} index for external clocks *CMU_ECLK* [z:z]

The Configurable Resolution Generation (CRG) subunit generates eight dedicated clock resolutions (*CMU_CLK_RES* [x:x], where x ∈ {0, 1, ..., 7}) for the following GTM modules: TIM, ATOM, TBU, and CDTM, TIO, and MON. Every resolution signal has its own configuration register (**CMU_CLK_[x]_CTRL**) to configure its clock resolution generator. The input to a clock resolution generator can either be *CMU_ECLK1_EN* or *CMU_GCLK_EN* depending on the configuration of **CMU_CLK_CTRL.CLK[x]_EXT_DIVIDER** .

The Fixed Clock Resolution Generation (FCRG) subunit generates predefined non-configurable clock resolutions *CMU_FXCLK_RES* [y:y] (y ∈ {0, 1, ..., 4}) for the TOM, CDTM, and MON modules. In the subunit there is a multiplexer that can be configured to select *CMU_GCLK_EN* or one of the eight *CMU_CLK_RES* [x:x] (x ∈ {0, 1, ..., 7}) as input to the FCR generators within. The dividing factors for a *CMU_FXCLK_RES* [y:y] clock resolution is non-configurable and is fixed to 2^{4*y} .

The External Clock Generation (EGU) subunit is able to generate up to three chip external clock signals visible at *CMU_ECLK* [z:z] (z ∈ {0, 1, 2}) with a duty cycle of approximately 50%.

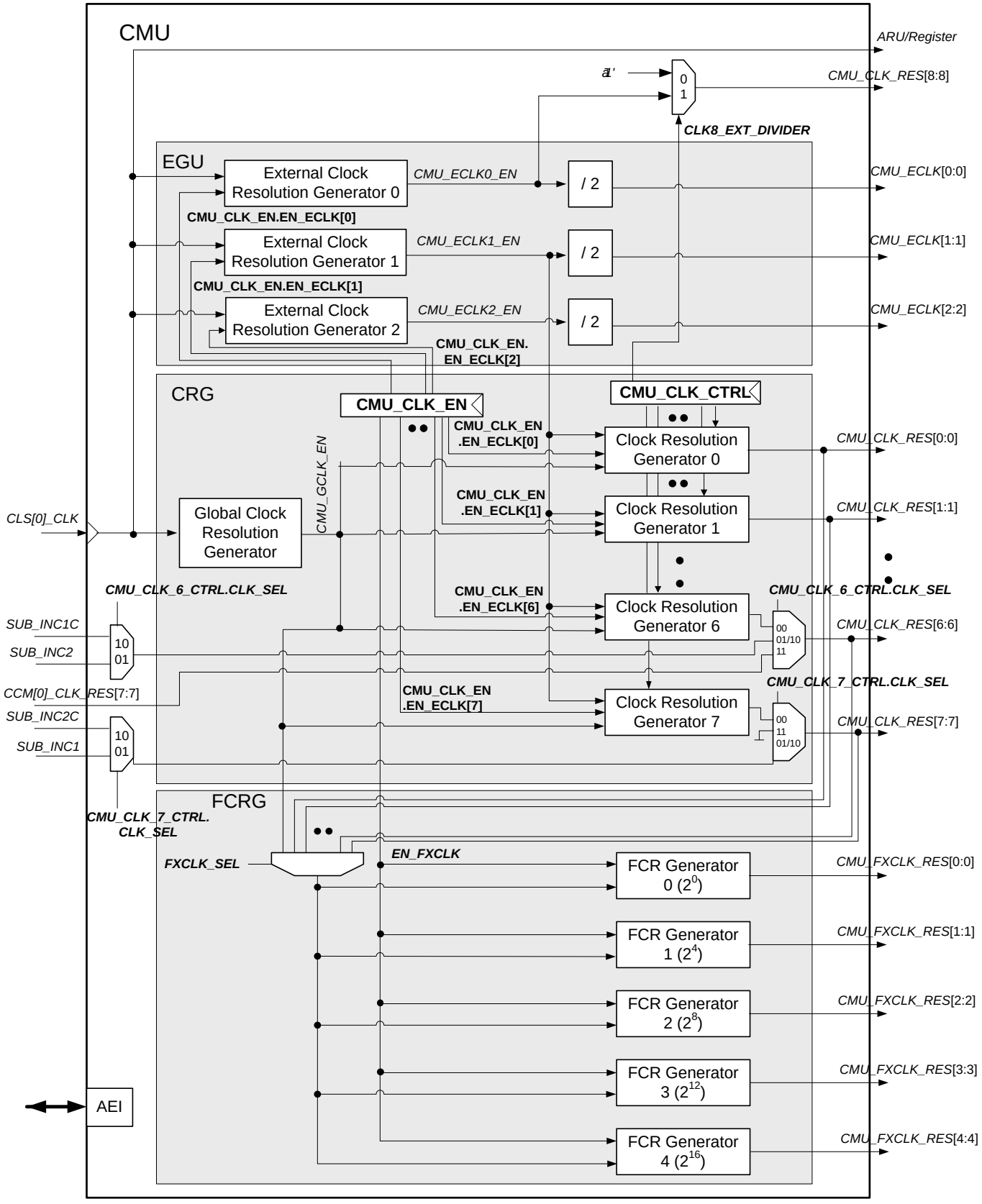
The External Clock Generation (EGU) subunit is able to generate clock *CMU_CLK_RES* [8:8] for module CCM to manage 2 clock domains.

The clock source signals *CMU_CLK_RES* [x:x] (x∈{0, 1,..., 7}) and *CMU_FXCLK_RES* [y:y] are implemented in the form of enable signals for the corresponding registers, which means that the actual clock signal of all registers always use *CLS[0]_CLK* signal.

The four configurable clock resolution signals *CMU_CLK_RES* [0:0], *CMU_CLK_RES* [1:1], *CMU_CLK_RES* [6:6] and *CMU_CLK_RES* [7:7] are used for the TIM filter counters.

Although the clock resolutions and external clocks (*CMU_CLK_EN* , *CMU_FXCLK_RES* , *CMU_ECLK*) are run-time configurable, any change during run-time may result in an unpredictable behavior in the GTM-IP. It is strongly recommended to configure them only during initialization phase.

Figure 36 CMU Block Diagram



10.2 Global Clock Resolution Generator

The subblock Global Clock Resolution Generator can be used to generate a common clock resolution signal (*CMU_GCLK_EN*) with reference to cluster 0 clock (*CLS[0]_CLK*).

The common clock resolution signal is generated by the Global Clock Resolution Generator. Depending on the configuration of the numerator **CMU_GCLK_NUM.GCLK_NUM** and the denominator **CMU_GCLK_DEN.GCLK_DEN** , the Global Clock Resolution Generator attempts to assert *CMU_GCLK_EN* for **CMU_GCLK_DEN.GCLK_DEN** clock cycles distributed at regular intervals (when possible) in every **CMU_GCLK_NUM.GCLK_NUM** clock cycles. Depending on the configuration, the ratio between the numerator to the denominator might result in a fractional number leading to the introduction of some irregularities in these intervals.

In theory, if the common clock resolution signal is used as a clock enable in conjunction with the cluster clock, it is possible to regulate the effective clock frequency (denoted as 1/T *CMU_GCLK_EN*) at which the registers are clocked. The effective clock frequency can range between 0% (when *CMU_GCLK_EN* is low all the time) and 100% (when *CMU_GCLK_EN* is high all the time) of the cluster clock frequency depending on the configuration. It is important to emphasize that T *CMU_GCLK_EN* is only a period of a virtual clock whose frequency is equivalent to the effective clock frequency resulting from using *CMU_GCLK_EN* as a clock enable to the registers the cluster clock is connected to. The following algorithm describes how to achieve a particular T *CMU_GCLK_EN* = (Z/N) * T *CLS[0]_CLK* , where Z = **CMU_GCLK_NUM.GCLK_NUM** ; N = **CMU_GCLK_DEN.GCLK_DEN** ; Z>= N > 0:

1. INIT−phase: set remainder (R), operand1 (OP1) and operand2 (OP2) register during INIT−phase (with implicit conversion to signed): R=Z, OP1=N, O−P2=N−Z.
2. After leaving INIT−phase (at least one *CMU_CLK_RES* [x:x] has been enabled) the sign of remainder R for each *CLS[0]_CLK* cycle is checked:
3. If R>=0: keep *CMU_GCLK_EN* ='0', and update R=R−OP1.
4. If R<0: set *CMU_GCLK_EN* ='1', and update R=R−OP2.

After at most (Z/N+1) subtractions (3. in the list above) there will be a negative R and an active phase of the generated clock enable (for one cycle) will be triggered (4. in the list above). The remainder R is a measure for the distance to a real Z/N clock and will be considered for the next generated clock enable cycle phase. The new R value will be R=R+(Z−N). In the worst case, the remainder R will sum up to an additional cycle in the generated clock enable period after Z−cycles. In other cases, equally distributed additional cycles will be inserted for the generated clock enable. If Z is an integer multiple of N, no additional cycles will be included for the generated clock enable at all.

Note:

For a better resource sharing all arithmetic has been reduced to subtractions and the initialization of the remainder R uses the complement of (Z−N).

10.3 Configurable clock Resolution Generation (CRG)

The CRG subunit generates up to eight configurable clock resolution signals with reference to an input signal. The input signal can either be the common clock resolution signal *CMU_GCLK_EN* or *CMU_ECLK1_EN* depending on the configuration of **CMU_CLK_CTRL.CLK[x]_EXT_DIVIDER** .

Each clock resolution signal *CMU_CLK_RES* [x:x] has its own configuration register **CMU_CLK_[x]_CTRL** .

The value specified in **CMU_CLK_[x]_CTRL.CLK_CNT** determines how many pulses (of one *CLS[0]_CLK* clock cycle duration) from the input signal are suppressed before one is wired to *CMU_CLK_RES* [x:x].

When a *CMU_CLK_RES* [x:x] is used as a clock enable in conjunction with cluster clock, it is possible to regulate the effective clock frequency (denoted as f *CMU_CLK_RES* [x:x] = 1/T *CMU_CLK_RES* [x:x]) at which the registers are clocked. It is important to emphasize that T *CMU_CLK_RES* [x:x] is only a period of a virtual clock whose frequency is equivalent to the effective clock frequency resulting from using *CMU_CLK_RES* [x:x] as a clock enable to the registers the cluster clock is connected to. From that, T *CMU_CLK_RES* [x:x] can be expressed as: T *CMU_CLK_RES* [x:x] = (**CMU_CLK_[x]_CTRL.CLK_CNT** +1) * T *CMU_GCLK_EN* . Depending on the configuration of **CMU_CLK_CTRL.CLK[x]_EXT_DIVIDER** , T *CMU_GCLK_EN*
can be substituted with T *CMU_ECLK1_EN* . The corresponding waveform is shown in Figure *37 "Wave Form of Generated Clock Signal"*

Each clock resolution signal *CMU_CLK_RES* [z:z] (z ∈ {0, 1, ..., 5} can be enabled individually by appropriately configuring the corresponding **CMU_CLK_EN.EN_CLK[z]** bit field. Enabling of *CMU_CLK_RES* [6:6] is only possible if the bit field **CMU_CLK_EN.EN_CLK[6]** is appropriately configured and the bit field **CMU_CLK_6_CTRL.CLK_SEL** is set to 0. Similarly, enabling of *CMU_CLK_RES* [7:7] is only possible if the bit field **CMU_CLK_EN.EN_CLK[7]** is appropriately configured and the bit field **CMU_CLK_7_CTRL.CLK_SEL** is set to 0.

Clock resolutions *CMU_CLK_RES* [6:6] and *CMU_CLK_RES* [7:7] may instead be configured to use the signal *SUB_INC1* and *SUB_INC2* coming from module DPLL as clock enables depending on the configuration bit fields **CMU_CLK_6_CTRL.CLK_SEL** and **CMU_CLK_7_CTRL.CLK_SEL** , respectively.

The clock resolution *CMU_CLK_RES* [8:8] can be configured by **CMU_CLK_CTRL.CLK8_EXT_DIVIDER** to select between a constant '1' and *CMU_ECLK0_EN* .

The input to a clock source divider can either be *CMU_ECLK1_EN* or *CMU_GCLK_EN* depending on the configuration of **CMU_CLK_CTRL.CLK[x]_EXT_DIVIDER** .

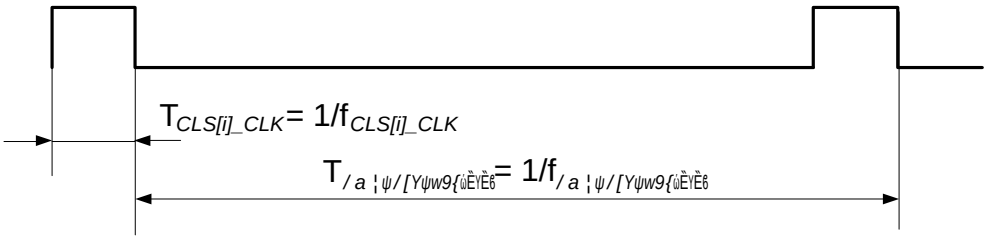
To avoid unexpected behavior of the hardware, the configuration of register **CMU_CLK_[x]_CTRL** and **CMU_CLK_CTRL** can only be changed, when the corresponding clock resolution signal *CMU_CLK_RES* [x:x] and *CMU_ECLK* [1:1] are disabled.

Further, any changes to the registers **CMU_GCLK_NUM** and **CMU_GCLK_DEN** can only be performed, when all clock resolution signals (*CMU_CLK_RES* [x:x] and *CMU_FXCLK_RES* [y:y]) are disabled.

The clock resolution signals *CMU_CLK_RES* [x:x] and *CMU_FXCLK_RES* [y:y] are implemented in the form of enable signals for the corresponding registers, which means that the actual clock signal of all registers always use the *CLS[i]_CLK* signal.

The hardware guarantees that all clock resolution signals *CMU_CLK_RES* [x:x], which are enabled simultaneously, remain synchronous to each other. Synchronous enabling here means that the bit fields **CMU_CLK_EN.EN_CLK[x]** in the register **CMU_CLK_EN** are set by the same write access.

Figure 37 Wave Form of Generated Clock Signal



10.4 Fixed Clock Resolution Generation (FCRG)

The FCRG subunit generates fixed clock resolutions signals based on the *CMU_GCLK_EN* signal or one of the eight *CMU_CLK_RES* [x:x] resolution signals. The selection of which signal is configurable via **CMU_FXCLK_CTRL.FXCLK_SEL** bit field. These clock resolutions are used for the PWM generation inside the TOM modules.

All clock resolutions *CMU_FXCLK_RES* [y:y] can be enabled/disabled simultaneously by appropriately configuring bit field **CMU_CLK_EN.-EN_FXCLK** .

The dividing factors are defined as 2^0, 2^4, 2^8, 2^{12}, and 2^{16}. An FCR Generator with a dividing factor configured as 2^q shall only wire one pulse (of one *CLS[0]_CLK* clock cycle) from every 2^q pulses to the output. The remaining pulses (2^q-1) are suppressed.

10.5 External Generation Unit (EGU)

The EGU subunit generates up to three separate clock output signals *CMU_ECLK* [z:z].

Each of these clock signals is derived from the corresponding External Clock Resolution Generator subblock, which generates a clock signal derived from the GTM-IP input clock *CLS[0]_CLK* .

In contrast to the signals *CMU_CLK_RES* [x:x] and *CMU_FXCLK_RES* [y:y], which are clock resolution signals effectively regulating the clock frequency when used as clock enables in conjunction with the cluster clock, the signals *CMU_ECLK* [z:z] have a duty cycle of approximately 50%. A *CMU_CLK_RES* [x:x] can be used as a true clock signal for external peripheral components.

The signal *CMU_ECLK1_EN* could be selected (instead of *CMU_GCLK_EN*) as input for the clock resolution generators to be used as a reference in the generation of the *CMU_CLK_RES* [x:x] signals by appropriately configuring **CMU_CLK_CTRL.CLK[x]_EXT_DIVIDER** . In addition, *CMU_ECLK0_EN* could be wired (instead of '1') to the all-time enabled *CMU_CLK_RES* [8:8] by appropriately configuring **CMU_CLK_CTRL.CLK8_EXT_DIVIDER** .

Each of the external clock dividers is enabled/disabled by appropriately configuring bit field **CMU_CLK_EN.EN_ECLK[z]** .

Similar to 'Global Clock Resolution Generator', an 'External Clock Resolution Generator' generates a clock resolution signal (let *CMU_ECLK_-EN* $\in$ { *CMU_ECLK0_EN* , *CMU_ECLK1_EN* , *CMU_ECLK2_EN* }). In theory, if a clock resolution signal *CMU_ECLK_EN* is used as a clock enable in conjunction with the cluster clock, it is possible to regulate the effective clock frequency (denoted as 1/T *CMU_ECLK_EN*) at which the registers are clocked. It is important to emphasize that T *CMU_ECLK_EN* is only a period of a virtual clock whose frequency is equivalent to the effective clock frequency resulting from using *CMU_ECLK_EN* as a clock enable to the registers the cluster clock is connected to. The following algorithm describes how to achieve a particular T *CMU_ECLK_EN* and the corresponding *CMU_ECLK* [z:z] by appropriately configuring **CMU_ECLK_[z]_NUM** and **CMU_ECLK_[z]_DEN** . Let:

$T_{CMU\_ECLK0\_EN}$ =(**CMU_ECLK_[0]_NUM** / **CMU_ECLK_[0]_DEN**)*$T_{CLS[0]\_CLK}$
$T_{CMU\_ECLK1\_EN}$ =(**CMU_ECLK_[1]_NUM** / **CMU_ECLK_[1]_DEN**)*$T_{CLS[0]\_CLK}$
$T_{CMU\_ECLK2\_EN}$ =(**CMU_ECLK_[2]_NUM** / **CMU_ECLK_[2]_DEN**)*$T_{CLS[0]\_CLK}$

and is implemented according to the following algorithm
(Z: **CMU_ECLK_[z]_NUM.ECLK_NUM** ; N: **CMU_ECLK_[z]_DEN.ECLK_DEN** ; Z>=N >0 ; Z>=N ; *CMU_ECLK* [z:z]='0'):

1. INIT-phase: Set remainder (R), operand1 (OP1) and operand2 (OP2) register during INIT-phase (with implicit conversion to signed): R=Z, OP1=N, OP2=N-Z.
2. After leaving INIT-phase (*CMU_ECLK* [z:z] has been enabled) the sign of remainder R will be checked every (*CLS[0]_CLK* clock cycle):
3. If R>=0: keep the corresponding *CMU_ECLK_EN* at '0', keep *CMU_ECLK* [z:z], and update R=R-OP1.
4. If R<0: set corresponding *CMU_ECLK_EN* to '1', toggle *CMU_ECLK* [z:z], and update R=R-OP2.

After at most (Z/N+1) subtractions (3. in the list above), there will be a negative R and an active phase of the generated clock enable (for one cycle) will be triggered (4. in the list above). The remainder R is a measure for the distance to a real Z/N clock and will be considered for the next generated clock toggle phase. The new R value will be R=R+(Z-N). In the worst case, the remainder R will sum up to an additional cycle in the generated clock toggle period after Z-cycles. In other cases equally distributed additional cycles will be inserted for the generated clock toggle. If Z is an integer multiple of N, no additional cycles will be included for the generated clock toggle at all.

Attention: an external clock signal (*CMU_ECLK* [z:z]) shall have exactly 50% duty cycle only when the ratio **CMU_ECLK_[z]_NUM.ECLK_NUM** / **CMU_ECLK_[z]_DEN.ECLK_DEN** results in an integer value.

Note:

For a better resource sharing, all arithmetic has been reduced to subtractions and the initialization of the remainder R uses the complement of (Z−N).

The initial value of the *CMU_ECLK* [z:z] output is low.

10.6 CMU Configuration Registers Description

10.6.1 CMU_CLK_EN

Description	CMU clock enable
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_CLK_EN
Address	$0x80$
C−Name	GTM.CLS[0].CMU.CLK_EN

Interface: MCS[i]

Name	CMU_CLK_EN
Address	$0x80$
C−Name	

EN_CLK[x]	
Description	Enable clock resolution CMU_CLK_RES[x:x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * x + 1 : 2 * x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	−
Initial value	0
Protect Enable Cond	−
Reset group	HW_RESET: async GTM_RESET: async
W−Coding	0b00 : Don't care, bits will not be changed 0b01 : Disable clock signal and reinitialize internal states 0b10 : Enable clock signal 0b11 : Don't care, bits will not be changed

EN_CLK[x]	
R-Coding	0b00 : Clock is disabled 0b01 : Unused 0b10 : Unused 0b11 : Clock is enabled

EN_ECLK[z]	
Description	Enable clock CMU_ECLK[z:z]
Loop	$z = \{n : 0 <= n <= 2\}$
Bit Range	$[2 * z + 17 : 2 * z + 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Don't care, bits will not be changed 0b01 : Disable clock signal and reinitialize internal states 0b10 : Enable clock signal 0b11 : Don't care, bits will not be changed
R-Coding	0b00 : Clock is disabled 0b01 : Unused 0b10 : Unused 0b11 : Clock is enabled

EN_FXCLK	
Description	Enable clock resolution CMU_FXCLK_RES
Loop	-
Bit Range	$[23 : 22]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	$\mathrm{NTOM} > 0$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Don't care, bits will not be changed 0b01 : Disable clock signal and reinitialize internal states 0b10 : Enable clock signal 0b11 : Don't care, bits will not be changed

EN_FXCLK	
R-Coding	0b00 : Clock is disabled 0b01 : Unused 0b10 : Unused 0b11 : Clock is enabled

10.6.2 CMU_GCLK_NUM

Description	The numerator for CMU global clock resolution generator
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_GCLK_NUM
Address	$0x84$
C-Name	GTM.CLS[0].CMU.GCLK_NUM

Interface: MCS[i]

Name	CMU_GCLK_NUM
Address	$0x84$
C-Name	

GCLK_NUM	
Description	Numerator for global clock resolution generator. Defines numerator of the fractional divider.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	(CMU_CLK_EN.EN_CLK[0] || CMU_CLK_EN.EN_CLK[1] || CMU_CLK_EN.EN_CLK[2] || CMU_CLK_EN.EN_CLK[3] || CMU_CLK_EN.EN_CLK[4] || CMU_CLK_EN.EN_CLK[5] || CMU_CLK_EN.EN_CLK[6] || CMU_CLK_EN.EN_CLK[7] || CMU_CLK_EN.EN_FXCLK)
Reset group	HW_RESET: async GTM_RESET: async
	Note: Value can only be modified when all clock enables **CMU_CLK_EN.EN_CLK[x]** and the **CMU_CLK_EN.EN_FXCLK** are disabled. Protected by (**CMU_CLK_EN.EN_CLK[x]** != 0 or **CMU_CLK_EN.EN_FXCLK** != 0) **Note:** The CMU hardware alters the content of **CMU_GCLK_NUM.GCLK_NUM** and **CMU_GCLK_DEN.GCLK_DEN** automatically to 0x1, if **CMU_GCLK_NUM.GCLK_NUM** is specified less than **CMU_GCLK_DEN.GCLK_DEN** or one of the values is specified with a value zero. Thus, a secure way for altering the values is writing twice to the register **CMU_GCLK_NUM** followed by a single write to register **CMU_GCLK_DEN** .

10.6.3 CMU_GCLK_DEN

Description	The denominator for CMU global clock resolution generator
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_GCLK_DEN
Address	$0x88$
C-Name	GTM.CLS[0].CMU.GCLK_DEN

Interface: MCS[i]

Name	CMU_GCLK_DEN
Address	$0x88$
C-Name	

GCLK_DEN	
Description	Denominator for global resolution generator. Defines denominator of the fractional divider.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	(CMU_CLK_EN.EN_CLK[0] || CMU_CLK_EN.EN_CLK[1] || CMU_CLK_EN.EN_CLK[2] || CMU_CLK_EN.EN_CLK[3] || CMU_CLK_EN.EN_CLK[4] || CMU_CLK_EN.EN_CLK[5] || CMU_CLK_EN.EN_CLK[6] || CMU_CLK_EN.EN_CLK[7] || CMU_CLK_EN.EN_FXCLK)
Reset group	HW_RESET: async GTM_RESET: async
	Note: Value can only be modified when all clock enables **CMU_CLK_EN.EN_CLK[x]** and the **CMU_CLK_EN.EN_FXCLK** are disabled. Protected by (**CMU_CLK_EN.EN_CLK[x]** != 0 or **CMU_CLK_EN.EN_FXCLK** != 0) **Note:** The CMU hardware alters the content of **CMU_GCLK_NUM.GCLK_NUM** and **CMU_GCLK_DEN.GCLK_DEN** automatically to 0x1, if **CMU_GCLK_NUM.GCLK_NUM** is specified less than **CMU_GCLK_DEN.GCLK_DEN** or one of the values is specified with a value zero. Thus, a secure way for altering the values is writing twice to the register **CMU_GCLK_NUM** followed by a single write to register **CMU_GCLK_DEN** .

10.6.4 CMU_CLK_[x]_CTRL

Description	CMU control for clock resolution generator [x]
Loop	$x = \{n : 0 <= n <= 5\}$
Condition	

Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_CLK_0_CTRL
Address	$0x8C$
C-Name	GTM.CLS[0].CMU.CLK_0_CTRL
Name	CMU_CLK_1_CTRL
Address	$0x90$
C-Name	GTM.CLS[0].CMU.CLK_1_CTRL
Name	CMU_CLK_2_CTRL
Address	$0x94$
C-Name	GTM.CLS[0].CMU.CLK_2_CTRL
Name	CMU_CLK_3_CTRL
Address	$0x98$
C-Name	GTM.CLS[0].CMU.CLK_3_CTRL
Name	CMU_CLK_4_CTRL
Address	$0x9C$
C-Name	GTM.CLS[0].CMU.CLK_4_CTRL
Name	CMU_CLK_5_CTRL
Address	$0xA0$
C-Name	GTM.CLS[0].CMU.CLK_5_CTRL

Interface: MCS[i]

Name	CMU_CLK_0_CTRL
Address	$0x8C$
C-Name	
Name	CMU_CLK_1_CTRL
Address	$0x90$
C-Name	
Name	CMU_CLK_2_CTRL
Address	$0x94$
C-Name	
Name	CMU_CLK_3_CTRL
Address	$0x98$
C-Name	
Name	CMU_CLK_4_CTRL
Address	$0x9C$
C-Name	
Name	CMU_CLK_5_CTRL
Address	$0xA0$

C-Name	

CLK_CNT	
Description	Clock count. Defines count value for the clock divider.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	(CMU_CLK_EN.EN_CLK[x] || CMU_CLK_EN.EN_ECLK[1])
Reset group	HW_RESET: async GTM_RESET: async
	Note: Value can only be modified when clock enable **CMU_CLK_EN.EN_CLK[x]** (x∈{0, 1,..., 5}) and **CMU_CLK_EN.-EN_ECLK[1]** are disabled. Protected by (**CMU_CLK_EN.EN_CLK[x]** != 0 or **CMU_CLK_EN.EN_ECLK[1]** != 0)

10.6.5 CMU_CLK_6_CTRL

Description	CMU control for clock resolution generator 6
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_CLK_6_CTRL
Address	$0xA4$
C-Name	GTM.CLS[0].CMU.CLK_6_CTRL

Interface: MCS[i]

Name	CMU_CLK_6_CTRL
Address	$0xA4$
C-Name	

CLK_CNT	
Description	Clock count. Define count value for the clock resolution generator responsible for CMU_CLK_RES[6:6] generation.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False

CLK_CNT	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(CMU_CLK_EN.EN_CLK[6] || CMU_CLK_EN.EN_ECLK[1])
Reset group	HW_RESET: async GTM_RESET: async
	Note: Value can only be modified when clock enable **CMU_CLK_EN.EN_CLK[6]** and **CMU_CLK_EN.EN_ECLK[1]** are disabled. Protected by (**CMU_CLK_EN.EN_CLK[6]** != 0 or **CMU_CLK_EN.EN_ECLK[1]** != 0)

CLK_SEL	
Description	Source selection
Loop	–
Bit Range	[25 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(CMU_CLK_EN.EN_CLK[6] || CMU_CLK_EN.EN_ECLK[1])
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	NDPLL == 1 0b00 : Use output of clock resolution generator 6 0b01 : Use signal *SUB_INC2* of module DPLL 0b10 : Use signal *SUB_INC1C* of module DPLL 0b11 : Use signal *CCM[0]_CLK_RES* [7:7] of sub-module CCM0
RW-Coding	NDPLL == 0 0b00 : Use output of clock resolution generator 6 0b01 : input tied to low, no DPLL present in device 0b10 : input tied to low, no DPLL present in device 0b11 : Use signal *CCM[0]_CLK_RES* [7:7] of sub-module CCM0
	Note: Value can only be modified when clock enable **CMU_CLK_EN.EN_CLK[6]** and **CMU_CLK_EN.EN_ECLK[1]** are disabled. Protected by (**CMU_CLK_EN.EN_CLK[6]** != 0 or **CMU_CLK_EN.EN_ECLK[1]** != 0)

10.6.6 CMU_CLK_7_CTRL

Description	CMU control for clock resolution generator 7
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	CMU_CLK_ENABLE == 1

Interface: CPU

Name	CMU_CLK_7_CTRL
Address	$0xA8$
C-Name	GTM.CLS[0].CMU.CLK_7_CTRL

Interface: MCS[i]

Name	CMU_CLK_7_CTRL
Address	$0xA8$
C-Name	

CLK_CNT	
Description	Clock count. Define count value for the clock resolution generator responsible for CMU_CLK_RES[7:7] generation.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	(CMU_CLK_EN.EN_CLK[7] || CMU_CLK_EN.EN_ECLK[1])
Reset group	HW_RESET: async GTM_RESET: async
	Note: Value can only be modified when clock enable **CMU_CLK_EN.EN_CLK[7]** and **CMU_CLK_EN.EN_ECLK[1]** are disabled. Protected by (**CMU_CLK_EN.EN_CLK[7]** != 0 or **CMU_CLK_EN.EN_ECLK[1]** != 0)

CLK_SEL	
Description	Source selection
Loop	-
Bit Range	$[25:24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	(CMU_CLK_EN.EN_CLK[7] || CMU_CLK_EN.EN_ECLK[1])
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	NDPLL == 1 0b00 : Use output of clock resolution generator 7 0b01 : Use signal *SUB_INC1* of module DPLL 0b10 : Use signal *SUB_INC2C* of module DPLL 0b11 : Reserved, no clock is selected

CLK_SEL	
RW-Coding	NDPLL == 0 0b00 : Use output of clock resolution generator 7 0b01 : input tied to low, no DPLL present in device 0b10 : input tied to low, no DPLL present in device 0b11 : Reserved, input tied to low
	Note: Value can only be modified when clock enable **CMU_CLK_EN.EN_CLK[7]** and **CMU_CLK_EN.EN_ECLK[1]** are disabled. Protected by (**CMU_CLK_EN.EN_CLK[7]** != 0 or **CMU_CLK_EN.EN_ECLK[1]** != 0)

10.6.7 CMU_ECLK_[z]_NUM

Description	The numerator for the external clock resolution generator [z]
Loop	$z = \{n : 0 <= n <= 2\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_ECLK_0_NUM
Address	$0xAC$
C-Name	GTM.CLS[0].CMU.ECLK_0_NUM
Name	CMU_ECLK_1_NUM
Address	$0xB4$
C-Name	GTM.CLS[0].CMU.ECLK_1_NUM
Name	CMU_ECLK_2_NUM
Address	$0xBC$
C-Name	GTM.CLS[0].CMU.ECLK_2_NUM

Interface: MCS[i]

Name	CMU_ECLK_0_NUM
Address	$0xAC$
C-Name	
Name	CMU_ECLK_1_NUM
Address	$0xB4$
C-Name	
Name	CMU_ECLK_2_NUM
Address	$0xBC$
C-Name	

ECLK_NUM	
Description	The numerator for external clock resolution generator. Defines numerator of the fractional divider.
Loop	-

ECLK_NUM	
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	(CMU_CLK_EN.EN_ECLK[z])
Reset group	HW_RESET: async GTM_RESET: async
	Note: Value can only be modified when clock enable **CMU_CLK_EN.EN_ECLK[z]** is disabled. Protected by (**CMU_CLK_EN.EN_ECLK[z]** != 0) **Note:** The CMU hardware alters the content of **CMU_ECLK_[z]_NUM** and **CMU_ECLK_[z]_DEN** automatically to 0x1, if **CMU_ECLK_[z]_NUM** is specified less than **CMU_ECLK_[z]_DEN** or one of the values is specified with a value zero. Thus, a secure way for altering the values is writing twice to the register **CMU_ECLK_[z]_NUM** followed by a single write to register **CMU_ECLK_[z]_DEN** .

10.6.8 CMU_ECLK_[z]_DEN

Description	The denominator for the external clock resolution generator [z]
Loop	$z = \{n : 0 <= n <= 2\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_ECLK_0_DEN
Address	$0xB0$
C-Name	GTM.CLS[0].CMU.ECLK_0_DEN
Name	CMU_ECLK_1_DEN
Address	$0xB8$
C-Name	GTM.CLS[0].CMU.ECLK_1_DEN
Name	CMU_ECLK_2_DEN
Address	$0xC0$
C-Name	GTM.CLS[0].CMU.ECLK_2_DEN

Interface: MCS[i]

Name	CMU_ECLK_0_DEN
Address	$0xB0$
C-Name	
Name	CMU_ECLK_1_DEN

Address	$0xB8$
C-Name	
Name	CMU_ECLK_2_DEN
Address	$0xC0$
C-Name	

ECLK_DEN	
Description	The denominator for external clock resolution generator. Defines denominator of the fractional divider.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	(CMU_CLK_EN.EN_ECLK[z])
Reset group	HW_RESET: async GTM_RESET: async
	Note: Value can only be modified when clock enable **CMU_CLK_EN.EN_ECLK[z]** is disabled. Protected by (**CMU_CLK_EN.EN_ECLK[z]** != 0) **Note:** The CMU hardware alters the content of **CMU_ECLK_[z]_NUM** and **CMU_ECLK_[z]_DEN** automatically to 0x1, if **CMU_ECLK_[z]_NUM** is specified less than **CMU_ECLK_[z]_DEN** or one of the values is specified with a value zero. Thus, a secure way for altering the values is writing twice to the register **CMU_ECLK_[z]_NUM** followed by a single write to register **CMU_ECLK_[z]_DEN** .

10.6.9 CMU_FXCLK_CTRL

Description	CMU control for selection of FCR subblock input
Loop	
Condition	$\mathrm{NTOM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_FXCLK_CTRL
Address	$0xC4$
C-Name	GTM.CLS[0].CMU.FXCLK_CTRL

Interface: MCS[i]

Name	CMU_FXCLK_CTRL
Address	$0xC4$

C-Name	

FXCLK_SEL	
Description	Input selection for EN_FXCLK line
Loop	-
Bit Range	$[3:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	(CMU_CLK_EN.EN_FXCLK)
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b0000 : *CMU_GCLK_EN* selected. 0b0001 : *CMU_CLK_RES* [0:0] selected. 0b0010 : *CMU_CLK_RES* [1:1] selected. 0b0011 : *CMU_CLK_RES* [2:2] selected. 0b0100 : *CMU_CLK_RES* [3:3] selected. 0b0101 : *CMU_CLK_RES* [4:4] selected. 0b0110 : *CMU_CLK_RES* [5:5] selected. 0b0111 : *CMU_CLK_RES* [6:6] selected. 0b1000 : *CMU_CLK_RES* [7:7] selected.
	Note: This value can only be written, when the *CMU_FXCLK_RES* generation is disabled. See bits [23:22] in register **CMU_CLK_EN** . Protected by (**CMU_CLK_EN.EN_FXCLK** != 0) **Note:** Other values for **CMU_FXCLK_CTRL.FXCLK_SEL** are reserved and should not be used, but they behave like **CMU_FXCLK_CTRL.FXCLK_SEL** = 0.

10.6.10 CMU_GLB_CTRL

Description	CMU synchronizing ARU and clock source
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_GLB_CTRL
Address	$0xC8$
C-Name	GTM.CLS[0].CMU.GLB_CTRL

Interface: MCS[i]

Name	CMU_GLB_CTRL
Address	$0xC8$

C-Name	

ARU_ADDR_RSTGLB	
Description	Reset ARU caddr counter and ARU dynamic route counter
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : ARU counter has no synchronization with CMU clock resolution 1 : Start of ARU counter is synchronized with enabling of CMU clock resolution
	Note: Writing value 1 to this bit field results in a request to reset the ARU caddr counter and ARU dynamic route counter. The next write access to register **CMU_CLK_EN** applies the ARU caddr counter reset, ARU dynamic route counter reset and resets this bit. This feature can be used to synchronize the ARU round trip time to the CMU clocks. **Note:** This bit is write protected. Before writing to this bit, set bit **GTM_CTRL.RF_PROT** to 0. Protected by (**GTM_CTRL.RF_PROT** = 1)

10.6.11 CMU_CLK_CTRL

Description	CMU control for clock resolution generator
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMU_CLK_CTRL
Address	$0xCC$
C-Name	GTM.CLS[0].CMU.CLK_CTRL

Interface: MCS[i]

Name	CMU_CLK_CTRL
Address	$0xCC$
C-Name	

CLK[x]_EXT_DIVIDER	
Description	Input selection for Clock Resolution Generator [x]

CLK[x]_EXT_DIVIDER	
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(CMU_CLK_EN.EN_CLK[x] || CMU_CLK_EN.EN_ECLK[1])
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Use Clock resolution *CMU_GCLK_EN* 1 : Use Clock resolution *CMU_ECLK1_EN* , which corresponds to the clock *CMU_ECLK* [1:1]
	Note: Value can only be modified when clock enable **CMU_CLK_EN.EN_CLK[x]** and **CMU_CLK_EN.EN_ECLK[1]** are disabled. Protected by (**CMU_CLK_EN.EN_CLK[x]** != 0 or **CMU_CLK_EN.EN_ECLK[1]** != 0)

CLK8_EXT_DIVIDER	
Description	Source selection for CMU_CLK_RES[8:8]
Loop	–
Bit Range	$[8 : 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(CMU_CLK_EN.EN_ECLK[0])
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Use Clock resolution '1', which corresponds to the cluster clock *CLS[0]_CLK* operating frequency *CLS[0]_CLK* 1 : Use Clock resolution *CMU_ECLK0_EN* , which corresponds to the clock *CMU_ECLK* [0:0]
	Note: Value can only be modified when **CMU_CLK_EN.EN_ECLK[0]** is disabled. Protected by (**CMU_CLK_EN.EN_ECLK[0]** != 0)

10.7 CMU Port Description

10.7.1 CMU Ports

Loop	–
Condition	–
Format	–

CMU_FXCLK_RES	
Description	CMU fixed clocks with predefined and non configurable clock dividers
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CMU_ECLK	
Description	CMU external clocks
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CMU_CLK_RES	
Description	CMU clock resolutions
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	8 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

10.8 CMU Signal Description

10.8.1 CMU Signals

Loop	–
Condition	–
Format	–

EN_FXCLK	
Description	Enables/disables the clock source for FCR generators

EN_FXCLK	
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMU_GCLK_EN	
Description	Common clock resolution signal generated by Global Clock Resolution Generator
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMU_ECLK0_EN	
Description	Clock resolution signal generated by External Clock Resolution Generator 0
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMU_ECLK1_EN	
Description	Clock resolution signal generated by External Clock Resolution Generator 1
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMU_ECLK2_EN	
Description	Clock resolution signal generated by External Clock Resolution Generator 2
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMU_ECLK_EN	
Description	Refers to CMU_ECLK0_EN or CMU_ECLK1_EN or CMU_ECLK2_EN depending on which External Clock Resolution Generator is under consideration
Loop	–
Condition	–
Signal Type	INT
Assignment	–

11 Cluster Configuration Module (CCM)

11.1 Overview

As already mentioned in chapter *3 "GTM Architecture"* , each submodule of the GTM is aligned explicitly to a cluster. The Cluster Configuration Module (CCM) enables the configuration of several cluster specific options namely:

- cluster's clock frequency,
- module clock gating,
- status observation of the cluster's MCS bus master (AEM),
- address range protection, and
- global architecture configuration.

Indices and their range as used inside this chapter are:

- i:={0, 1,..., NCMM−1} instance index of cluster / CCM module
- a:={0, 1,..., NARP−1} index of address range protectors

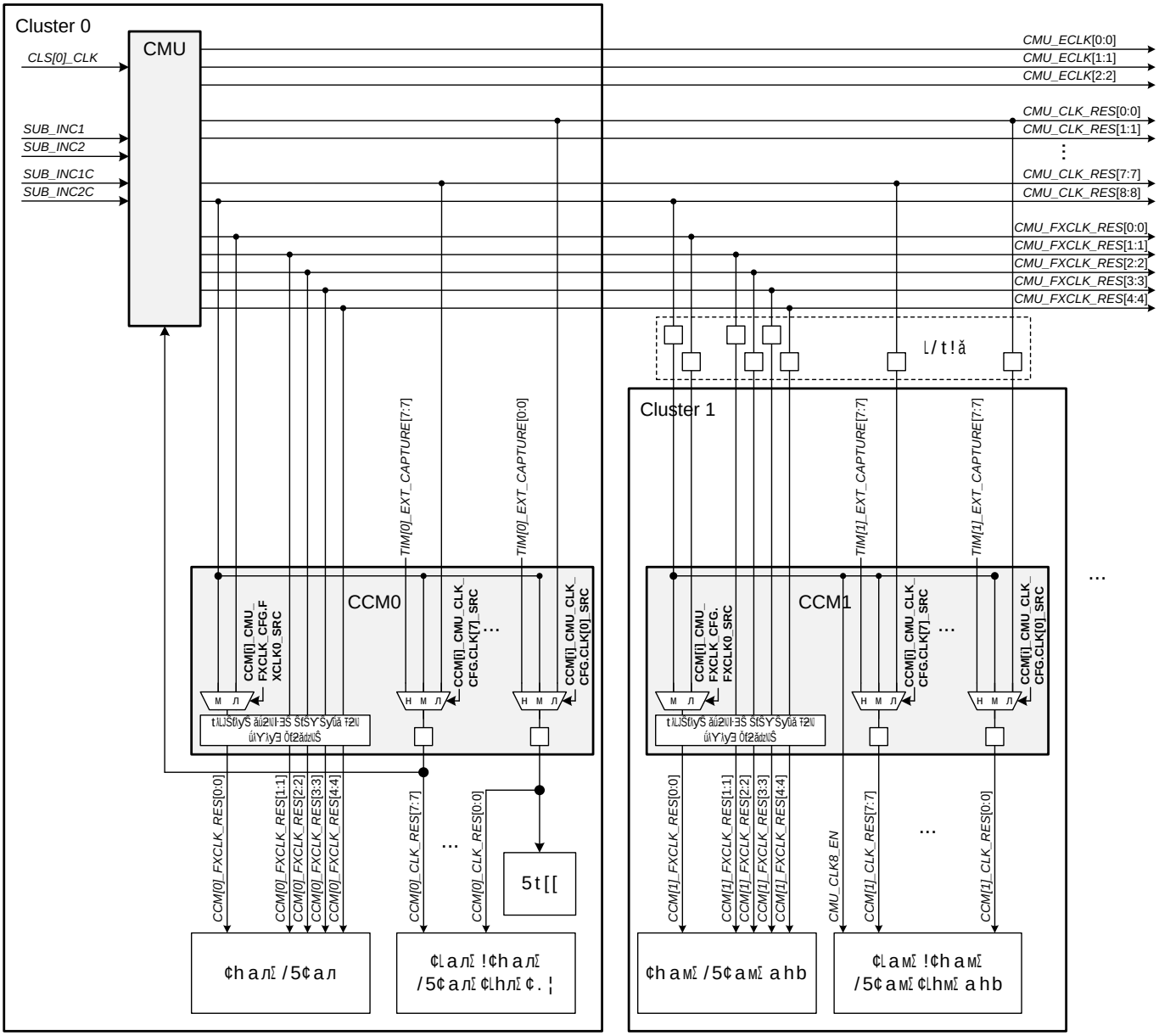
11.2 Cluster CMU clock resolution

The registers **CCM[i]_CMU_CLK_CFG** and **CCM[i]_CMU_FXCLK_CFG** allow the configuration of various cluster clock frequencies.

Figure *38 "Cluster Clock Resolution Wiring"* shows important details about the wiring of the cluster's local clock signals. It is worth mentioning that *CCM[0]_CLK_RES* [7:7] is routed to the CMU module in which it can be used for clock resolution generation.

Clock resolution signals (*CMU_CLK_RES* [x:x], x ∈ {0, 1, ..., 8}) are routed from cluster 0 to the remaining clusters through ICPAs *3.10 "Inter-Cluster Pulse Adapter"* . When a clock resolution signal with an effective clock frequency *10.2 "Global Clock Resolution Generator"* greater than *CLK* / 2 of is routed from cluster 0 with *CLS[0]_CLK* = *CLK* to cluster z ∈ {1, ..., NCMM−1} with *CLS[z]_CLK* = *CLK* / 2, the effective clock frequency of the resolution signal at the output of ICPA is limited to *CLK* / 2.

Figure 38 Cluster Clock Resolution Wiring

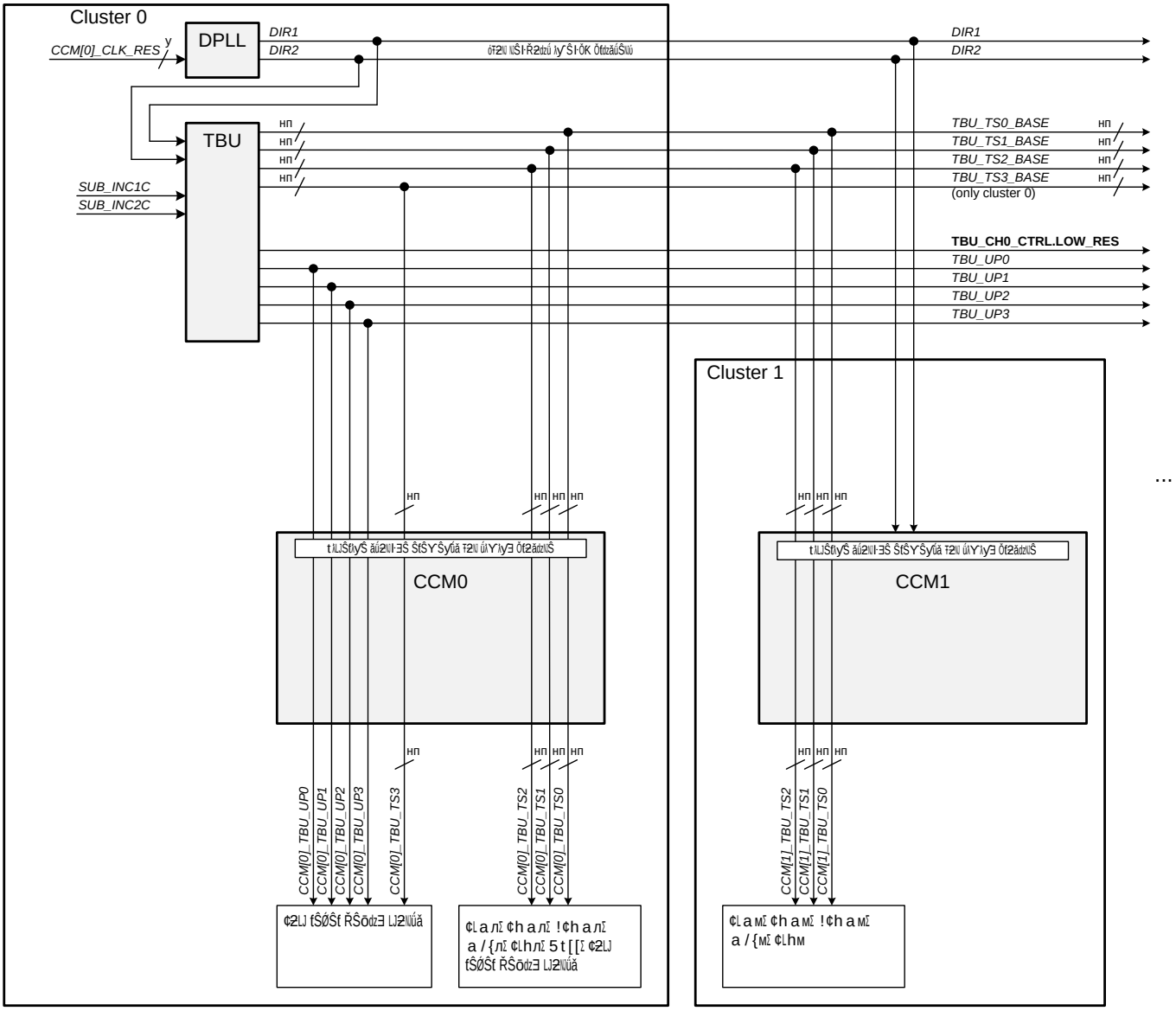


The register **CCM[i]_AEIM_STA** captures the address and the reason of the first invalid AEIM bus master access of the cluster's MCS module.

11.3 Cluster Timebase distribution

The content of this chapter has to be edited.

Figure 39 Cluster Timebase Wiring



11.4 Address Range Protection

The CCM also provides so called address range protectors (ARPs) up to NARP. An ARP can be used to define a configurable write protected address range in order to support enhanced safety features.

The protected address range is mapped to the address range of the cluster's MCS RAM port.

Each ARP[a] can be configured by the registers **CCM[i]_ARP[a]_CTRL** and **CCM[i]_ARP[a]_PROT** , where the register **CCM[i]_ARP[a]_CTRL** enables to configure the size and base address offset for an ARP and the register **CCM[i]_ARP[a]_PROT** configures, that an MCS-channel x with a set bit field **CCM[i]_ARP[a]_PROT.WPROT[x]** cannot write to the corresponding a-th ARP. Whenever an MCS-channel x is writing to an ARP that does not allow a write access from channel x by the configuration register **CCM[i]_ARP[a]_PROT** , the write access is discard. The bit field **CCM[i]_ARP[a]_CTRL.WPROT_AEI** allows to configure if a CPU write access (via AEI slave interface) to the a-th ARP is protected. If the CPU wants to write to the a-th ARP while **CCM[i]_ARP[a]_CTRL.WPROT_AEI** is set, the write access will be discarded and the AEI status signal will signalize an invalid module access.

Considering the size and base address of an ARP, it should be noted that the configuration possibilities are limited. Details about the configuration can be found in the register description of **CCM[i]_ARP[a]_CTRL** .

The bit field **CCM[i]_ARP[a]_CTRL.DIS_PROT** changes the meaning of an ARP configuration in such a way that it explicitly allows an MCS-channel x with a set bit field **CCM[i]_ARP[a]_PROT.WPROT[x]** to write to the a-th ARP. Accordingly, if the bit **CCM[i]_ARP[a]_CTRL.DIS_PROT** is set while the bit **CCM[i]_ARP[a]_CTRL.WPROT_AEI** is also set, the a-th ARP explicitly allows a write access from the CPU to the a-th ARP. A meaningful application of an ARP with a bit field **CCM[i]_ARP[a]_CTRL.DIS_PROT** that is set for an MCS-channel x has another ARP with a surrounding wider address range that defines a write protection for MCS-channel x and some other MCS-channels. Since the address range of an ARP can surround another ARP, it is possible to configure contradictory conditions for MCS-channels or the CPU within the overlapping area (e.g. if ARP y surrounds ARP z and ARP y allows a write access for an MCS-channel x but ARP z prohibits a write access for MCS-channel x). In order to resolve this ambiguity, the following rule is defined: A write protection for a specific address c concerning MCS-channel x (the CPU) is active, if and only if, address c is covered by at least one ARP with a cleared bit

CCM[i]_ARP[a]_CTRL.DIS_PROT and a set bit **CCM[i]_ARP[a]_PROT.WPROT[x]** (**CCM[i]_ARP[a]_CTRL.WPROT_AEI**) and there exists no ARP covering address c with a set bit field **CCM[i]_ARP[a]_CTRL.DIS_PROT** and a set bit field **CCM[i]_ARP[a]_PROT.WPROT[x]** (**CCM[i]_ARP[a]_CTRL.WPROT_AEI**).

11.5 CCM Configuration Registers Description

11.5.1 CCM[i]_PROT

Description	CCM[i] Protection Register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CCM}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_PROT
Address	$0x20000 * i + 0x41FC$
C-Name	GTM.CLS[i].CCM.PROT

Interface: MCS[i]

Name	CCM[i]_PROT
Address	$0x41FC$
C-Name	

CLS_PROT	
Description	Cluster Protection
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Write Protection of cluster configuration registers disabled. 1 : Write Protection of cluster configuration registers enabled.

11.5.2 CCM[i]_CFG

Description	CCM[i] Configuration Register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	$\mathrm{CCM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_CFG
Address	$0x20000 * i + 0x41F8$
C-Name	GTM.CLS[i].CCM.CFG

Interface: MCS[i]

Name	CCM[i]_CFG
Address	$0x41F8$
C-Name	

EN_TIM	
Description	Enable TIM
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NTIM} > i$
Initial value	$\mathrm{NTIM} > i$
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable clock signal for submodule TIM. 1 : Enable clock signal for submodule TIM.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

EN_TOM_SPE_TDTM	
Description	Enable TOM, SPE and TDTM
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NTOM} > i \vert\vert \mathrm{NSPE} > i$
Initial value	$\mathrm{NTOM} > i \vert\vert \mathrm{NSPE} > i$
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async

EN_TOM_SPE_TDTM	
RW-Coding	0 : Disable clock signal for modules TOM, SPE, and its related DTM modules. 1 : Enable clock signal for modules TOM, SPE, and its related DTM modules.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

EN_ATOM_ADTM	
Description	Enable ATOM and ADTM
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NATOM} > i$
Initial value	$\mathrm{NATOM} > i$
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable clock signal for modules ATOM and its related DTM modules. 1 : Enable clock signal for modules ATOM and its related DTM modules.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

EN_MCS	
Description	Enable MCS
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > i$
Initial value	$\mathrm{NMCS} > i$
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable clock signal for module MCS. 1 : Enable clock signal for module MCS.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

EN_DPLL_MAP	
Description	Enable DPLL and MAP
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NDPLL} > i$
Initial value	$\mathrm{NDPLL} > i$
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable clock signal for modules DPLL and MAP. 1 : Enable clock signal for modules DPLL and MAP.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

EN_BRC	
Description	Enable BRC
Loop	–
Bit Range	$[5:5]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NBRC} > i$
Initial value	$\mathrm{NBRC} > i$
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable clock signal for module BRC. 1 : Enable clock signal for module BRC.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

EN_PSM	
Description	Enable PSM
Loop	–
Bit Range	$[6:6]$
Access Type	RW
Volatile	True
Multithread	False

EN_PSM	
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > i$
Initial value	$\mathrm{NPSM} > i$
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable clock signal for module PSM. 1 : Enable clock signal for module PSM.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

EN_CMP_MON	
Description	Enable CMP and MON
Loop	–
Bit Range	$[7 : 7]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$i == 1 \&\& (\mathrm{NCMP} > 0 \| \| \mathrm{NMON} > 0)$
Initial value	$i == 1 \&\& (\mathrm{NCMP} > 0 \| \| \mathrm{NMON} > 0)$
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable clock signal for modules CMP and MON. 1 : Enable clock signal for modules CMP and MON.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

EN_TIO_DTM	
Description	Enable TIO and connected DTM
Loop	–
Bit Range	$[8 : 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{CTIO}[i] == 1 \&\& i < \mathrm{NCCM}$
Initial value	1
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async

EN_TIO_DTM	
RW-Coding	0 : Disable clock signal for module TIO and its related DTM modules. 1 : Enable clock signal for module TIO and its related DTM modules.
	Note: This bit is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

CLS_CLK_DIV	
Description	Cluster Clock Divider
Loop	-
Bit Range	$[17 : 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$1 + \mathrm{CFG\_CLOCK\_RATE}$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0b00 : Cluster is disabled 0b01 : Cluster is enabled without clock divider 0b10 : Cluster is enabled with clock divider 2
	Note: The value of this bit field mirrors the bit field **GTM_CLS_CLK_CFG.CLS[i]_CLK_DIV** , whereas i equals the cluster index.

TBU_DIR[z]	
Description	DIR1 input signal of module TBU timebase [z].
Loop	$z = \{n : 1 <= n <= 2\}$
Bit Range	$[z + 29 : z + 29]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NDPLL} > 0$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0b0 : indicating forward direction 0b1 : indicating backward direction

Note:
The module specific clock enable registers (bit field EN_*) are only implemented if the corresponding module is available in the i-th cluster.

11.5.3 CCM[i]_CMU_CLK_CFG

Description	CCM[i] CMU Clock Configuration Register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CCM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_CMU_CLK_CFG
Address	$0x20000 * i + 0x41F0$
C-Name	GTM.CLS[i].CCM.CMU_CLK_CFG

Interface: MCS[i]

Name	CCM[i]_CMU_CLK_CFG
Address	$0x41F0$
C-Name	

CLK[y]_SRC	
Description	Clock [y] source signal selector
Loop	$y = \{n : 0 <= n <= 7\}$
Bit Range	$[y * 4 + 1 : y * 4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Use *CMU_CLK_RES* [y:y] signal of CMU as *CCM[i]_CLK_RES* [y:y] signal within Cluster. 0b01 : Use *CMU_CLK_RES* [8:8] signal of CMU as *CCM[i]_CLK_RES* [y:y] signal within Cluster. 0b10 : Use *TIM_EXT_CAPTURE* [y:y] of TIM[i] signal as *CCM[i]_CLK_RES* [y:y] signal within cluster.

Note:
The bit fields of this register are only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

11.5.4 CCM[i]_CMU_FXCLK_CFG

Description	CCM[i] CMU Fixed Clock Configuration Register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$i < \text{NTOM}$
Storage Type	REGISTER

Clock Active Cond	$\mathrm{CCM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_CMU_FXCLK_CFG
Address	$0x20000 * i + 0x41F4$
C-Name	GTM.CLS[i].CCM.CMU_FXCLK_CFG

Interface: MCS[i]

Name	CCM[i]_CMU_FXCLK_CFG
Address	$0x41F4$
C-Name	

FXCLK0_SRC	
Description	Fixed clock 0 source signal selector
Loop	-
Bit Range	$[3:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b0000 : Use *CMU_FXCLK_RES* [0:0] signal of CMU as *CCM[i]_FXCLK_RES* [0:0] signal within Cluster. 0b0001 : Use *CMU_CLK_RES* [8:8] signal of CMU as *CCM[i]_FXCLK_RES* [0:0] signal within Cluster.
	Note: These bits are only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

11.5.5 CCM[i]_AEIM_STA

Description	CCM[i] MCS Bus Master Status Register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$i < \mathrm{NMCS}$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CCM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_AEIM_STA
Address	$0x20000 * i + 0x41D8$

C–Name	GTM.CLS[i].CCM.AEIM_STA

Interface: MCS[i]

Name	CCM[i]_AEIM_STA
Address	$0x41D8$
C–Name	

AEIM_XPT_ADDR	
Description	Exception address
Loop	–
Bit Range	$[15:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Invalid bus master (AEIM) address of MCS module.

AEIM_XPT_STA	
Description	AEIM exception status
Loop	–
Bit Range	$[25:24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0b00 : No invalid MCS bus master access occurred 0b10 : Illegal module access of MCS bus master access. 0b11 : Invalid MCS bus master access to an unsupported address.

Note:

Only the first invalid AEIM bus master access of the MCS updates this register with the invalid AEIM address (bit field **CCM[i]_AEIM_STA.AEIM_XPT_ADDR**) and the reason of the invalid access (bit field **CCM[i]_AEIM_STA.AEIM_XPT_STA**). A write access to this register (independent of the written data), always resets the bit fields **CCM[i]_AEIM_STA.AEIM_XPT_STA** and **CCM[i]_AEIM_STA.AEIM_XPT_ADDR** and the next invalid AEIM access is captured again by this register.

Note:

If the i–th cluster does not provide an MCS module, this register is not available.

11.5.6 CCM[i]_ARP[a]_CTRL

Description	CCM[i] Address Range Protector [a] Control Register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; a = \{n : 0 <= n <= \text{NARP} - 1\}$
Condition	$i < \text{NMCS}$
Storage Type	REGISTER
Clock Active Cond	$\text{CCM}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_ARP[a]_CTRL
Address	$0x20000 * i + 0x8 * a + 0x4000$
C–Name	GTM.CLS[i].CCM.ARP[a].CTRL

Interface: MCS[i]

Name	CCM[i]_ARP[a]_CTRL
Address	$0x8 * a + 0x4000$
C–Name	

ADDR	
Description	ARP base address.
Loop	–
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
	Base address for address range protector a. **Note:** The bits 0 to 4 are functionally used for the definition of an ARP but they are always read and written as zeros. **Note:** The actual base address for a protected address range is only defined by the upper 16–(**CCM[i]_ARP[a]_CTRL.SIZE** +2) bits (bit position 2+ **CCM[i]_ARP[a]_CTRL.SIZE** to bit position 15) of bit field **CCM[i]_ARP[a]_CTRL.ADDR** . The lower **CCM[i]_ARP[a]_CTRL.SIZE** +2 bits (bit 0 to **CCM[i]_ARP[a]_CTRL.SIZE** +1) are ignored for the address calculation and assumed as zeros. **Note:** This bit field is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

SIZE	
Description	Size of ARP
Loop	–

SIZE	
Bit Range	[19 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	3
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
	Size of memory range protector a. **Note:** The actual size of a protected memory range is defined as 2^ **CCM[i]_ARP[a]_CTRL.SIZE** address locations, whereas the bit field **CCM[i]_ARP[a]_CTRL.SIZE** is interpreted as an unsigned integer number. **Note:** The values 0, 1 and 2 can be configured, but internally address range will never be assumed lesser than **CCM[i]_ARP[a]_CTRL.SIZE** = 3. **Note:** This bit field is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

DIS_PROT	
Description	Disable ARP protection.
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Bit **CCM[i]_ARP[a]_PROT.WPROT[x]** (**CCM[i]_ARP[a]_CTRL.WPROT_AEI**) defines write protection for selected address range. 1 : Bit **CCM[i]_ARP[a]_PROT.WPROT[x]** (**CCM[i]_ARP[a]_CTRL.WPROT_AEI**) explicitly allows write access to selected address range.
	Note: This bit field is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

WPROT_AEI	
Description	AEI slave write protection.
Loop	–
Bit Range	[31 : 31]
Access Type	RW

WPROT_AEI	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Write protection to address range from AEI slave is disabled. 1 : Write protection to address range from AEI slave is enabled.
	Note: This bit field is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

Note:

The address range interval that is protected by this ARP can be calculated as [(**CCM[i]_ARP[a]_CTRL.ADDR** AND NOT 4*(2^ **CCM[i]_ARP[a]_CTRL.SIZE** −1)); (**CCM[i]_ARP[a]_CTRL.ADDR** AND NOT 4*(2^ **CCM[i]_ARP[a]_CTRL.SIZE** −1)) + 4*(2^ **CCM[i]_ARP[a]_CTRL.SIZE** −1)] assuming a byte wise addressing, an unsigned integer representation for the bit fields **CCM[i]_ARP[a]_CTRL.SIZE** and **CCM[i]_ARP[a]_CTRL.ADDR** . NOT and AND are bitwise logical operators. The incrementation interval for neighboring memory location is always 4.

11.5.7 CCM[i]_ARP[a]_PROT

Description	CCM[i] Address Range Protector [a] Protection Register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; a = \{n : 0 <= n <= \mathrm{NARP} - 1\}$
Condition	$i < \mathrm{NMCS}$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CCM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_ARP[a]_PROT
Address	$0x20000 * i + 0x8 * a + 0x4004$
C–Name	GTM.CLS[i].CCM.ARP[a].PROT

Interface: MCS[i]

Name	CCM[i]_ARP[a]_PROT
Address	$0x8 * a + 0x4004$
C–Name	

WPROT[x]	
Description	Write Protection MCS–channel [x].
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	False
Multithread	False

WPROT[x]	
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	CCM[i]_PROT.CLS_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Write protection to ARP's address range for MCS-channel x is disabled. 1 : Write protection to ARP's address range for MCS-channel x is enabled.

Note:
Only the first T bits of this register (bit 0 to T-1) are functionally implemented. The other bits (bit T to 31) are reserved. Parameter T reflects the number of available MCS-channels in the cluster's MCS module.

Note:
The meaning of the bit field **CCM[i]_ARP[a]_PROT.WPROT[x]** can be changed by the bit field **CCM[i]_ARP[a]_CTRL.DIS_PROT** .

Note:
The bit field is only writable if bit field **CCM[i]_PROT.CLS_PROT** is cleared.

11.5.8 CCM[i]_HW_CONF

Description	CCM[i] Hardware Configuration Register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CCM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_HW_CONF
Address	$0x20000 * i + 0x41DC$
C-Name	GTM.CLS[i].CCM.HW_CONF

Interface: MCS[i]

Name	CCM[i]_HW_CONF
Address	$0x41DC$
C-Name	

GRSTEN	
Description	Global Reset Enable
Loop	-
Bit Range	[0 : 0]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	GRSTEN

GRSTEN	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Global GTM reset register disabled 1 : Global GTM reset register enabled

BRIDGE_MODE_RST	
Description	Bridge mode after reset
Loop	–
Bit Range	[1 : 1]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Bridge starts in synchronous mode after reset 1 : Bridge starts in asynchronous mode after reset

SYNC_INPUT_REG	
Description	Additional pipelined stage in synchronous bridge mode
Loop	–
Bit Range	[2 : 2]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	SYNC_INPUT_REG
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No additional pipelined stage implemented. 1 : additional pipelined stage implemented. All accesses in synchronous mode will be increased by one clock cycle.
	Note: This register is only relevant (if existing) for synchronous bridge mode

CFG_CLOCK_RATE	
Description	Clocks per ARU transfer
Loop	–
Bit Range	[3 : 3]

CFG_CLOCK_RATE	
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	CFG_CLOCK_RATE
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Each system clock an ARU transfer is scheduled 1 : Each second system clock an ARU transfer is scheduled. ARU transfer rate is half the system clock frequency.
	Note: This value defines also the availability of configuration bits in register **GTM_CLS_CLK_CFG** . if **CCM[i]_HW_CONF.CFG_CLOCK_RATE** =0, only the values 0b00 and 0b01 are valid for bit fields **CCM[i]_CFG.CLS_CLK_DIV** . if **CCM[i]_HW_CONF.CFG_CLOCK_RATE** =1, only the values 0b00, 0b01 and 0b10 are valid for bit fields **CCM[i]_CFG.CLS_CLK_DIV** .

ATOM_OUT_RST	
Description	CCM_ATOM_OUT reset level
Loop	–
Bit Range	[4 : 4]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	ATOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : *CCM_ATOM_OUT* reset level is '0' 1 : *CCM_ATOM_OUT* reset level is '1'
	Note: This value represents the ATOM output level after reset. The inverse value of this bit is the reset value of bit **ATOM[i]_CH[x]_CTRL.SL** in all ATOM channels.

ATOM_TRIG_CHAIN	
Description	ATOM trigger chain length without synchronization register
Loop	–
Bit Range	[7 : 5]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

ATOM_TRIG_CHAIN	
Initial value	ATOM_TRIG_CHAIN
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	It defines after which ATOM instance count a synchronization register is introduced into trigger chain (after *ATOM_TRIGOUT* output of instance [i] and *ATOM_TRIGIN* input of instance [i]+1). Valid values are 1 to 7. One (1) means that after each instance a synchronization register is placed.

TOM_OUT_RST	
Description	CCM_TOM_OUT reset level
Loop	–
Bit Range	[8 : 8]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	TOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : *CCM_TOM_OUT* reset level is '0' 1 : *CCM_TOM_OUT* reset level is '1'
	Note: This value represents the TOM output level after reset. The inverse value of this bit is the reset value of bit **TOM[i]_CH[x]_CTRL.SL** in all TOM channels.

TOM_TRIG_CHAIN	
Description	TOM trigger chain length without synchronization register
Loop	–
Bit Range	[11 : 9]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	TOM_TRIG_CHAIN
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	It defines, after which TOM instance count a synchronization register is introduced into trigger chain (after *TOM_TRIGOUT* output of instance [i] and *TOM_TRIGIN* input of instance [i]+1). Valid values are 1 to 7. 1 means that after each instance a synchronization register is placed.

RAM_INIT_RST	
Description	RAM initialization from reset
Loop	–
Bit Range	$[12 : 12]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : RAM is not initialized after reset 1 : RAM is initialized after reset

ERM	
Description	Enable RAM1 MSB for available MCS modules
Loop	–
Bit Range	$[13 : 13]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$i < \mathrm{NMCS}$
Initial value	$\mathrm{MCS\_ERM}[i]$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : MSB of MCS RAM1 address of MCS instance i is not used 1 : MSB of MCS RAM1 address of MCS instance i is used
	Note: This bit reflects the state of the configuration parameter MCS_ERM[i] mentioned in Appendix B.

RESET_ACTIVE	
Description	Active level of asynchronous reset
Loop	–
Bit Range	$[15 : 15]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$\mathrm{RESET\_ACTIVE}$

RESET_ACTIVE	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : asynchronous reset active low 1 : asynchronous reset active high

IRQ_MODE_LEVEL	
Description	Signalize availability of Level IRQ mode
Loop	–
Bit Range	[16 : 16]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : level mode not available 1 : level mode available

IRQ_MODE_PULSE	
Description	Signalize availability of Pulse IRQ mode
Loop	–
Bit Range	[17 : 17]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : pulse mode not available 1 : pulse mode available

IRQ_MODE_PULSE_NOTIFY	
Description	Signalize availability of Pulse Notify IRQ mode
Loop	–
Bit Range	[18 : 18]
Access Type	R
Volatile	False
Multithread	False

IRQ_MODE_PULSE_NOTIFY	
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : pulse notify mode not available 1 : pulse notify mode available

IRQ_MODE_SINGLE_PULSE	
Description	Signalize availability of Single Pulse IRQ mode
Loop	–
Bit Range	[19 : 19]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : single pulse mode not available 1 : single pulse mode available

ATOM_TRIG_INTCHAIN	
Description	ATOM internal trigger chain length for pipeline register
Loop	–
Bit Range	[23 : 20]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	ATOM_TRIG_INTCHAIN
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	4 : Pipeline register present in trigger chain signals of channel 4. This will not have any functional impact if cluster i of device has FAST_CLK_CLS[i]=0 or if the cluster is enabled with clock divider 2 (**GTM_CLS_CLK_CFG.CLS[i]_CLK_DIV** = 2). 8 : No pipeline register in trigger chain signals channel 4 exists

TOM_TRIG_INTCHAIN	
Description	TOM internal trigger chain length for pipeline register

TOM_TRIG_INTCHAIN	
Loop	–
Bit Range	[28 : 24]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	TOM_TRIG_INTCHAIN
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	8 : Pipeline register in trigger chain signals of channel 8 present. This will not have any functional impact if cluster i of device has FAST_CLK_CLS[i]=0 defined. 16 : No pipeline register in trigger chain signals channel 8 exists

INT_CLK_EN_GEN	
Description	Internal clock enable generation
Loop	–
Bit Range	[29 : 29]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	INT_CLK_EN_GEN
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : GTM external clock enable signals in use 1 : GTM internal clock enable signals in use

AEI_ADDR_PIPELINE_STAGE	
Description	Address pipeline stage implemented
Loop	–
Bit Range	[30 : 30]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	AEI_ADDR_PIPELINE_STAGE
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

AEI_ADDR_PIPELINE_STAGE	
R-Coding	0 : no address pipeline stage implemented 1 : address pipeline stage implemented

AEI_RDATA_PIPELINE_STAGE	
Description	Read data pipeline stage implemented
Loop	–
Bit Range	$[31 : 31]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	AEI_RDATA_PIPELINE_STAGE
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no read data pipeline stage implemented 1 : read data pipeline stage implemented

Note:
Reset value depends on the hardware configuration chosen by silicon vendor.

11.5.9 CCM[i]_TIM_AUX_IN_SRC

Description	CCM[i] TIM AUX Input Source Register
Loop	$i = \{n : 0 <= n <= \min(\text{NTIM}, \max(2 * \text{NTOM}, \text{NATOM})) - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CCM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_TIM_AUX_IN_SRC
Address	$0x20000 * i + 0x41E0$
C-Name	GTM.CLS[i].CCM.TIM_AUX_IN_SRC

Interface: MCS[i]

Name	CCM[i]_TIM_AUX_IN_SRC
Address	$0x41E0$
C-Name	

SRC_CH0	
Description	Defines CCM_AUX_IN source of TIM[i] channel 0
Loop	–

SRC_CH0	
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[0] == 0 && GTM_CFG.SRC_IN_MUX == 0 0 : DTM[h/4] of instance [k] Output *DTM_OUT0* [0:0] selected where h = mod(i, 2)*8 and k = i/2 1 : DTM4 of instance [i] Output *DTM_OUT0* [0:0] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[0] == 0 && GTM_CFG.SRC_IN_MUX == 1 0 : DTM0 of instance [i] Output *DTM_OUT0* [0:0] selected 1 : DTM4 of instance [i] Output *DTM_OUT0* [0:0] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[0] == 1 0 : DTM0 of instance [i] Output *DTM_OUT1* [1:1] selected 1 : DTM4 of instance [i] Output *DTM_OUT1* [1:1] selected

SRC_CH1	
Description	Defines CCM_AUX_IN source of TIM[i] channel 1
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[1] == 0 && GTM_CFG.SRC_IN_MUX == 0 0 : DTM[h/4] of instance [k] Output *DTM_OUT0* [1:1] selected where h = mod(i, 2)*8 and k = i/2 1 : DTM4 of instance [i] Output *DTM_OUT0* [1:1] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[1] == 0 && GTM_CFG.SRC_IN_MUX == 1 0 : DTM0 of instance [i] Output *DTM_OUT0* [1:1] selected 1 : DTM4 of instance [i] Output *DTM_OUT0* [1:1] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[1] == 1 0 : DTM0 of instance [i] Output *DTM_OUT1* [2:2] selected 1 : DTM4 of instance [i] Output *DTM_OUT1* [2:2] selected

SRC_CH2	
Description	Defines CCM_AUX_IN source of TIM[i] channel 2
Loop	–
Bit Range	[2 : 2]

SRC_CH2	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[2] == 0 && GTM_CFG.SRC_IN_MUX == 0 0 : DTM[h/4] of instance [k] Output *DTM_OUT0* [2:2] selected where h = mod(i, 2)*8 and k = i/2 1 : DTM4 of instance [i] Output *DTM_OUT0* [2:2] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[2] == 0 && GTM_CFG.SRC_IN_MUX == 1 0 : DTM0 of instance [i] Output *DTM_OUT0* [2:2] selected 1 : DTM4 of instance [i] Output *DTM_OUT0* [2:2] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[2] == 1 0 : DTM0 of instance [i] Output *DTM_OUT1* [3:3] selected 1 : DTM4 of instance [i] Output *DTM_OUT1* [3:3] selected

SRC_CH3	
Description	Defines CCM_AUX_IN source of TIM[i] channel 3
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[3] == 0 && GTM_CFG.SRC_IN_MUX == 0 0 : DTM[h/4] of instance [k] Output *DTM_OUT0* [3:3] selected where h = mod(i, 2)*8 and k = i/2 1 : DTM4 of instance [i] Output *DTM_OUT0* [3:3] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[3] == 0 && GTM_CFG.SRC_IN_MUX == 1 0 : 0 = DTM0 of instance [i] Output *DTM_OUT0* [3:3] selected 1 : DTM4 of instance [i] Output *DTM_OUT0* [3:3] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[3] == 1 0 : DTM1 of instance [i] Output *DTM_OUT1* [0:0] selected 1 : DTM5 of instance [i] Output *DTM_OUT1* [0:0] selected

SRC_CH4	
Description	Defines CCM_AUX_IN source of TIM[i] channel 4
Loop	–
Bit Range	[4 : 4]
Access Type	RW

SRC_CH4	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[4] == 0 && GTM_CFG.SRC_IN_MUX == 0 0 : DTM[h/4+1] of instance [k] Output *DTM_OUT0* [0:0] selected where h = mod(i, 2)*8 and k = i/2 1 : DTM5 of instance [i] Output *DTM_OUT0* [0:0] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[4] == 0 && GTM_CFG.SRC_IN_MUX == 1 0 : DTM1 of instance [i] Output *DTM_OUT0* [0:0] selected 1 : DTM5 of instance [i] Output *DTM_OUT0* [0:0] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[4] == 1 0 : DTM1 of instance [i] Output *DTM_OUT1* [1:1] selected 1 : DTM5 of instance [i] Output *DTM_OUT1* [1:1] selected

SRC_CH5	
Description	Defines CCM_AUX_IN source of TIM[i] channel 5
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[5] == 0 && GTM_CFG.SRC_IN_MUX == 0 0 : DTM[h/4+1] of instance [k] Output *DTM_OUT0* [1:1] selected where h = mod(i, 2)*8 and k = i/2 1 : DTM5 of instance [i] Output *DTM_OUT0* [1:1] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[5] == 0 && GTM_CFG.SRC_IN_MUX == 1 0 : DTM1 of instance [i] Output *DTM_OUT0* [1:1] selected 1 : DTM5 of instance [i] Output *DTM_OUT0* [1:1] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[5] == 1 0 : DTM1 of instance [i] Output *DTM_OUT1* [2:2] selected 1 : DTM5 of instance [i] Output *DTM_OUT1* [2:2] selected

SRC_CH6	
Description	Defines CCM_AUX_IN source of TIM[i] channel 6
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	False

SRC_CH6	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[6] == 0 && GTM_CFG.SRC_IN_MUX == 0 0 : DTM[h/4+1] of instance [k] Output *DTM_OUT0* [2:2] selected where h = mod(i, 2)*8 and k = i/2 1 : DTM5 of instance [i] Output *DTM_OUT0* [2:2] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[6] == 0 && GTM_CFG.SRC_IN_MUX == 1 0 : DTM1 of instance [i] Output *DTM_OUT0* [2:2] selected 1 : DTM5 of instance [i] Output *DTM_OUT0* [2:2] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[6] == 1 0 : DTM1 of instance [i] Output *DTM_OUT1* [3:3] selected 1 : DTM5 of instance [i] Output *DTM_OUT1* [3:3] selected

SRC_CH7	
Description	Defines CCM_AUX_IN source of TIM[i] channel 7
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[7] == 0 && GTM_CFG.SRC_IN_MUX == 0 0 : DTM[h/4+1] of instance [k] Output *DTM_OUT0* [3:3] selected where h = mod(i, 2)*8 and k = i/2 1 : DTM5 of instance [i] Output *DTM_OUT0* [3:3] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[7] == 0 && GTM_CFG.SRC_IN_MUX == 1 0 : DTM1 of instance [i] Output *DTM_OUT0* [3:3] selected 1 : DTM5 of instance [i] Output *DTM_OUT0* [3:3] selected
RW-Coding	CCM[i]_TIM_AUX_IN_SRC.SEL_OUT_N_CH[7] == 1 0 : DTM0 of instance [i] Output *DTM_OUT1* [0:0] selected 1 : DTM4 of instance [i] Output *DTM_OUT1* [0:0] selected

SEL_OUT_N_CH[x]	
Description	Use timer output signals (coming from DTM_OUT0 ports) or the inverted output signals (coming from DTM_OUT1 ports) as CCM_AUX_IN source of TIM[i] channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	RW
Volatile	False

SEL_OUT_N_CH[x]	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Use *DTM_OUT0* [q:q] signal as *CCM_AUX_IN* source of TIM[i] channel x; q = mod(x, 4) 1 : Use *DTM_OUT1* [q:q] signal as *CCM_AUX_IN* source of TIM[i] channel x; q = mod(x+1, 4)

11.5.10 CCM[i]_EXT_CAP_EN

Description	CCM[i] External Capture Enable Register
Loop	$i = \{n : 0 <= n <= \min(\text{NMCS}, \text{NTIM}) - 1\}$
Condition	$i < \text{NMCS}\&\&i < \text{NTIM}$
Storage Type	REGISTER
Clock Active Cond	$\text{CCM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_EXT_CAP_EN
Address	$0x20000 * i + 0x41E4$
C-Name	GTM.CLS[i].CCM.EXT_CAP_EN

Interface: MCS[i]

Name	CCM[i]_EXT_CAP_EN
Address	$0x41E4$
C-Name	

TIM_I_EXT_CAP_EN[x]	
Description	TIM_EXT_CAPTURE signal [x] forwarding enable
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

TIM_I_EXT_CAP_EN[x]	
RW-Coding	0 : disable forwarding of signal *TIM_EXT_CAPTURE* [x:x] of TIM[i] to MCS[i] 1 : enable forwarding of signal *TIM_EXT_CAPTURE* [x:x] of TIM[i] to MCS[i]
	Note: The trigger event forwarding is possible from TIM[i] and TIM[i+1] to MCS[i].

TIM_IP1_EXT_CAP_EN[x]	
Description	TIM_EXT_CAPTURE[x:x] of instance TIM[i+1] signal [x] forwarding enable
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x+8 : x+8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$i + 1 < \mathrm{NTIM}$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : disable forwarding of signal *TIM_EXT_CAPTURE* [x:x] of instance TIM[i+1] to MCS[i] 1 : enable forwarding of signal *TIM_EXT_CAPTURE* [x:x] of instance TIM[i+1] to MCS[i]

11.5.11 CCM[i]_TOM_OUT

Description	CCM[i] TOM Output Register
Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CCM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_TOM_OUT
Address	$0x20000 * i + 0x41E8$
C-Name	GTM.CLS[i].CCM.TOM_OUT

Interface: MCS[i]

Name	CCM[i]_TOM_OUT
Address	$0x41E8$
C-Name	

TOM_OUT[x]	
Description	Output level snapshot of CCM_TOM_OUT channel [x]

TOM_OUT[x]	
Loop	$x = \{n : 0 <= n <= 15\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$i < \mathrm{NTOM}$
Initial value	TOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_TOM_OUT* [x:x] of channel x (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF** for chosen value.

TOM_OUT_N[x]	
Description	Output level snapshot of CCM_TOM_OUT_N channel [x]
Loop	$x = \{n : 0 <= n <= 15\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$i < \mathrm{NTOM}$
Initial value	TOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_TOM_OUT_N* [x:x] of channel x (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF** for chosen value.

11.5.12 CCM[i]_ATOM_OUT

Description	CCM[i] ATOM Output Register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	CCM[i]_CLK_ENABLE == 1

Interface: CPU

Name	CCM[i]_ATOM_OUT
Address	$0x20000 * i + 0x41EC$
C-Name	GTM.CLS[i].CCM.ATOM_OUT

Interface: MCS[i]

Name	CCM[i]_ATOM_OUT
Address	$0x41EC$
C-Name	

ATOM_I_OUT[x]	
Description	Output level snapshot of CCM_ATOM_OUT channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$i <$ NATOM
Initial value	ATOM_OUT_RST
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_ATOM_OUT* [x:x] of channel x (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF** for chosen value.

ATOM_I_OUT_N[x]	
Description	Output level snapshot of CCM_ATOM_OUT_N channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$i <$ NATOM
Initial value	ATOM_OUT_RST
Protect Enable Cond	-

ATOM_I_OUT_N[x]	
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_ATOM_OUT_N* [x:x] of channel x (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF** for chosen value.

ATOM_IP1_OUT[x]	
Description	Output level snapshot of ATOM[i+1]_OUT channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$i + 1 < \mathrm{NATOM}$
Initial value	$\mathrm{ATOM\_OUT\_RST}$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_ATOM_OUT* [x:x] of instance i+1 channel x (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF** for chosen value.

ATOM_IP1_OUT_N[x]	
Description	Output level snapshot of CCM_ATOM_OUT[x:x] of instance i+1 channel x
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$i + 1 < \mathrm{NATOM}$
Initial value	$\mathrm{ATOM\_OUT\_RST}$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

ATOM_IP1_OUT_N[x]	
	Actual level of primary output ports *CCM_ATOM_OUT_N* [x:x] of instance i+1 channel x (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF** for chosen value.

11.5.13 CCM[i]_TIO_G0_OUT

Description	CCM[i] TIO Group 0,1 Output Register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{CCM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_TIO_G0_OUT
Address	$0x20000 * i + 0x41CC$
C-Name	GTM.CLS[i].CCM.TIO_G0_OUT

Interface: MCS[i]

Name	CCM[i]_TIO_G0_OUT
Address	$0x41CC$
C-Name	

TIO_G0_OUT[c]	
Description	Output level snapshot of TIO[i]_G0_OUT channel [c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c : c]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	TIO_OUT_RST
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_TIO_G0_OUT* [c:c] of channel c (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF2.TIO_OUT_RST** for chosen value.

TIO_G1_OUT[c]	
Description	Output level snapshot of TIO[i]_G1_OUT channel [c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c + 8 : c + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIO\_CH8} >= 2$
Initial value	$\mathrm{TIO\_OUT\_RST}$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_TIO_G1_OUT* [c:c] of channel c (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF2.TIO_OUT_RST** for chosen value.

TIO_G0_OUT_N[c]	
Description	Output level snapshot of channel [c].
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c + 16 : c + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$\mathrm{TIO\_OUT\_RST}$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_TIO_G0_OUT_N* [c:c] of channel c (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF2.TIO_OUT_RST** for chosen value.

TIO_G1_OUT_N[c]	
Description	Output level snapshot of channel [c].
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c + 24 : c + 24]$
Access Type	R
Volatile	True

TIO_G1_OUT_N[c]	
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIO\_CH8} >= 2$
Initial value	$\mathrm{TIO\_OUT\_RST}$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_TIO_G1_OUT_N* [c:c] of channel c (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF2.TIO_OUT_RST** for chosen value.

11.5.14 CCM[i]_TIO_G2_OUT

Description	CCM[i] TIO Group 2 Output Register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{NTIO\_CH8} == 3$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CCM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_TIO_G2_OUT
Address	$0x20000 * i + 0x41D0$
C-Name	GTM.CLS[i].CCM.TIO_G2_OUT

Interface: MCS[i]

Name	CCM[i]_TIO_G2_OUT
Address	$0x41D0$
C-Name	

TIO_G2_OUT[c]	
Description	Output level snapshot of TIO[i]_G2_OUT channel [c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c : c]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$\mathrm{TIO\_OUT\_RST}$
Protect Enable Cond	–

TIO_G2_OUT[c]	
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_TIO_G2_OUT* [c:c] of channel c (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF2.TIO_OUT_RST** for chosen value.

TIO_G2_OUT_N[c]	
Description	Output level snapshot of TIO[i]_G2_OUT_N channel [c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c + 16 : c + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$\mathrm{TIO\_OUT\_RST}$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Actual level of primary output ports *CCM_TIO_G2_OUT_N* [c:c] of channel c (after DTM) **Note:** Reset value depends on the hardware configuration chosen by silicon vendor. See **CCM[i]_HW_CONF2.TIO_OUT_RST** for chosen value.

11.5.15 CCM[i]_HW_CONF2

Description	CCM[i] 2. Hardware Configuration Register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CCM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	CCM[i]_HW_CONF2
Address	$0x20000 * i + 0x41D4$
C-Name	GTM.CLS[i].CCM.HW_CONF2

Interface: MCS[i]

Name	CCM[i]_HW_CONF2
Address	$0x41D4$

C-Name	

AXIM_ID_WIDTH	
Description	Defines the number of lower bits which are used for the AXIM transaction IDs
Loop	-
Bit Range	[4 : 0]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	AXIM_ID_WIDTH
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	The value of **AXIM[i]_SLOT[s]_CFG2.AXI_ID** [AXIM_ID_WIDTH-1:0] is used as AXI master ID on the bus.

AXIM_PRIV_ACC	
Description	Privileged AXI master access constant
Loop	-
Bit Range	[5 : 5]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	AXIM_PRIV_ACC
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Privileged AXI master access is not allowed. 1 : Privileged AXI master access is allowed.

AXIM_SEC_ACC	
Description	Secure AXI master access constant
Loop	-
Bit Range	[6 : 6]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	AXIM_SEC_ACC
Protect Enable Cond	-

AXIM_SEC_ACC	
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Secure AXI master access is not allowed. 1 : Secure AXI master access is allowed.

AXIM_POSTED_WRITE	
Description	Write transaction without response
Loop	–
Bit Range	[7 : 7]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	AXIM_POSTED_WRITE
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : always use none-posted write. 1 : use none-posted write.

TIO_OUT_RST	
Description	Reset level for all TIO output signals.
Loop	–
Bit Range	[9 : 9]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	TIO_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : reset level of TIO outputs is '0' 1 : reset level of TIO outputs is '1'
	Note: This value represents the TIO output level after reset.

AXIS_DATA_SIZE	
Description	Defines the data bus width of the AXI slave interface
Loop	–
Bit Range	[16 : 16]
Access Type	R
Volatile	False

AXIS_DATA_SIZE	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	AXIS_DATA_WIDTH == 64
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : AXI Slave bus width 32 bit 1 : AXI Slave bus width 64 bit

AXIM_DATA_SIZE	
Description	Defines the data bus width of the AXI master interface
Loop	–
Bit Range	[18 : 18]
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	AXIM_DATA_WIDTH == 64
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : AXI Master bus width 32 bit 1 : AXI Master bus width 64 bit

11.6 CCM Port Description

11.6.1 CCM clock / reset/ infrastructure interface

Loop	–
Condition	–
Format	–

CCM_RESET	
Description	Asynchronous reset, active reset level defined by constant RESET_ACTIVE_C
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_ENCLK	
Description	Enable signal of CCM_SYS_CLK/CLK
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_CLK	
Description	Clock signal, possible divider values 0,1,2, only gated by gtm_halt
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_ATOM_OUT	
Description	Value of register CCM[i]_ATOM_OUT.ATOM_I_OUT[x]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_ATOM_OUT_N	
Description	Value of register CCM[i]_ATOM_OUT.ATOM_I_OUT_N[x]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–

CCM_ATOM_OUT_N	
Operational Reset	GTM_RESET
Reset Value	–

CCM_TOM_OUT	
Description	Value of register CCM[i]_TOM_OUT.TOM_OUT[x]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	15 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TOM_OUT_N	
Description	Value of register CCM[i]_TOM_OUT.TOM_OUT_N[x]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	15 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TIO_G0_OUT	
Description	Value of register CCM[i]_TIO_G0_OUT.TIO_G0_OUT[c]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TIO_G1_OUT	
Description	Value of register CCM[i]_TIO_G0_OUT.TIO_G1_OUT[c]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR

CCM_TIO_G1_OUT	
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TIO_G2_OUT	
Description	Value of register CCM[i]_TIO_G2_OUT.TIO_G2_OUT[c]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TIO_G0_OUT_N	
Description	Value of register CCM[i]_TIO_G0_OUT.TIO_G0_OUT_N[c]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TIO_G1_OUT_N	
Description	Value of register CCM[i]_TIO_G0_OUT.TIO_G1_OUT_N[c]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TIO_G2_OUT_N	
Description	Value of register CCM[i]_TIO_G2_OUT.TIO_G2_OUT_N[c]

CCM_TIO_G2_OUT_N	
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

11.6.2 CCM TBU interface

Loop	–
Condition	–
Format	–

CCM_CMU_CLK_EN	
Description	Enable signal of the CMU clocks
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CMU_CLK_EN	
Description	Enable signal vector of CMU clocks 0 .. 8
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CMU_FXCLK_EN	
Description	Enable signal of the CMU fixed clocks
Loop	–
Condition	–
Logical Name	–

CMU_FXCLK_EN	
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

11.6.3 CCM CLS interface

Loop	–
Condition	–
Format	–

CLS_CLK_0_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLS_CLK_1_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLS_CLK_2_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–

CLS_CLK_2_EN	
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLS_CLK_3_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLS_CLK_4_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLS_CLK_5_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLS_CLK_6_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–

CLS_CLK_6_EN	
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLS_CLK_7_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLS_CLK_TIO_EN	
Description	Cluster clock enable signals to switch on/off the clock for the cluster submodules (see register CCM[i]_CFG)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CLK_CLK_DIV	
Description	Value of register CCM[i]_CFG.CLS_CLK_DIV
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	1 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

11.6.4 CCM Ports

Loop	–
Condition	–
Format	–

CCM_AUX_IN	
Description	Auxiliary input source of TIM[i]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM[i]_CLK_RES	
Description	Resolution in use, selected k-th clock output of i-th CCM instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	8 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM[i]_FXCLK_RES	
Description	Resolution in use, selected input fix clock of instance i-th
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM[i]_TBU_TS0	
Description	Time stamp of TBU channel 0, 24 bits of time base register TBU_CH0_BASE (26:3/23:0 dependent on TBU_CH0_CTRL.LOW_RES)
Loop	–
Condition	–

CCM[i]_TBU_TS0	
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM[i]_TBU_TS1	
Description	Time/Angle value of TBU channel 1, time/angle base register TBU_CH1_BASE
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM[i]_TBU_TS2	
Description	Time/Angle value of tbu_CH2, time/angle base register TBU_CH2_BASE
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

GTM_HALT	
Description	Global debug signal for halting the entire GTM
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

11.7 CCM signal description

The register **CCM[i]_CFG** allows disabling the system clock signal for unused submodules of the i-th cluster.

11.7.1 CCM[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	–
Format	–

CCM[j]_CLK_ENABLE	
Description	CCM[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	CLS[j]_CLK_ENABLE

11.7.2 CMU_CLK_ENABLE

Loop	–
Condition	–
Format	–

CMU_CLK_ENABLE	
Description	CMU interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE

11.7.3 TBU_CLK_ENABLE

Loop	–
Condition	–
Format	–

TBU_CLK_ENABLE	
Description	TBU interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE

11.7.4 ARCH_CLK_ENABLE

Loop	–
Condition	–
Format	–

ARCH_CLK_ENABLE	
Description	ARCH interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLK_EN

11.7.5 ARU_CLK_ENABLE

Loop	–
Condition	$\mathrm{NARU} == 1$
Format	–

ARU_CLK_ENABLE	
Description	ARU interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE

11.7.6 MCFG_CLK_ENABLE

Loop	–
Condition	$\mathrm{NMCFG} == 1$
Format	–

MCFG_CLK_ENABLE	
Description	MCFG interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE

11.7.7 ICM_CLK_ENABLE

Loop	–
Condition	–
Format	–

ICM_CLK_ENABLE	
Description	ICM interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE

11.7.8 TIM[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NTIM} - 1\}$
Condition	–
Format	–

TIM[j]_CLK_ENABLE	
Description	TIM[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NTIM]
Assignment	CLS[j]_CLK_ENABLE && CCM[j]_CFG.EN_TIM

11.7.9 TOM[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NTOM} - 1\}$
Condition	–
Format	–

TOM[j]_CLK_ENABLE	
Description	TOM[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	CLS[j]_CLK_ENABLE && CCM[j]_CFG.EN_TOM_SPE_TDTM

11.7.10 SPE[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Format	–

SPE[j]_CLK_ENABLE	
Description	SPE[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NSPE]
Assignment	CLS[j]_CLK_ENABLE && CCM[j]_CFG.EN_TOM_SPE_TDTM

11.7.11 ATOM[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NATOM} - 1\}$
Condition	–
Format	–

ATOM[j]_CLK_ENABLE	
Description	ATOM[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	CLS[j]_CLK_ENABLE && CCM[j]_CFG.EN_ATOM_ADTM

11.7.12 MCS[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	–
Format	–

MCS[j]_CLK_ENABLE	
Description	MCS[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NMCS]
Assignment	CLS[j]_CLK_ENABLE && CCM[j]_CFG.EN_MCS

11.7.13 AXIM[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NAXIM} - 1\}$
Condition	–
Format	–

AXIM[j]_CLK_ENABLE	
Description	AXIM[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NAXIM]
Assignment	CLS[j]_CLK_ENABLE && CCM[j]_CFG.EN_MCS

11.7.14 DPLL_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NDPLL} - 1\}$
Condition	–
Format	–

DPLL_CLK_ENABLE	
Description	DPLL interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE && CCM[0]_CFG.EN_DPLL_MAP

11.7.15 MCS2DPLL_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NDPLL} - 1\}$
Condition	–
Format	–

MCS2DPLL_CLK_ENABLE	
Description	MCS2DPLL interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE && CCM[0]_CFG.EN_DPLL_MAP

11.7.16 MAP_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NMAP} - 1\}$
Condition	–
Format	–

MAP_CLK_ENABLE	
Description	MAP interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE && CCM[0]_CFG.EN_DPLL_MAP

11.7.17 BRC_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NBRC} - 1\}$
Condition	–
Format	–

BRC_CLK_ENABLE	
Description	BRC interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[0]_CLK_ENABLE && CCM[0]_CFG.EN_BRC

11.7.18 PSM[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NPSM} - 1\}$
Condition	–
Format	–

PSM[j]_CLK_ENABLE	
Description	PSM[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[PSM]
Assignment	CLS[j]_CLK_ENABLE && CCM[j]_CFG.EN_PSM

11.7.19 CMP_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NCMP} - 1\}$
Condition	–
Format	–

CMP_CLK_ENABLE	
Description	CMP interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[1]_CLK_ENABLE && CCM[1]_CFG.EN_CMP_MON

11.7.20 MON_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NMON} - 1\}$
Condition	–
Format	–

MON_CLK_ENABLE	
Description	MON interface clock is enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	CLS[1]_CLK_ENABLE && CCM[1]_CFG.EN_CMP_MON

11.7.21 TIO[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	–
Format	–

TIO[j]_CLK_ENABLE	
Description	TIO[j] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	(CTIO[j] == 1) && (CLS[j]_CLK_ENABLE && CCM[j]_CFG.EN_TIO_DTM)

11.7.22 CDTM[j]_CLK_ENABLE

Loop	$j = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; d = \{n : 0 <= n <= 11\}$
Condition	–
Format	–

CDTM[j]_DTM[d]_CLK_ENABLE	
Description	CDTM[j]_DTM[d] interface clock is enabled
Loop	–
Condition	–
Signal Type	ARRAY[NCCM,12]
Assignment	((((d < 4) && (TOM_OUT_N_WITH_DTM[j][d * 4] == 1) && CCM[j]_CFG.EN_TOM_SPE_TDTM) || (((d == 4) || (d == 5)) && (ATOM_OUT_N_WITH_DTM[j][(d – 4) * 4] == 1) && CCM[j]_CFG.EN_ATOM_ADTM) || ((d >= 6) && (TIO_OUT_N_WITH_DTM[j][(d – 6) * 4] == 1) && CCM[j]_CFG.EN_TIO_DTM))) && CLS[j]_CLK_ENABLE

12 Time Base Unit (TBU)

12.1 Overview

The Time Base Unit TBU provides common time bases for the GTM-IP. The TBU submodule is organized in channels, where the number of channels is device dependent. There are up to four channels implemented inside the TBU.

Indices and their range as used inside this chapter are:

- x:={1, 2} index for time base channel 1 and channel 2
- y:={0, 1, 2, 3} index of TBU channel

The time base register **TBU_CH0_BASE** of TBU channel 0 is 27 bits wide and it is configurable whether the lower 24 bits or the upper 24 bits are provided to the GTM as signal *TBU_TS0_BASE* .
The two TBU channels 1 and channel 2 have a time base register **TBU_CH[x]_BASE** (x∈{1, 2}) of 24-bit length. The time base register values *TBU_TS1_BASE* and *TBU_TS2_BASE* are provided to subsequent submodules of the GTM.
The time base register of TBU channel 3 **TBU_CH3_BASE** is 24 bits wide. It is used as a modulo counter by **TBU_CH3_BASE_MARK** to get a relative angle clock to **TBU_CH[x]_BASE** . The absolute angle clock value for the current **TBU_CH3_BASE** is captured in **TBU_CH3_BASE_CAPTURE** .

TBU_CH[x]_BASE = **TBU_CH3_BASE_CAPTURE** + **TBU_CH3_BASE** − *DIR1* * **TBU_CH3_BASE_MARK**

TBU_CH[x]_BASE = **TBU_CH3_BASE_CAPTURE** + **TBU_CH3_BASE** − *DIR2* * **TBU_CH3_BASE_MARK**

DIR1 : direction value for time base channel 1
0 up counter
1 down counter

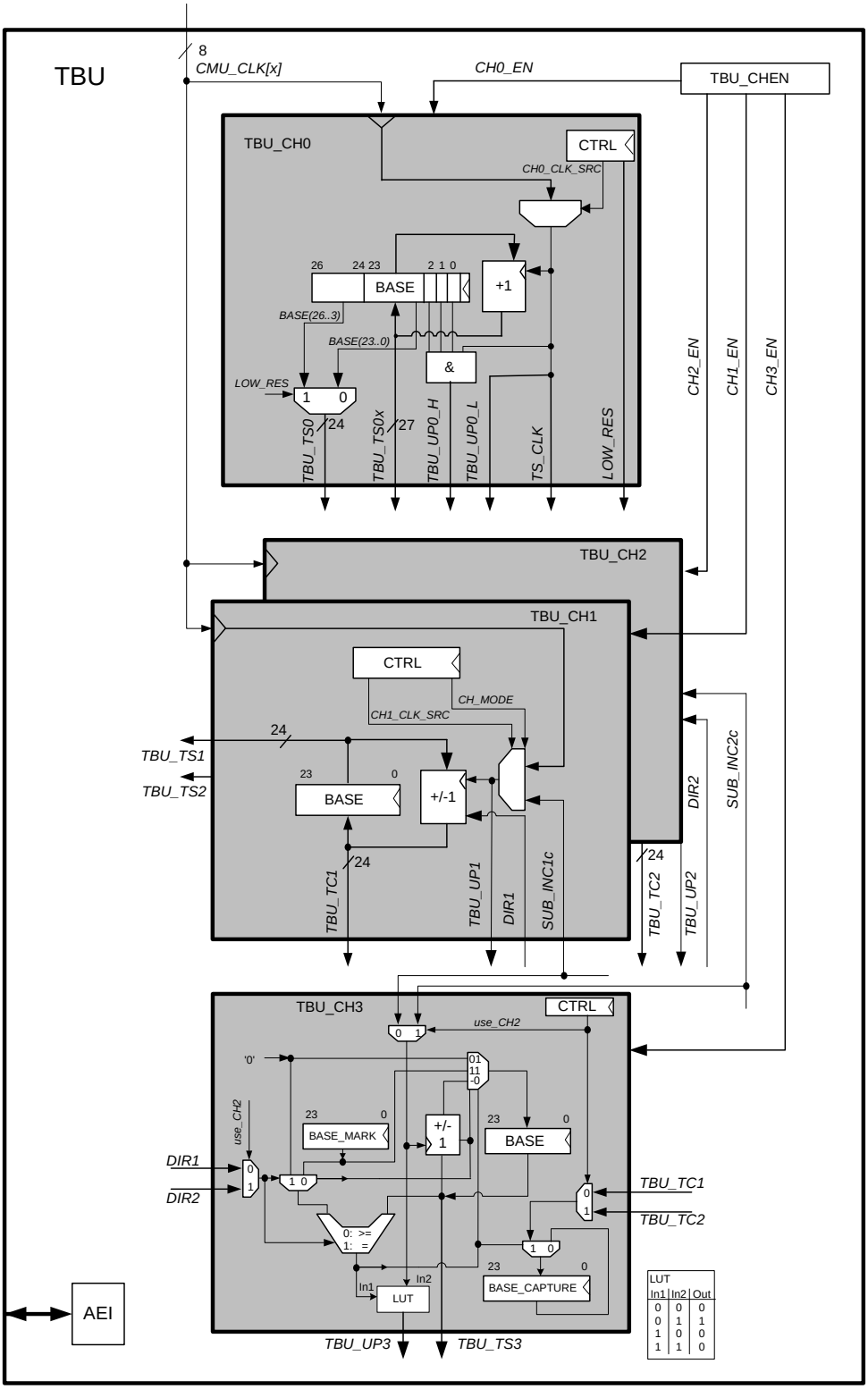
DIR2 : direction value for time base channel 2
0 up counter
1 down counter

Note:
The right-hand sum is limited to 24-bit.

The *TBU_UP1* , *TBU_UP2* and *TBU_UP3* signals are set to high for a single cls_clk period, whenever the corresponding signals *TBU_TS1_BASE* , *TBU_TS2_BASE* and *TBU_TS3_BASE* are updated. The signal *TBU_UP0_L* is set to high for a single cls_clk period, if the signal *TBU_TS0_BASE* and *TBU_TS0X* are updated and *TBU_UP0_H* is set to high for a single cls_clk period, whenever the upper 24-bit of *TBU_TS0_BASE* is updated.

The time base channels can run independently of each other and can be enabled and disabled synchronously by control bits in a global TBU channel enable register **TBU_CHEN** . Chapter *40 "TBU Block Diagram"* shows a block diagram of the Time Base Unit.

Figure 40 TBU Block Diagram



Depending on the device a third TBU channel exists which offers the same functionality as the time base channel 1.

The configuration of the independent time base channels **TBU_CH[x]_BASE** is done via the AEI interface. The TBU channel 0 to channel 2 may select one of the eight *CCM[0]_CLK_RES* [y:y] (y∈{0, 1,..., 7}) signals coming from the CMU submodule.

For TBU channel 1 an additional clock signal *SUB_INC1C* and for TBU channel 2 an additional clock signal *SUB_INC1C* comes from the DPLL and can be selected as input clock for the **TBU_CH[x]_BASE** . This clock in combination with the *DIR1* and *DIR2* signals determines the counter direction of the **TBU_CH[x]_BASE** .

The selected time stamp clock signal for the TBU channel 0 subunit is served via the *TS_CLK* signal line to the DPLL submodule. The *TS_CLK* signal equals the signal *TBU_UP0* .

12.2 TBU Channels

The time base values are generated within the TBU time base channels in two independent and one dependent operation modes.

In all modes, the time base registers **TBU_CH0_BASE** , **TBU_CH[x]_BASE** and **TBU_CH3_BASE** can be initialized with a start value just before enabling the corresponding TBU channel.

Moreover, the time base registers **TBU_CH0_BASE** , **TBU_CH[x]_BASE** and **TBU_CH3_BASE** can always be read in order to determine the actual value of the counter.

12.2.1 Independent Modes

12.2.1.1 Free Running Counter Mode

TBU channel 0 provides a 27−bit counter in a free running counter mode. Depending on the bit field **TBU_CH0_CTRL.LOW_RES** , the lower 24 bits **TBU_CH0_BASE.BASE** [23:0] or the upper 24 bits **TBU_CH0_BASE.BASE** [26:3] are provided to the GTM submodules.

TBU channel 1 and channel 2 provides a 24−bit counter in a free running counter mode enabled by reset of **TBU_CH1_CTRL.CH_MODE** and **TBU_CH2_CTRL.CH_MODE** .

In TBU free running counter mode, the time base registers **TBU_CH0_BASE** and **TBU_CH[x]_BASE** are updated on every specified incoming clock event by the selected signal *CCM[0]_CLK_RES* [y:y] (y∈{0, 1,..., 7}) (depending on **TBU_CH0_CTRL** , **TBU_CH1_CTRL** and **TBU_CH2_CTRL** register). In general, the time base register **TBU_CH[x]_BASE** is incremented on every *CCM[0]_CLK_RES* [y:y] clock tick.

12.2.1.2 Forward/Backward Counter Mode

TBU channel 1 and channel 2 provides a 24−bit forward/backward counter enabled by set of **TBU_CH1_CTRL.CH_MODE** and **TBU_CH2_CTRL.CH_MODE** . In this mode, the *DIR1* and *DIR2* signals provided by the DPLL are taken into account.

The value of the time base register **TBU_CH[x]_BASE** is incremented in case *DIR1* and *DIR2* signals are equal to '0' and decremented in case *DIR1* and *DIR2* signals are '1'.

12.2.2 Dependent Mode

12.2.2.1 Modulo Counter Mode

TBU channel 3 provides a 24 bit forward/backward modulo counter. The clock *SUB_INC1C* and *SUB_INC2C* and counter direction *DIR1* and *DIR2* provided by DPLL is selected by **TBU_CH3_CTRL.USE_CH2** .

The modulo value is defined in **TBU_CH3_BASE_MARK** . In forward counter mode, if **TBU_CH3_BASE** value reaches **TBU_CH3_BASE_MARK** . **TBU_CH3_BASE** is reset and *TBU_TS1_BASE* and *TBU_TS2_BASE* are captured in **TBU_CH3_BASE_CAPTURE** . In backward counter mode, if **TBU_CH3_BASE** value reaches '0' **TBU_CH3_BASE** is set to **TBU_CH3_BASE_MARK** and *TBU_TS1_BASE* and *TBU_TS2_BASE* are captured in **TBU_CH3_BASE_CAPTURE** .

12.3 TBU Configuration Registers Description

12.3.1 TBU_CHEN

Description	TBU global channel enable
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CHEN

Address	$0x100$
C-Name	GTM.CLS[0].TBU.CHEN

Interface: MCS[i]

Name	TBU_CHEN
Address	$0x100$
C-Name	

ENDIS_CH[y]	
Description	TBU channel [y] enable/disable control
Loop	$y = \{n : 0 <= n <= 3\}$
Bit Range	$[2 * y + 1 : 2 * y]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	$y < (3 + \text{USE\_TBU\_CH3})$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0b00 : Channel is disabled 0b01 : Unused 0b10 : Unused 0b11 : Channel is enabled
W-Coding	0b00 : Don't care, bits will not be changed 0b01 : Disable channel 0b10 : Enable channel 0b11 : Don't care, bits will not be changed

Note:
These bits are only applicable if channel is implemented for this device, otherwise read and write as zero

12.3.2 TBU_CH0_CTRL

Description	TBU channel 0 control
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH0_CTRL
Address	$0x104$
C-Name	GTM.CLS[0].TBU.CH0_CTRL

Interface: MCS[i]

Name	TBU_CH0_CTRL
Address	$0x104$
C-Name	

LOW_RES	
Description	TBU_CH0_BASE register resolution
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[0]
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : TBU channel uses lower counter bits [23:0] 1 : TBU channel uses upper counter bits [26:3]
	Note: The two resolutions for the TBU channel 0 can be used in the TIM channel 0 and the DPLL submodules. **Note:** This value can only be modified if channel 0 is disabled.

CH_CLK_SRC	
Description	Clock source for channel 0, channel 1 and channel 2 time base counter
Loop	-
Bit Range	$[3:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[0]
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : *CCM[0]_CLK_RES* [0:0] selected 0b001 : *CCM[0]_CLK_RES* [1:1] selected 0b010 : *CCM[0]_CLK_RES* [2:2] selected 0b011 : *CCM[0]_CLK_RES* [3:3] selected 0b100 : *CCM[0]_CLK_RES* [4:4] selected 0b101 : *CCM[0]_CLK_RES* [5:5] selected 0b110 : *CCM[0]_CLK_RES* [6:6] selected 0b111 : *CCM[0]_CLK_RES* [7:7] selected

CH_CLK_SRC	
	Note: This value can only be modified if channel 0 is disabled.

12.3.3 TBU_CH0_BASE

Description	TBU channel 0 base
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH0_BASE
Address	$0x108$
C-Name	GTM.CLS[0].TBU.CH0_BASE

Interface: MCS[i]

Name	TBU_CH0_BASE
Address	$0x108$
C-Name	

BASE	
Description	Time base value for channel 0
Loop	-
Bit Range	$[26:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[0]
Reset group	HW_RESET: async GTM_RESET: async
	Note: The value of **TBU_CH0_BASE.BASE** can only be written if the TBU channel 0 is disabled **Note:** If channel 0 is enabled, a read access to this register provides the current value of the underlying 27 bit counter.

12.3.4 TBU_CH1_CTRL

Description	TBU channel 1 control
Loop	

Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH1_CTRL
Address	$0x10C$
C-Name	GTM.CLS[0].TBU.CH1_CTRL

Interface: MCS[i]

Name	TBU_CH1_CTRL
Address	$0x10C$
C-Name	

CH_MODE	
Description	Channel mode
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[1]
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Free running counter mode 1 : Forward/backward counter mode
	Note: This value can only be modified if channel 1 is disabled. In free running counter mode the CMU clock source specified by **TBU_CH1_CTRL.CH_CLK_SRC** is used for the counter. In forward/backward counter mode the *SUB_INC1C* clock signal in combination with the *DIR1* input signal is used to determine the counter direction and clock frequency.

CH_CLK_SRC	
Description	Clock source for channel 1 time base counter
Loop	-
Bit Range	$[3:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

CH_CLK_SRC	
Protect Enable Cond	TBU_CHEN.ENDIS_CH[1]
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : *CCM[0]_CLK_RES* [0:0] selected 0b001 : *CCM[0]_CLK_RES* [1:1] selected 0b010 : *CCM[0]_CLK_RES* [2:2] selected 0b011 : *CCM[0]_CLK_RES* [3:3] selected 0b100 : *CCM[0]_CLK_RES* [4:4] selected 0b101 : *CCM[0]_CLK_RES* [5:5] selected 0b110 : *CCM[0]_CLK_RES* [6:6] selected 0b111 : *CCM[0]_CLK_RES* [7:7] selected
	Note: This value can only be modified if channel 1 is disabled

12.3.5 TBU_CH2_CTRL

Description	TBU channel 2 control
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH2_CTRL
Address	$0x114$
C-Name	GTM.CLS[0].TBU.CH2_CTRL

Interface: MCS[i]

Name	TBU_CH2_CTRL
Address	$0x114$
C-Name	

CH_MODE	
Description	Channel mode
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[2]
Reset group	HW_RESET: async GTM_RESET: async

CH_MODE	
RW-Coding	0 : Free running counter mode 1 : Forward/backward counter mode
	Note: This value can be modified only if channel 2 is disabled. In free running counter mode the CMU clock source specified by **TBU_CH2_CTRL.CH_CLK_SRC** is used for the counter. In forward/backward counter mode the *SUB_INC2C* clock signal in combination with the *DIR2* input signal is used to determine the counter direction and clock frequency.

CH_CLK_SRC	
Description	Clock source for channel 2 time base counter
Loop	-
Bit Range	$[3:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[2]
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : *CCM[0]_CLK_RES* [0:0] selected 0b001 : *CCM[0]_CLK_RES* [1:1] selected 0b010 : *CCM[0]_CLK_RES* [2:2] selected 0b011 : *CCM[0]_CLK_RES* [3:3] selected 0b100 : *CCM[0]_CLK_RES* [4:4] selected 0b101 : *CCM[0]_CLK_RES* [5:5] selected 0b110 : *CCM[0]_CLK_RES* [6:6] selected 0b111 : *CCM[0]_CLK_RES* [7:7] selected
	Note: This value can only be modified if channel 2 is disabled

12.3.6 TBU_CH[x]_BASE

Description	TBU channel [x] base
Loop	$x = \{n : 1 <= n <= 2\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH1_BASE
Address	$0x110$
C-Name	GTM.CLS[0].TBU.CH1_BASE
Name	TBU_CH2_BASE

Address	$0x118$
C-Name	GTM.CLS[0].TBU.CH2_BASE

Interface: MCS[i]

Name	TBU_CH1_BASE
Address	$0x110$
C-Name	
Name	TBU_CH2_BASE
Address	$0x118$
C-Name	

BASE	
Description	Time base value for channel [x]
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[x]
Reset group	HW_RESET: async GTM_RESET: async
	Note: The value of **TBU_CH[x]_BASE.BASE** can be written only if the corresponding TBU channel y (y∈{1, 2}) is disabled **Note:** If the corresponding channel y (y∈{1, 2}) is enabled, a read access to this register provides the current value of the underlying counter.

12.3.7 TBU_CH3_CTRL

Description	TBU channel 3 control
Loop	
Condition	$\mathrm{USE\_TBU\_CH3} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH3_CTRL
Address	$0x11C$
C-Name	GTM.CLS[0].TBU.CH3_CTRL

Interface: MCS[i]

Name	TBU_CH3_CTRL
Address	$0x11C$
C-Name	

CH_MODE	
Description	Channel mode
Loop	-
Bit Range	$[0:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Free running counter mode; not available with TBU channel 3 1 : Forward/backward counter mode

USE_CH2	
Description	Channel selector for modulo counter
Loop	-
Bit Range	$[4:4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[3]
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : TBU_CH1 values used. (*SUB_INC1C* for clock, *DIR1* for counter direction, *TBU_TS1_BASE* for capturing) 1 : TBU_CH2 values used. (*SUB_INC2C* for clock, *DIR2* for counter direction, *TBU_TS2_BASE* for capturing)
	Note: This value can only be modified if channel 3 is disabled

12.3.8 TBU_CH3_BASE

Description	TBU channel 3 base
Loop	
Condition	$\mathrm{USE\_TBU\_CH3} > 0$
Storage Type	REGISTER

Clock Active Cond	$\mathrm{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH3_BASE
Address	$0x120$
C-Name	GTM.CLS[0].TBU.CH3_BASE

Interface: MCS[i]

Name	TBU_CH3_BASE
Address	$0x120$
C-Name	

BASE	
Description	Time base value for channel 3
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[3]
Reset group	HW_RESET: async GTM_RESET: async
	Note: The value of **TBU_CH3_BASE.BASE** can only be written if the corresponding TBU channel 3 is disabled **Note:** If the corresponding channel 3 is enabled, a read access to this register provides the current value of the underlying counter.

12.3.9 TBU_CH3_BASE_MARK

Description	TBU channel 3 modulo value
Loop	
Condition	$\mathrm{USE\_TBU\_CH3} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH3_BASE_MARK
Address	$0x124$
C-Name	GTM.CLS[0].TBU.CH3_BASE_MARK

Interface: MCS[i]

Name	TBU_CH3_BASE_MARK
Address	$0x124$
C-Name	

BASE_MARK	
Description	Modulo value for channel 3
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TBU_CHEN.ENDIS_CH[3]
Reset group	HW_RESET: async GTM_RESET: async
	Note: The value of **TBU_CH3_BASE_MARK.BASE_MARK** can only be written if the corresponding TBU channel 3 is disabled

12.3.10 TBU_CH3_BASE_CAPTURE

Description	TBU channel 3 base captured
Loop	
Condition	$\mathrm{USE\_TBU\_CH3} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TBU\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TBU_CH3_BASE_CAPTURE
Address	$0x128$
C-Name	GTM.CLS[0].TBU.CH3_BASE_CAPTURE

Interface: MCS[i]

Name	TBU_CH3_BASE_CAPTURE
Address	$0x128$
C-Name	

BASE_CAPTURE	
Description	Captured value of time base channel 1 or channel 2
Loop	-
Bit Range	$[23:0]$

BASE_CAPTURE	
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: When **TBU_CH3_CTRL.USE_CH2** =0 *TBU_TS1_BASE* is captured and if **TBU_CH3_CTRL.USE_CH2** is setting *TBU_TS2_BASE* is captured.

12.4 TBU Port Description

12.4.1 TBU Ports

Loop	–
Condition	–
Format	–

TBU_TS0_BASE	
Description	TBU channel 0, lower 24 bits of TBU_CH0_BASE[23:0]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_TS0X	
Description	TBU channel 0 to 27 bits of TBU_CH0_BASE[26:0]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	26 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_TS1_BASE	
Description	TBU channel 1, TBU_CH1_BASE[23:0]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_TS2_BASE	
Description	TBU channel 2, TBU_CH2_BASE[23:0]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_TS3_BASE	
Description	TBU channel 3, TBU_CH3_BASE[23:0]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_UP0	
Description	Indication of update
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–

TBU_UP0	
Operational Reset	GTM_RESET
Reset Value	–

TBU_UP0_H	
Description	Indication of update of TBU_TS0_BASE and TBU_TS0X
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_UP0_L	
Description	Indication of update of TBU_TS0X[23:3]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_UP1	
Description	Indication of update of TBU_TS1_BASE
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_UP2	
Description	Indication of update of TBU_TS2_BASE
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC

TBU_UP2	
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TBU_UP3	
Description	Indication of update of TBU_TS3_BASE
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

13 Timer Input Module (TIM)

13.1 Overview

The Timer Input Module (TIM) is responsible for filtering and capturing input signals of the GTM. Several characteristics of the input signals can be measured inside the TIM channels. For advanced data processing, the detected input characteristics of the TIM module can be routed through the ARU to subsequent processing units of the GTM.

Input characteristics mean either time stamp values of detected input rising or falling edges together with the new signal level or the number of edges received since channel enable together with the actual time stamp or PWM signal duration for a whole PWM period.

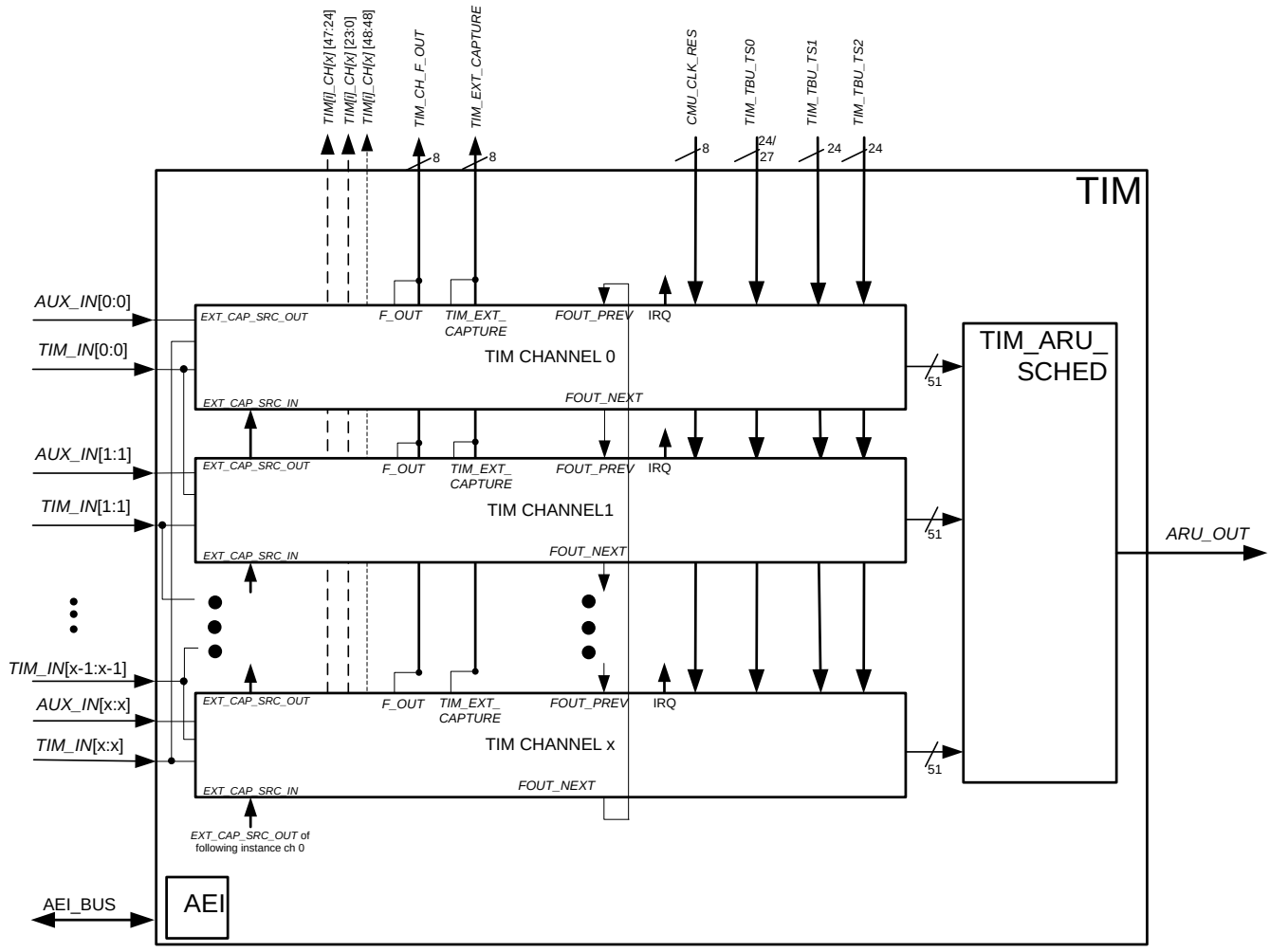
Indices and their range as used inside this chapter are:

- NTIM: number of TIM
- i:={0, 1,..., NTIM−1} instance index of cluster / module
- x:={0, 1,..., 7} channel index in a TIM instance
- m: number of channels available in the TIM sub−module
- y:={0, 1,..., m−1} index for the number TIM sub−module channels

The number of channels m inside a TIM sub−module depends on the device.

The architecture of TIM is shown in Figure *41 "TIM Block Diagram"* .

Figure 41 TIM Block Diagram



Each of the m dedicated input signals are filtered inside the FLT_CHx sub−unit of the TIM module.

Note:

The input signals *TIM_IN* [x:x]/ *TIM_IN* [x−1:x−1] are driven by external logic and has no defined relationship to the internal cluster clock. To avoid metastability it is necessary to synchronize *TIM_IN* [x:x]/ *TIM_IN* [x−1:x−1] via two register stages, which leads to a delay of two cls_clk

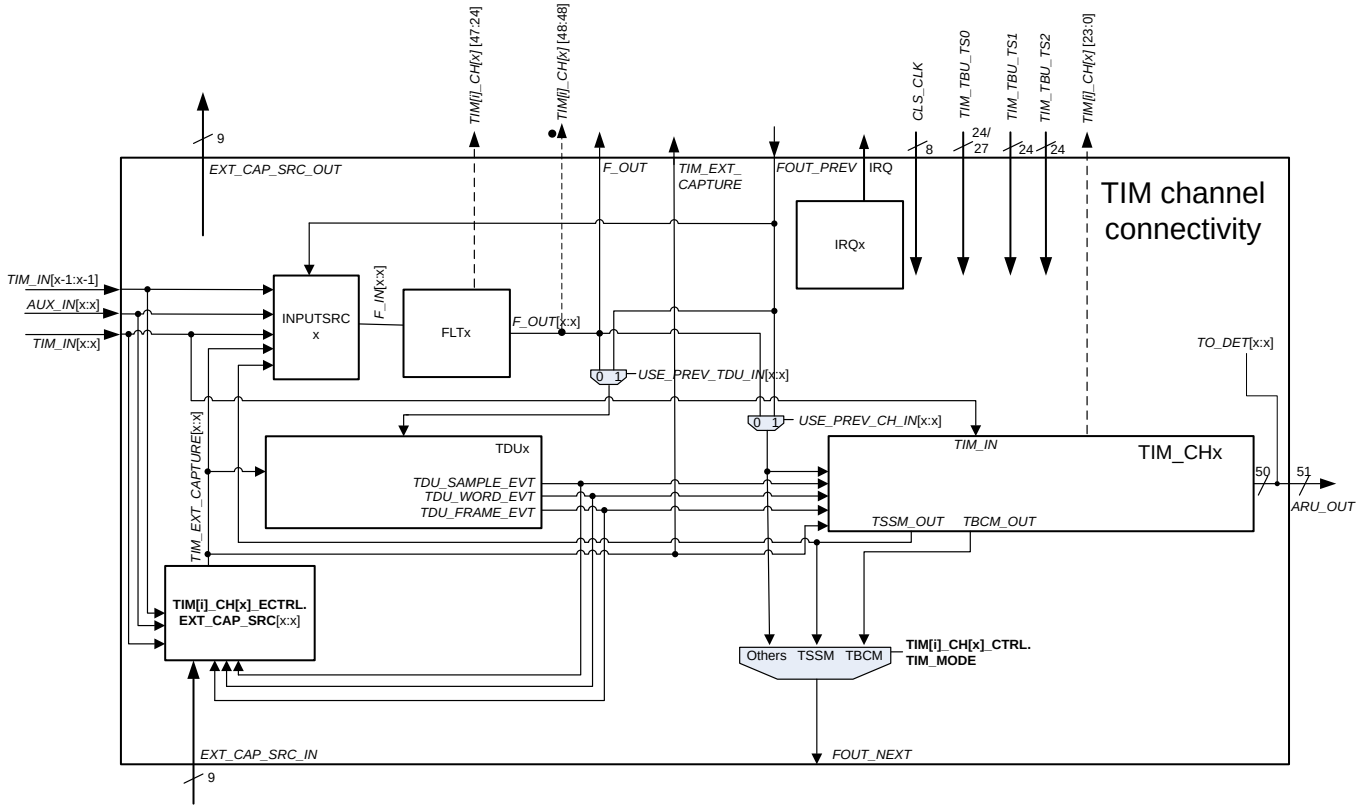
periods for these incoming signals. On the other hand the input signals *AUX_IN* [x:x] are not synchronized as these are generated already synchronously within the GTM IP.

The measurement values can be read by the CPU directly via the AEI-Bus or they can be routed through the ARU to other sub-modules of the GTM.

For the GTM-IP TIM instance i=0 only, the dashed signal outputs *TIM[i]_CH[x]* [23:0], *TIM[i]_CH[x]* [47:24] and *TIM[i]_CH[x]* [48:48] come from the TIM channel 0 to channel 5 and are connected to MAP sub-module. There, these signals are used for further processing and for routing to the DPLL.

The two (three) time bases coming from the TBU are connected to the TIM channels to annotate time stamps to incoming signals. For TIM0, the extended 27-bit width time base *TIM_TBU_TS0* is connected to the TIM channels, and the user has to select if the lower 24 bits (*TIM_TBU_TS0* [23:0]) or the higher 24 bits (*TIM_TBU_TS0* [26:3]) are stored inside the **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** registers.

Figure 42 TIM Channel Internal Connectivity



- Above figure gives an overview of the channel internal connectivity of the sub units. The sub units with the major functionality are listed next:
- INPUTSRC_CHx: Select signal for processing by the filter unit FLT_CHx
- FLT_CHx: The filter unit provides different filter mechanisms described in more detail in chapter *13.2 "TIM Filter Functionality (FLT)"* .
- TDU_CHx: Timeout detection unit (no subsequent edge detected during a specified duration)
- TIM_CHx: Measurement unit; different measurements strategies configurable on the filtered signal
- IRQ_CHx: Local interrupt controller (enabling, status, ..)
- **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** : Selects a local signal *TIM_EXT_CAPTURE* [x:x] which is needed by certain functions
- Details are given in the next chapters.

Note:

Depending on the values of the configuration bit fields **TIM[i]_CH[x]_ECTRL.USE_PREV_TDU_IN** , **TIM[i]_CH[x]_ECTRL.USE_PREV_CH_IN** , it is possible to operate on the signal of the local channel x or the previous channel x-1.

Depending on the value of the configuration bit field **TIM[i]_CH[x]_CTRL.TIM_MODE** it is possible to provide different signals (via *FOUT_NEXT*) to the next channel.

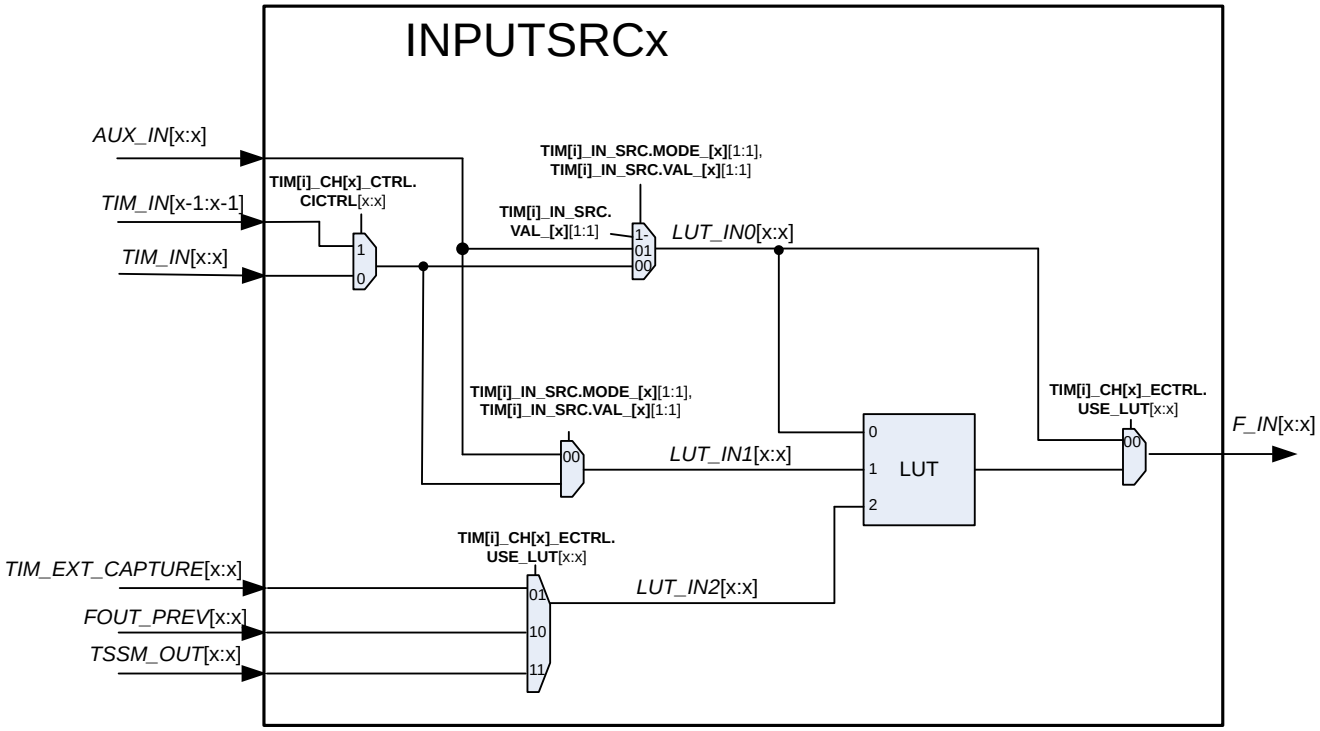
In TBCM mode each capture event selected by the sensitive edges (**TIM[i]_CH[x]_CNTS.CNTS**) will be forwarded with the value of **TIM[i]_CH[x]_ECNT** [0] to the following channel (via *FOUT_NEXT*).

13.1.1 Input Source Selection INPUTSRCx

It can be configured which source is used for processing in the FLT,TDU,TIM_CH units. It can be selected by the bit fields **TIM[i]_CH[x]_CTRL.CICTRL** and **TIM[i]_IN_SRC.MODE_[x]** , **TIM[i]_IN_SRC.VAL_[x]** in the register **TIM[i]_IN_SRC** which source is in use.

Alternatively, the signal *F_IN* [x:x] can be generated by a 8-bit lookup table, which allows to define any function of 3 input sources.

Figure 43 INPUTSRC Block Diagram



If **TIM[i]_CH[x]_ECTRL.USE_LUT** =0b00 is set, the lookup table signal generation is bypassed and the signal is selected as follows:

In a certain **TIM[i]_IN_SRC.MODE_[x]** , **TIM[i]_IN_SRC.VAL_[x]** combination the input signal *F_IN* [x:x] can be driven by **TIM[i]_IN_SRC.VAL_[x]** [1:1] with 0 or 1 directly.

Due to the fact that all 8 channels are bundled in the register **TIM[i]_IN_SRC** a synchronous control of all 8 input channels is possible.

Two adjacent channels can be combined by setting the **TIM[i]_CH[x]_CTRL.CICTRL** bit field. This allows combining of complex measurements on one input signal with two TIM channels.

The additional signal *AUX_IN* [x:x] can be selected as an input signal. The source of this signal is defined in the chapter *5 "TIM Auxiliary Input Multiplexing"* .

If **TIM[i]_CH[x]_ECTRL.USE_LUT** !=0b00 is set, the lookup table signal generation with following inputs is in use. See Figure *43 "INPUTSRC Block Diagram"* :

Input *LUT_IN0* [x:x] selection:
TIM[i]_INP_VAL.TIM_IN[x] if **TIM[i]_CH[x]_CTRL.CICTRL** =0 and **TIM[i]_IN_SRC.MODE_[x]** [1:1] = 0 and **TIM[i]_IN_SRC.VAL_[x]** [1:1] = 0;
TIM[i]_INP_VAL.TIM_IN[x-1] if **TIM[i]_CH[x]_CTRL.CICTRL** =1 and **TIM[i]_IN_SRC.MODE_[x]** [1:1]=0 and **TIM[i]_IN_SRC.VAL_[x]** [1:1] = 0;
AUX_IN [x:x] if **TIM[i]_IN_SRC.MODE_[x]** [1:1]=0 and **TIM[i]_IN_SRC.VAL_[x]** [1:1] = 1;
TIM[i]_IN_SRC.VAL_[x] if **TIM[i]_IN_SRC.MODE_[x]** [1:1] = 1;

Input *LUT_IN1* [x:x] selection:
AUX_IN [x:x] if **TIM[i]_IN_SRC.MODE_[x]** [1:1] = 0 and **TIM[i]_IN_SRC.VAL_[x]** [1:1] = 0
TIM[i]_INP_VAL.TIM_IN[x] if **TIM[i]_CH[x]_CTRL.CICTRL** =0;
TIM[i]_INP_VAL.TIM_IN[x-1] if **TIM[i]_CH[x]_CTRL.CICTRL** =1

- Input *LUT_IN2* [x:x] selection:
- *TIM_EXT_CAPTURE* [x:x] if **TIM[i]_CH[x]_ECTRL.USE_LUT** =0b01
- *FOUT_PREV* [x:x] if **TIM[i]_CH[x]_ECTRL.USE_LUT** =0b10
- *TSSM_OUT* [x:x] if **TIM[i]_CH[x]_ECTRL.USE_LUT** =0b11

The lookup table is defined by the contents of the bit field **TIM[i]_CH[x]_TDUC.TO_CNT2** . The lookup_table_index is defined by *LUT_IN2* [x:x] & *LUT_IN1* [x:x] & *LUT_IN0* [x:x]. The signal *F_IN* [x:x] is generated by **TIM[i]_CH[x]_TDUC.TO_CNT2** [lookup_table_index].

If **TIM[i]_CH[x]_ECTRL.USE_LUT** !=0b00 is set, only limited functionality is available in the TDU. See bit field **TIM[i]_CH[x]_TDUV.SLICING** in the register **TIM[i]_CH[x]_TDUV** .

13.1.2 Input Observation

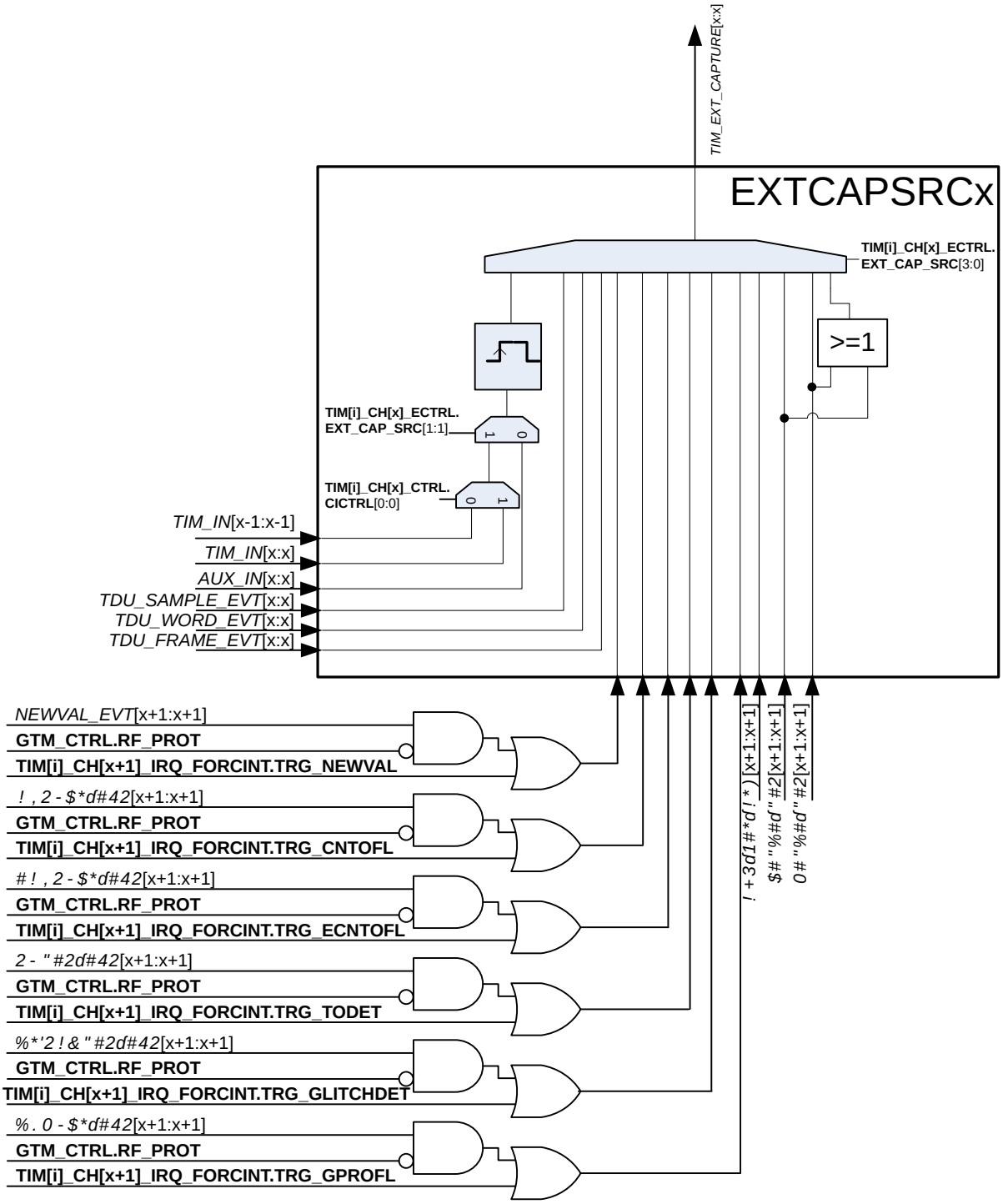
It is possible to observe for all channels of one instance by reading **TIM[i]_INP_VAL** the actual signal values of the following processing stages:

- *TIM_IN* [7:0] signals after TIM input synchronization
- *F_IN* [7:0] signals after TIM INPUTSRC selection (input to FLT)
- *F_OUT* [7:0] signals after TIM filter functionality (output of FLT)

13.1.3 External Capture Source Selection EXTCAPSRCx

Each channel can operate on an external capture signal *TIM_EXT_CAPTURE* . The source to use for this signal can be configured by the bit field **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** .

Figure 44 EXTCAPSRC Block Diagram



A pulse generation for each rising edge of the selected input signal **TIM[i]_INP_VAL.TIM_IN[x]** , **TIM[i]_INP_VAL.TIM_IN[x−1]** and *AUX_IN* [x:x] is applied. These pulses can be selected and used as *TIM_EXT_CAPTURE* [x:x] signal.

Alternatively, one of the events *TDU_SAMPLE_EVT* [x:x], *TDU_WORD_EVT* [x:x], *TDU_FRAME_EVT* [x:x] generated in the TDU can be selected.

Furthermore, one of the six interrupt sources of channel x+1 can be selected. Alternatively, they can be issued by a soft trigger using the corresponding bits in the register **TIM[i]_CH[x+1]_IRQ_FORCINT** . The pulses on **TIM[i]_CH[x+1]_IRQ_FORCINT** are allowed to pass through only when **GTM_CTRL.RF_PROT** = 0.

Additional sources of channel x+1 can be selected (rise/ fall/ both input edges or CMU clock resolution in use in channel x+1).

13.2 TIM Filter Functionality (FLT)

13.2.1 Overview

The TIM sub-module provides a configurable filter mechanism for each input signal. These filter mechanisms are provided inside the FLT sub-unit.

FLT architecture is shown in Figure *45 "FLT Architecture"* .

The filter includes a clock synchronization unit (CSU), an edge detection unit (EDU), and a filter counter associated with the filter unit (FLTU).

The CSU synchronizes the incoming signal *F_IN* with the selected filter clock frequency, which is controlled with the bit field **TIM[i]_CH[x]_CTRL.FLT_CNT_FRQ** .

The synchronized input signal *F_IN_SYNC* is used for further processing within the filter.

Within the FLT submodule there are two cluster clock cycle delays between *F_IN* and *F_OUT* , one in the CSU sub-block and one in the output signal generation sub-block (see *45 "FLT Architecture"*).

Note: Glitches with a duration go less than the selected CMU clock period is lost.

The filter modes can be applied individually to the falling and rising edges of an input signal. The following filter modes are available:

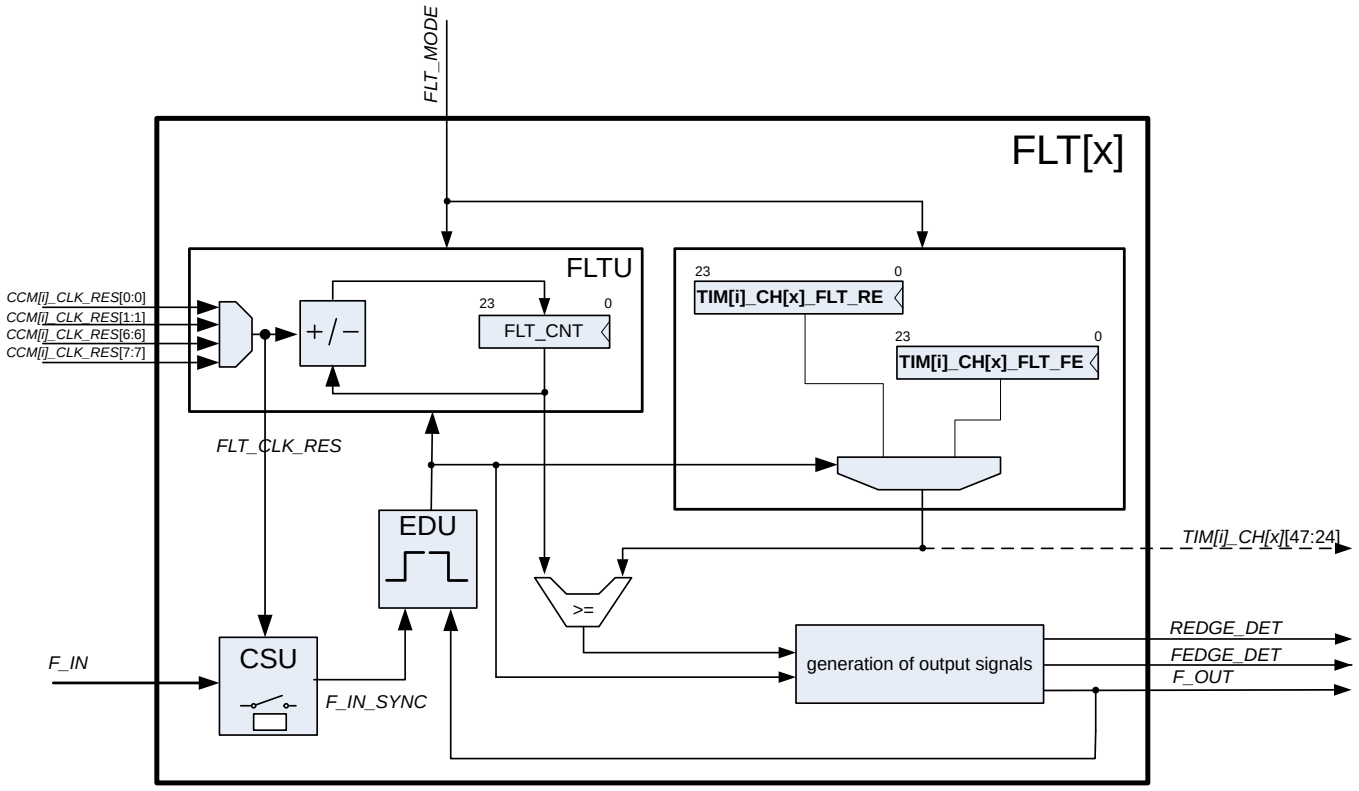
- immediate edge propagation mode,

- individual deglitch time mode (up/down counter), and

- individual deglitch time mode (hold counter).

- individual deglitch time mode (reset counter).

Figure 45 FLT Architecture



The filter parameters (deglitch and acceptance time) for the rising and falling edge can be configured inside the two filter parameter registers **TIM[i]_CH[x]_FLT_RE** (rising edge) and **TIM[i]_CH[x]_FLT_FE** (falling edge). The exact meaning of the parameter depends on the filter mode.

However the delay time T of both filter parameters **TIM[i]_CH[x]_FLT_FE** , **TIM[i]_CH[x]_FLT_RE** can always be determined by:
T=(**TIM[i]_CH[x]_FLT_RE.FLT_RE** +1)*$T_{FLT\_CLK\_RES}$,
T=(**TIM[i]_CH[x]_FLT_FE.FLT_FE** +1)*$T_{FLT\_CLK\_RES}$,
Whereas $T_{FLT\_CLK\_RES}$ is the clock period of the selected CMU clock signal in bit field **TIM[i]_CH[x]_CTRL.FLT_CNT_FRQ** of register **TIM[i]_CH[x]_CTRL** .

When a glitch is detected on an input signal, a pulse with the length of one cluster clock on the interrupt event signal *GLITCHDET_IRQ* is raised and a status flag **TIM[i]_CH[x]_IRQ_NOTIFY.GLITCHDET** is set.

Table *13 "Filter Parameter Summary for the Different Filter Modes"* gives an overview of the meanings for the registers **TIM[i]_CH[x]_FLT_RE** and **TIM[i]_CH[x]_FLT_FE** . In the individual deglitch time modes, the actual filter threshold for a detected regular edge is provided on the *TIM[i]_CH[x]* [47:24] output line. In the case of immediate edge propagation mode, a value of zero is provided on the *TIM[i]_CH[x]* [47:24] output line.

The *TIM[i]_CH[x]* [47:24] output line is used by the MAP sub-module for further processing (please see chapter *20 "TIM0 Input Mapping Module (MAP)"*).

Table 13 Filter Parameter Summary for the Different Filter Modes

Filter mode	Meaning of TIM[i]_CH[x]_FLT_RE	Meaning of TIM[i]_CH[x]_FLT_FE
Immediate edge propagation	Acceptance time for rising edge	Acceptance time for falling edge
Individual deglitch time (up/down counter)	Deglitch time for rising edge	Deglitch time for falling edge
Individual deglitch time (hold counter)	Deglitch time for rising edge	Deglitch time for falling edge
Individual deglitch time (reset counter)	Deglitch time for rising edge	Deglitch time for falling edge

A counter FLT_CNT is used to measure the glitch and acceptance times.

The frequency of the FLT_CNT counter is configurable in bit field **TIM[i]_CH[x]_CTRL.FLT_CNT_FRQ** .

The counter FLT_CNT is operating on *FLT_CLK_RES* clock resolution. The source for *FLT_CLK_RES* can be selected with **TIM[i]_CH[x]_CTRL.FLT_CNT_FRQ** from *CCM[i]_CLK_RES* [0:0], *CCM[i]_CLK_RES* [1:1], *CCM[i]_CLK_RES* [6:6] or the *CCM[i]_CLK_RES* [7:7] signals which were provided by the CCM[i] module.

The counter FLT_CNT and the registers **TIM[i]_CH[x]_FLT_FE** and **TIM[i]_CH[x]_FLT_RE** are 24-bit wide. For example, when the resolution of the *CCM[i]_CLK_RES* [0:0] signal is 50ns this allows maximal deglitch and acceptance times of about 838ms for the filter.

13.2.2 TIM Filter Modes

13.2.2.1 Immediate Edge Propagation Mode

In immediate edge propagation mode after detection of an edge, the new signal level on *F_IN_SYNC* is propagated to *F_OUT* with a delay of one $T_{FLT\_CLK\_RES}$ period and the new signal level remains unchanged until the configured acceptance time expires.

For each edge type the acceptance time can be specified separately in the **TIM[i]_CH[x]_FLT_RE** and **TIM[i]_CH[x]_FLT_FE** registers.

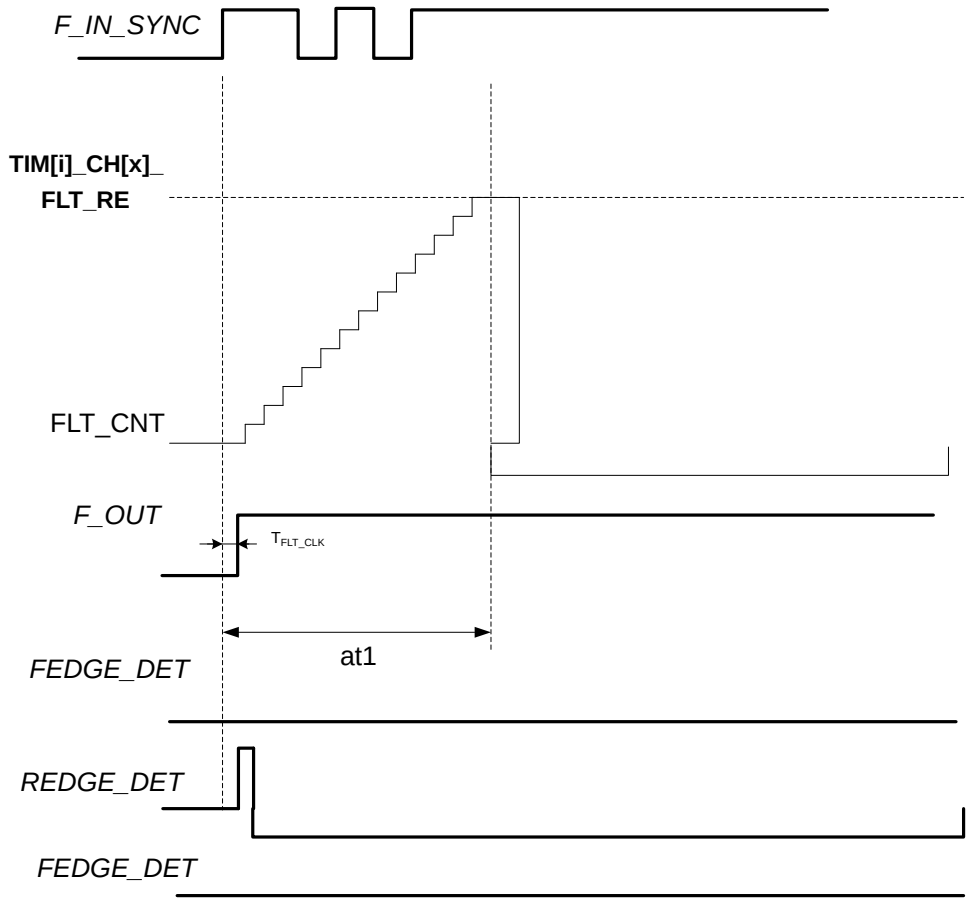
Each signal change on the input *F_IN_SYNC* during the duration of the acceptance time has no effect on the output signal level *F_OUT* of the filter but it sets the glitch **TIM[i]_CH[x]_IRQ_NOTIFY.GLITCHDET** bit.

After it expires an acceptance time the input signal *F_IN_SYNC* is observed and on signal level change the filter raises a new detected edge and the new signal level is propagated to *F_OUT* .

Independent of a signal level change the value of *F_OUT* is always set to *F_IN_SYNC* , when the acceptance time expires (see also Figure *47 "Immediate Edge Propagation Mode in the Case of a Rising and Falling Edge"*).

Figure *46 "Immediate Edge Propagation Mode in the Case of a Rising Edge"* shows an example for the immediate edge propagation mode, in the case of rising edge detection. Both, the signal before filtering (*F_IN*) and after filtering (*F_OUT*) are shown. The acceptance time at1 is specified in the register **TIM[i]_CH[x]_FLT_RE** .

Figure 46 Immediate Edge Propagation Mode in the Case of a Rising Edge

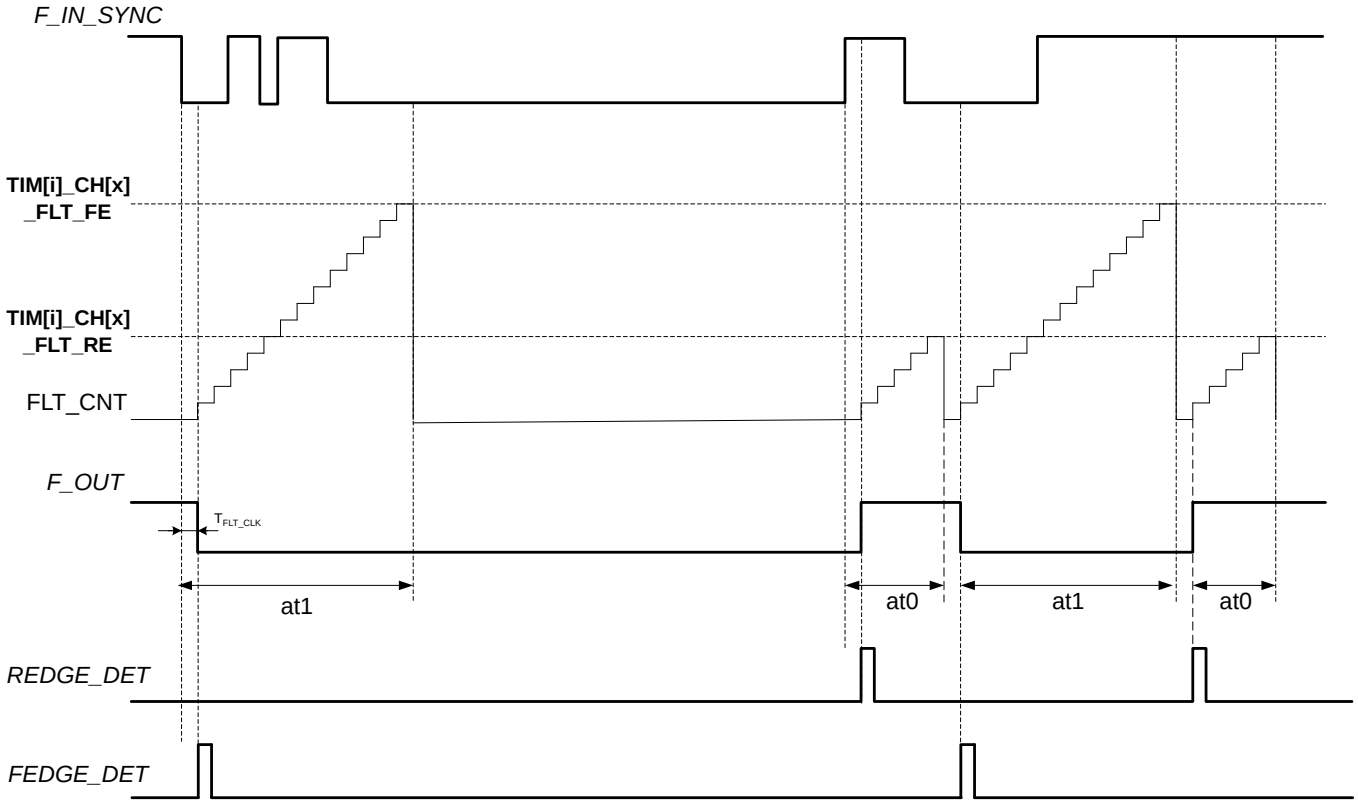


In immediate edge propagation mode, the glitch measurement mechanism is not applied to the edge detection. Detected edges on *F_IN_SYNC* are transferred directly to *F_OUT* .

The counter FLT_CNT is incremented until the acceptance time threshold is reached.

Figure *47 "Immediate Edge Propagation Mode in the Case of a Rising and Falling Edge"* shows a more complex example of the TIM filter, in which both, rising and falling edges are configured in immediate edge propagation mode.

Figure 47 Immediate Edge Propagation Mode in the Case of a Rising and Falling Edge



If the FLT_CNT has reached the acceptance time for a specific signal edge and the signal *F_IN_SYNC* has already changed to the opposite level of *F_OUT* , the opposite signal level is set to *F_OUT* and the acceptance time measurement is started immediately. Figure *47 "Immediate Edge Propagation Mode in the Case of a Rising and Falling Edge"* shows this scenario at the detection of the first rising edge and the second falling edge.

13.2.2.2 Individual Deglitch Time Mode (Up/Down Counter)

In individual deglitch time mode (up/down counter) each edge of an input signal can be filtered with an individual deglitch threshold filter value mentioned in the registers **TIM[i]_CH[x]_FLT_RE** and **TIM[i]_CH[x]_FLT_FE** , respectively.

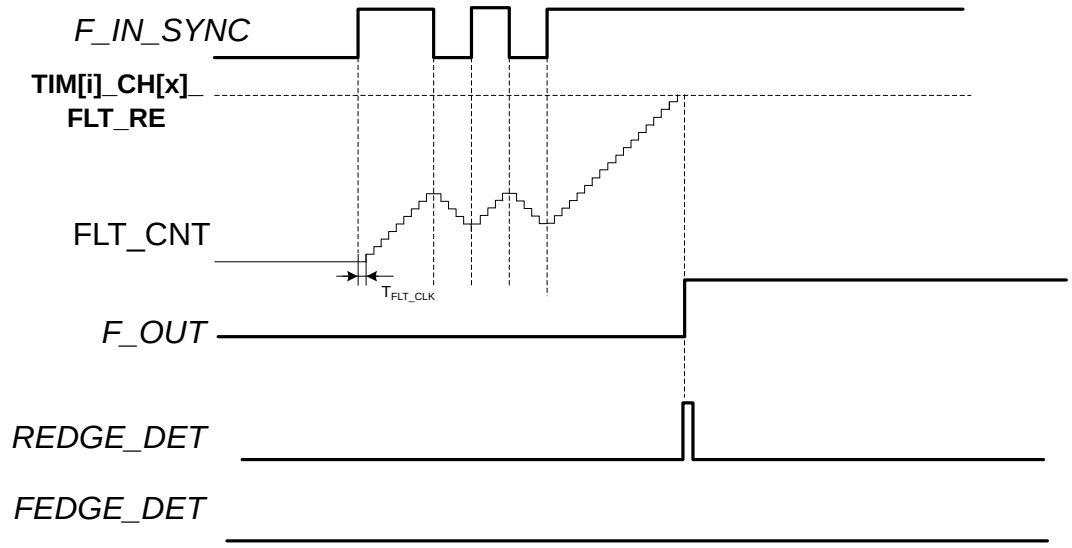
The filter counter register FLT_CNT is incremented when the signal level on *F_IN_SYNC* is unequal to the signal level on *F_OUT* and decremented if *F_IN_SYNC* equals *F_OUT* .

After FLT_CNT has reached a value of zero during decrementation the counter is stopped immediately.

If a glitch is detected a glitch detection bit **TIM[i]_CH[x]_IRQ_NOTIFY.GLITCHDET** is set.

The detected edge signal together with the new signal level is propagated to *F_OUT* after the individual deglitch threshold is reached. Figure *48 "Individual Deglitch Time Mode (Up/Down Counter) in the Case of a Rising Edge"* shows the behavior of the filter in individual deglitch time (up/down counter) mode in the case of the rising edge detection.

Figure 48 Individual Deglitch Time Mode (Up/Down Counter) in the Case of a Rising Edge



13.2.2.3 Individual Deglitch Time Mode (Hold Counter)

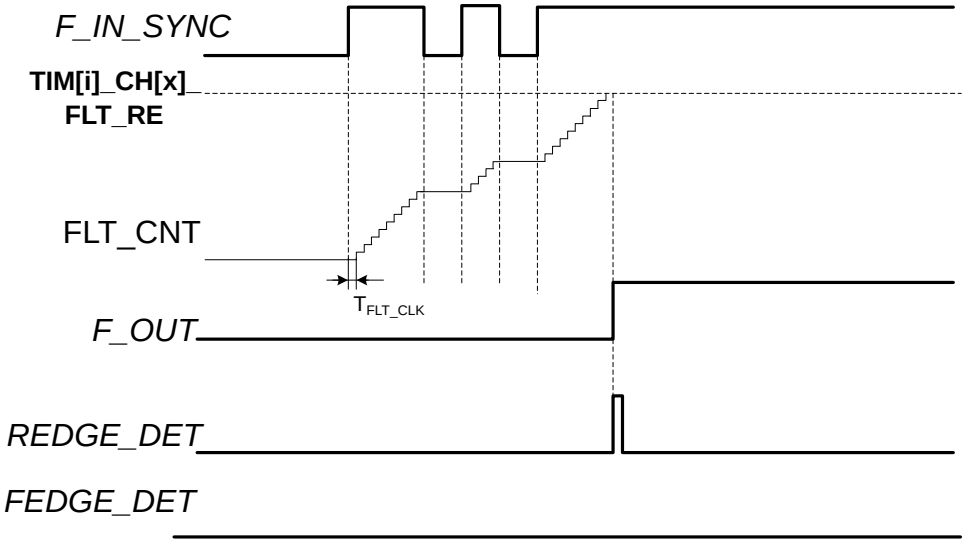
In individual deglitch time mode (hold counter) each edge of an input signal can be filtered with an individual deglitch threshold filter value mentioned in the registers **TIM[i]_CH[x]_FLT_RE** and **TIM[i]_CH[x]_FLT_FE** , respectively.

The filter counter register FLT_CNT is incremented when the signal level on *F_IN_SYNC* is unequal to the signal level on *F_OUT* and the counter value of FLT_CNT is hold if *F_IN* equals *F_OUT* .

If a glitch is detected the glitch detection bit **TIM[i]_CH[x]_IRQ_NOTIFY.GLITCHDET** is set.

The detected edge signal together with the new signal level is propagated to *F_OUT* after the individual deglitch threshold is reached. Figure *49 "Individual Deglitch Time Mode (hold counter) in the Case of a Rising Edge"* shows the behavior of the filter in individual deglitch time (hold counter) mode in the case of the rising edge detection.

Figure 49 Individual Deglitch Time Mode (hold counter) in the Case of a Rising Edge



13.2.2.4 Individual Deglitch Time Mode (Reset Counter)

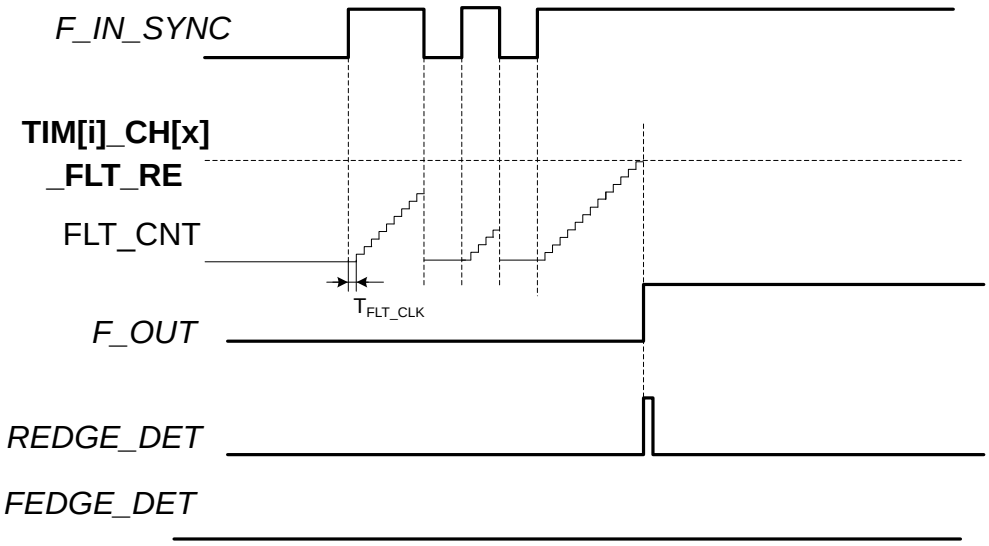
In individual deglitch time mode (reset counter) each edge of an input signal can be filtered with an individual deglitch threshold filter value mentioned in the registers **TIM[i]_CH[x]_FLT_RE** and **TIM[i]_CH[x]_FLT_FE** , respectively.

The filter counter register FLT_CNT is incremented when the signal level on *F_IN_SYNC* is unequal to the signal level on *F_OUT* and the counter value of FLT_CNT is reset to 0x000000 if *F_IN* equals *F_OUT* .

If a glitch is detected the glitch detection bit **TIM[i]_CH[x]_IRQ_NOTIFY.GLITCHDET** is set.

The detected edge signal together with the new signal level is propagated to *F_OUT* after the individual deglitch threshold is reached. Figure *50 "Individual Deglitch Time Mode (Reset Counter) in the Case of a Rising Edge"* shows the behavior of the filter in individual deglitch time (reset counter) mode in the case of the rising edge detection.

Figure 50 Individual Deglitch Time Mode (Reset Counter) in the Case of a Rising Edge



13.2.2.5 Immediate Edge Propagation and Individual Deglitch Mode

As already mentioned, the four different filter modes can be applied individually to each edge of the measured signal.

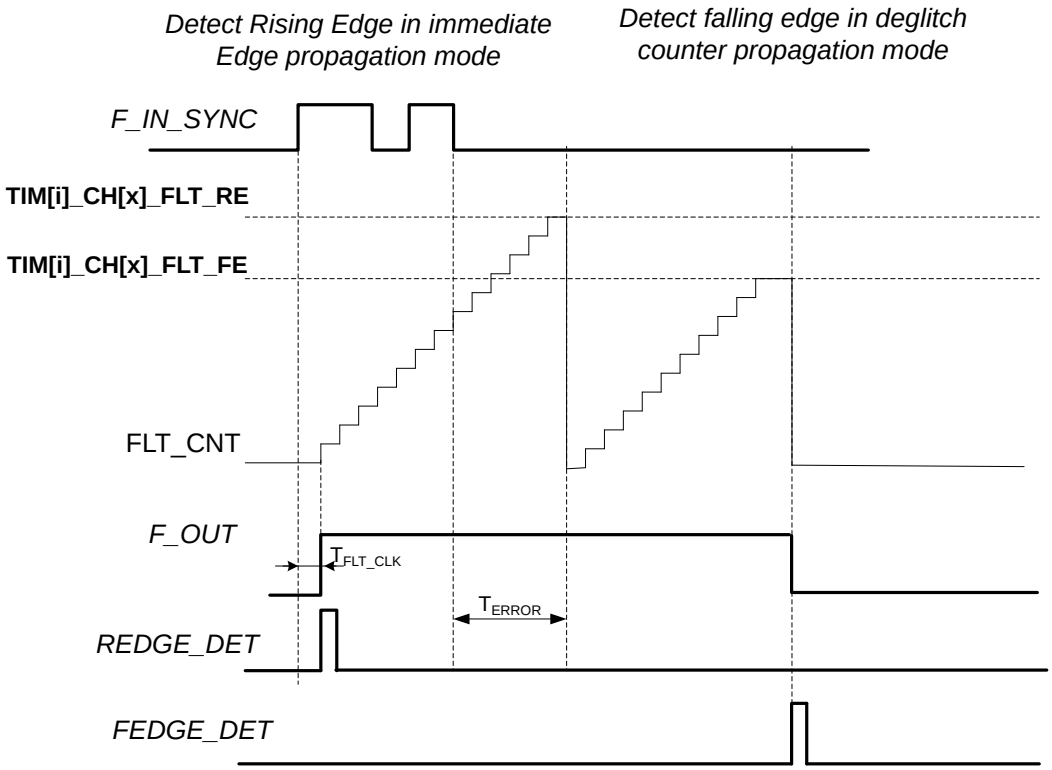
However, if one edge is configured with immediate edge propagation and the other edge with an individual deglitch mode (whether up/down counter, hold counter or reset counter) a special consideration has to be applied.

Assume that the rising edge is configured for immediate edge propagation and the falling edge with individual deglitch mode (up/down counter) as shown in Figure *51 "Mixed Mode Measurement"* .

If the falling edge of the incoming signal already occurs during the measuring of the acceptance time of the rising edge, the measurement of the deglitch time on the falling edge is started with delay, but immediately after the acceptance time measurement phase of the rising edge has finished.

Consequently, the deglitch counter cannot measure the time T_{ERROR}, as shown in Figure *51 "Mixed Mode Measurement"* .

Figure 51 Mixed Mode Measurement



13.2.3 TIM Filter Reconfiguration

If **TIM[i]_CH[x]_CTRL.FLT_EN** =1 a change of **TIM[i]_CH[x]_FLT_RE** or **TIM[i]_CH[x]_FLT_FE** will take place immediately.

If **TIM[i]_CH[x]_CTRL.FLT_EN** =1 a change of **TIM[i]_CH[x]_CTRL.FLT_MODE_RE** or **TIM[i]_CH[x]_CTRL.FLT_MODE_FE** will be used with the next occurring corresponding edge. If the mode is changed while the filter unit is processing a certain mode, it will end this edge filtering in the mode as started.

If **TIM[i]_CH[x]_CTRL.FLT_EN** =1 a change of **TIM[i]_CH[x]_CTRL.FLT_CTR_RE** , **TIM[i]_CH[x]_CTRL.FLT_CTR_FE** , **TIM[i]_CH[x]_ECTRL.-EFLT_CTR_RE** or **TIM[i]_CH[x]_ECTRL.EFLT_CTR_FE** will take place immediately.

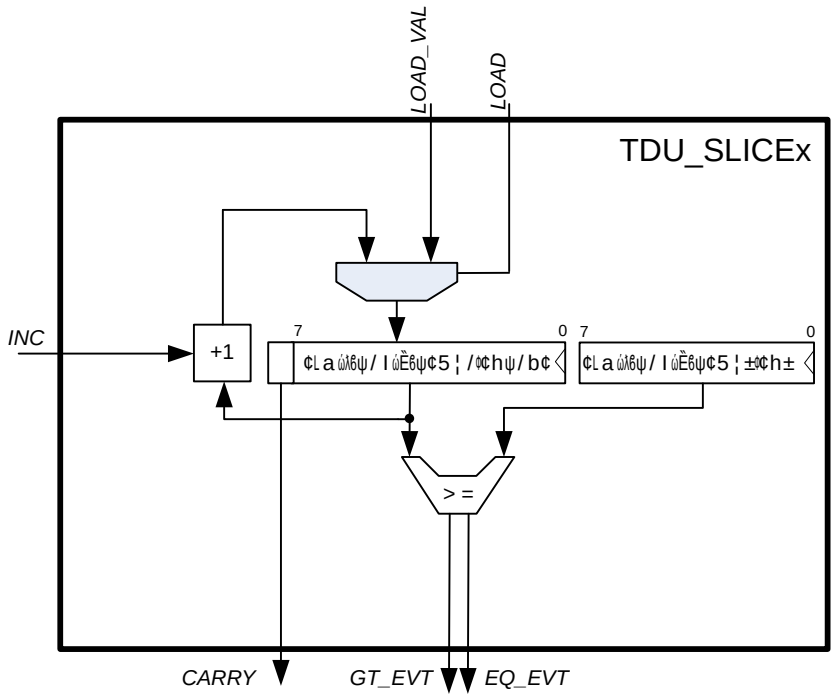
13.3 Timeout Detection Unit (TDU)

The Timeout Detection Unit (TDU) is responsible for timeout detection of the TIM input signals.

Each channel of the TIM sub-module has its own Timeout Detection Unit (TDU) where a timeout event can be set up on the filtered input signal of the corresponding channel.

In each timeout unit exist 3x 8 bit counter/comparator slices. A counter/comparator slice is shown below. The counter **TIM[i]_CH[x]_-TDUC.TO_CNT** will increment by signal *INC* . The counter can be loaded with the value *LOAD_VAL* if *LOAD* = 1. *GT_EVT* will be 1 if **TIM[i]_CH[x]_TDUC.TO_CNT** > **TIM[i]_CH[x]_TDUV.TOV** is fulfilled. *EQ_EVT* will be 1 if **TIM[i]_CH[x]_TDUC.TO_CNT** = **TIM[i]_CH[x]_TDU-V.TOV** is fulfilled.

Figure 52 Counter/Comparator Slice



The counter/comparator slices can be cascaded depending on the application needs to operate as:
3x 8 bit counter
1x 16 bit counter and 1x 8 bit counter
1x 24 bit counter
2x 8 bit counter.
This allows the user to use the functions
timeout on input signals
local CMU clock prescaler 8-bit
trigger event generation 8-bit (external capture, TIM_TODET_IRQ)
in parallel.
With usage of the 3x 8 bit counter it is possible to define different timeout values for the 2 signal levels.

Following table shows which functions can be used in parallel.

Table 14 Used Parallel Functions

Counter type	Timeout functionality	Generate local TIM CMU clk	Source for external capture to previous channel	Source for *TIM_TODET_IRQ*
24-bit	24-bit	no	*TDU_TIMEOUT_EVT* , *TDU_SAMPLE_EVT*	*TDU_TIMEOUT_EVT* , *TDU_SAMPLE_EVT*
1 x 8 bit 1 x 16 bit	16-bit local clk *TDU_SAMPLE_EVT* usable	yes	*TDU_TIMEOUT_EVT* , *TDU_FRAME_EVT* , *TDU_SAMPLE_EVT*	*TDU_TIMEOUT_EVT* , *TDU_FRAME_EVT* , *TDU_SAMPLE_EVT*
3x 8 bit	8-bit local clk *TDU_SAMPLE_EVT* usable	yes	*TDU_TIMEOUT_EVT* , *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT*	*TDU_TIMEOUT_EVT* , *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT*
3x 8 bit	no	yes	*TDU_TIMEOUT_EVT* , *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT*	*TDU_TIMEOUT_EVT* , *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT*
2x 8 bit	no	no	*TDU_TIMEOUT_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT*	*TDU_TIMEOUT_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT*

Next table shows which of the available 8-bit resources are cascaded with a chosen **TIM[i]_CH[x]_TDUV.SLICING** .

Table 15 Which of the Available 8 Bit Resources are Cascaded with a Chosen SLICING

Counter type	Counters count on	Counter resource generates	CLK selection
24-bit	CNT on TCS	**TIM[i]_CH[x]_CNT.CNT** = **TIM[i]_CH[x]_TDUC.TO_CNT2** & **TIM[i]_CH[x]_TDUC.TO_CNT1** & **TIM[i]_CH[x]_TDUC.TO_CNT** ; TCMP = **TIM[i]_CH[x]_TDUV.TOV2** & **TIM[i]_CH[x]_TDUV.TOV1** & **TIM[i]_CH[x]_TDUV.TOV** ; **TIM[i]_CH[x]_CNT.CNT** >= TCMP generates *TDU_SAMPLE_EVT* *TDU_TIMEOUT_EVT* = *TDU_SAMPLE_EVT* ; *TDU_FRAME_EVT* = 0 *TDU_WORD_EVT* = 0	TCS selected
3x 8 bit	**TIM[i]_CH[x]_TDUC.TO_CNT2** on TCS; **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_SAMPLE_EVT* **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_WORD_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT2** >= **TIM[i]_CH[x]_TDUV.TOV2** generates *TDU_SAMPLE_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT** >= **TIM[i]_CH[x]_TDUV.TOV** generates *TDU_WORD_EVT* **TIM[i]_CH[x]_TDUC.TO_CNT1** >= **TIM[i]_CH[x]_TDUV.TOV1** generates *TDU_FRAME_EVT* *TDU_TIMEOUT_EVT* = *TDU_WORD_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT2** : **TIM[i]_CH[x]_TDUV.TCS** selected; **TIM[i]_CH[x]_TDUC.TO_CNT** : *TDU_SAMPLE_EVT* selected with **TIM[i]_CH[x]_TDUV.TCS_USE_SAMPLE_EVT** =1 **TIM[i]_CH[x]_TDUC.TO_CNT1** : *TDU_WORD_EVT* selected with **TIM[i]_CH[x]_TDUV.TDU_SAME_CNT_CLK** =0
3x 8 bit	**TIM[i]_CH[x]_TDUC.TO_CNT2** on TCS; **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_SAMPLE_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_SAMPLE_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT2** >= **TIM[i]_CH[x]_TDUV.TOV2** generates *TDU_SAMPLE_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT** >= **TIM[i]_CH[x]_TDUV.TOV** generates *TDU_WORD_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT1** >= **TIM[i]_CH[x]_TDUV.TOV1** generates *TDU_FRAME_EVT* ; *TDU_TIMEOUT_EVT* = *TDU_WORD_EVT* or *TDU_FRAME_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT2** : **TIM[i]_CH[x]_TDUV.TCS** selected; **TIM[i]_CH[x]_TDUC.TO_CNT** : *TDU_SAMPLE_EVT* selected with **TIM[i]_CH[x]_TDUV.TCS_USE_SAMPLE_EVT** =1; **TIM[i]_CH[x]_TDUC.TO_CNT1** : *TDU_SAMPLE_EVT* selected with **TIM[i]_CH[x]_TDUV.TDU_SAME_CNT_CLK** =1

Counter type	Counters count on	Counter resource generates	CLK selection
3x 8 bit	**TIM[i]_CH[x]_TDUC.TO_CNT2** on TCS; **TIM[i]_CH[x]_TDUC.TO_CNT** on TCS; **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_WORD_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT2** >= **TIM[i]_CH[x]_TDUV.TOV2** generates *TDU_SAMPLE_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT** >= **TIM[i]_CH[x]_TDUV.TOV** generates *TDU_WORD_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT1** >= **TIM[i]_CH[x]_TDUV.TOV1** generates *TDU_FRAME_EVT* ; *TDU_TIMEOUT_EVT* = *TDU_WORD_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT2** : **TIM[i]_CH[x]_TDUV.TCS** selected; **TIM[i]_CH[x]_TDUC.TO_CNT** : **TIM[i]_CH[x]_TDUV.TCS** selected with **TIM[i]_CH[x]_TDUV.TCS_USE_SAMPLE_EVT** =0; **TIM[i]_CH[x]_TDUC.TO_CNT1** : *TDU_WORD_EVT* selected with **TIM[i]_CH[x]_TDUV.TDU_SAME_CNT_CLK** =0
3x 8 bit	**TIM[i]_CH[x]_TDUC.TO_CNT2** on TCS; **TIM[i]_CH[x]_TDUC.TO_CNT** on TCS; **TIM[i]_CH[x]_TDUC.TO_CNT1** on TCS	**TIM[i]_CH[x]_TDUC.TO_CNT2** >= **TIM[i]_CH[x]_TDUV.TOV2** generates *TDU_SAMPLE_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT** >= **TIM[i]_CH[x]_TDUV.TOV** generates *TDU_WORD_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT1** >= **TIM[i]_CH[x]_TDUV.TOV1** generates *TDU_FRAME_EVT* ; *TDU_TIMEOUT_EVT* = *TDU_WORD_EVT* or *TDU_FRAME_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT2** : **TIM[i]_CH[x]_TDUV.TCS** selected; **TIM[i]_CH[x]_TDUC.TO_CNT** : **TIM[i]_CH[x]_TDUV.TCS** selected with **TIM[i]_CH[x]_TDUV.TCS_USE_SAMPLE_EVT** =0; **TIM[i]_CH[x]_TDUC.TO_CNT1** : **TIM[i]_CH[x]_TDUV.TCS** selected with **TIM[i]_CH[x]_TDUV.TDU_SAME_CNT_CLK** =1
2x 8 bit	**TIM[i]_CH[x]_TDUC.TO_CNT** on TCS; **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_WORD_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT** >= **TIM[i]_CH[x]_TDUV.TOV** generates *TDU_WORD_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT1** >= **TIM[i]_CH[x]_TDUV.TOV1** generates *TDU_FRAME_EVT* ; *TDU_TIMEOUT_EVT* = *TDU_WORD_EVT* ; *TDU_SAMPLE_EVT* = 0	**TIM[i]_CH[x]_TDUC.TO_CNT** : TIM[i]_CH[x]_TDUV.TCS selected; **TIM[i]_CH[x]_TDUC.TO_CNT1** : *TDU_WORD_EVT* selected with TIM[i]_CH[x]_TDUV.TDU_SAME_CNT_CLK =0
2x 8 bit	**TIM[i]_CH[x]_TDUC.TO_CNT** on TCS; **TIM[i]_CH[x]_TDUC.TO_CNT1** on TCS	**TIM[i]_CH[x]_TDUC.TO_CNT** >= **TIM[i]_CH[x]_TDUV.TOV** generates *TDU_WORD_EVT* ; **TIM[i]_CH[x]_TDUC.TO_CNT1** >= **TIM[i]_CH[x]_TDUV.TOV1** generates *TDU_FRAME_EVT* ; *TDU_TIMEOUT_EVT* = *TDU_WORD_EVT* ; *TDU_SAMPLE_EVT* = 0	**TIM[i]_CH[x]_TDUC.TO_CNT** : **TIM[i]_CH[x]_TDUV.TCS** selected; **TIM[i]_CH[x]_TDUC.TO_CNT1** : **TIM[i]_CH[x]_TDUV.TCS** selected with **TIM[i]_CH[x]_TDUV.TDU_SAME_CNT_CLK** =1
1 x 8 bit 1 x 16 bit	**TIM[i]_CH[x]_TDUC.TO_CNT2** on TCS; **TIM[i]_CH[x]_CNT.CNT** on TCS	**TIM[i]_CH[x]_TDUC.TO_CNT2** >= **TIM[i]_CH[x]_TDUV.TOV2** generates *TDU_SAMPLE_EVT* ; CNT = **TIM[i]_CH[x]_TDUC.TO_CNT1** & **TIM[i]_CH[x]_TDUC.TO_CNT** ; TCMP = **TIM[i]_CH[x]_TDUV.TOV1** & **TIM[i]_CH[x]_TDUV.TOV** ; CNT >= TCMP generates *TDU_FRAME_EVT* ; *TDU_TIMEOUT_EVT* = *TDU_FRAME_EVT* ; *TDU_WORD_EVT* = 0	**TIM[i]_CH[x]_TDUC.TO_CNT2** : **TIM[i]_CH[x]_TDUV.TCS** selected; **TIM[i]_CH[x]_CNT.CNT** : **TIM[i]_CH[x]_TDUV.TCS** selected with **TIM[i]_CH[x]_TDUV.TCS_USE_SAMPLE_EVT** =0
1 x 8 bit 1 x 16 bit	**TIM[i]_CH[x]_TDUC.TO_CNT2** on TCS; **TIM[i]_CH[x]_CNT.CNT** on *TDU_SAMPLE_EVT*	**TIM[i]_CH[x]_TDUC.TO_CNT2** >= **TIM[i]_CH[x]_TDUV.TOV2** generates *TDU_SAMPLE_EVT* ; CNT = **TIM[i]_CH[x]_TDUC.TO_CNT1** & **TIM[i]_CH[x]_TDUC.TO_CNT** ; TCMP = **TIM[i]_CH[x]_TDUV.TOV1** & **TIM[i]_CH[x]_TDUV.TOV** ; CNT >= TCMP generates *TDU_FRAME_EVT* ; *TDU_TIMEOUT_EVT* = *TDU_FRAME_EVT* ; *TDU_WORD_EVT* = 0	**TIM[i]_CH[x]_TDUC.TO_CNT2** : **TIM[i]_CH[x]_TDUV.TCS** selected; **TIM[i]_CH[x]_CNT.CNT** : *TDU_SAMPLE_EVT* selected with **TIM[i]_CH[x]_TDUV.TCS_USE_SAMPLE_EVT** =1

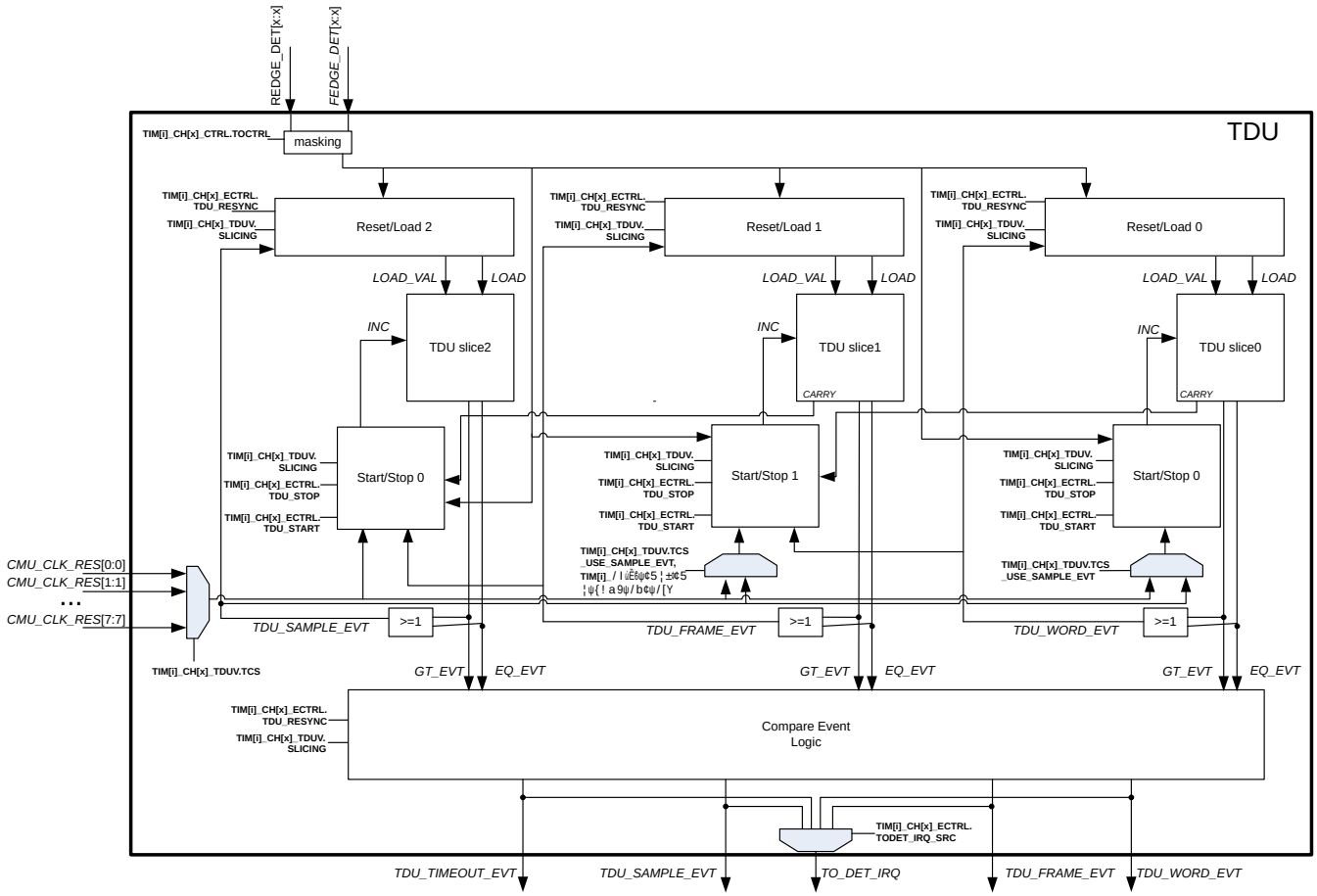
Based on a chosen counter configuration by **TIM[i]_CH[x]_TDUV.SLICING** it is possible to control the start behavior of the counters by **TIM[i]_CH[x]_ECTRL.TDU_START** in multiple ways. In addition, the stopping of the counters can be controlled by **TIM[i]_CH[x]_ECTRL.TD-**

U_STOP . Depending on the application needs it can be decided how the individual counter slices can be reset/reloaded by the configuration field **TIM[i]_CH[x]_ECTRL.TDU_RESYNC** .

Depending on the counter configuration, up to 4 internal compare events *TDU_TIMEOUT_EVT* , *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT* can be generated out of the 3 comparator slices. **TIM[i]_CH[x]_ECTRL.TODET_IRQ_SRC** can choose, which event to be used as *TIM_TODET_IRQ* [x:x] signal which will be accessible by the **TIM[i]_CH[x]_IRQ_NOTIFY.TODET** bit.

The TDU architecture is shown in *53 "Architecture of the TDU Sub-Unit"* .

Figure 53 Architecture of the TDU Sub-Unit



Each TDU_slice has its own start/stop control, based on the chosen configuration it will decide if the counter inside the TDU slice will increment on the resolution of the applied clock/event. The reset/load control decides based on the configuration settings and the compare result of the TDU slices those the counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** , **TIM[i]_CH[x]_TDUC.TO_CNT2** have to be reloaded. Depending on the chosen counter/compare configuration the compare event logic will generate based on the compare results of the 3 TDU slices and the chosen resolution the events *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT* .

The primary resolution on which the TDU is working can be specified with the bit field **TIM[i]_CH[x]_TDUV.TCS** of the register **TIM[i]_CH[x]_TDUV** . The corresponding input signal *CCM[i]_CLK_RES* [x:x] will be used to clock the TDU. The individual timeout/counter values have to be specified in number of ticks of the selected input clock signal in the fields **TIM[i]_CH[x]_TDUV.TOV** , **TIM[i]_CH[x]_TDUV.TOV1** , **TIM[i]_CH[x]_TDUV.TOV2** of the timeout value register **TIM[i]_CH[x]_TDUV** of the TIM channel x.

In case of cascading the bit slices by usage of **TIM[i]_CH[x]_TDUV.SLICING** and **TIM[i]_CH[x]_TDUV.TCS_USE_SAMPLE_EVT** and **TIM[i]_CH[x]_TDUV.TDU_SAME_CNT_CLK** the resolution for counting can be switched to the events *TDU_SAMPLE_EVT* or *TDU_WORD_EVT* . For more details see table above.

The counter compare units start operation on occurrence of the first "start event" configured by **TIM[i]_CH[x]_ECTRL.TDU_START** . They continue their operation until the first "stop event" configured by **TIM[i]_CH[x]_ECTRL.TDU_STOP** occurs.

In case of occurrence of a start event and a compare/count resolution event in the same clock cycle, the counters will increment or reload/reset based on **TIM[i]_CH[x]_ECTRL.TDU_RESYNC** immediately. No *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT* will be generated.

In case of occurrence of a stop event the counters will not change their values. In case of occurrence of a stop event and a compare/count resolution event in the same clock cycle the corresponding events *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT* will be generated.

In case of occurrence of a start event and a stop event in the same clock cycle the counters will not change their values. No *TDU_SAMPLE_EVT* , *TDU_WORD_EVT* , *TDU_FRAME_EVT* will be generated.

The function of the timeout unit (configured to **TIM[i]_CH[x]_ECTRL.TDU_RESYNC** =0000, **TIM[i]_CH[x]_ECTRL.TDU_START** =000) can be started or stopped by setting/resetting the **TIM[i]_CH[x]_CTRL.TOCTRL** bit.

Timeout detection can be enabled to be sensitive to falling, rising or both edges of the input signal by writing the corresponding values to the bit field **TIM[i]_CH[x]_CTRL.TOCTRL** .

Note:

The used signal names *REDGE_DET* [x:x] and *FEDGE_DET* [x:x] are related to the local channel x or the previous channel x−1, depending on the value of the configuration bit field **TIM[i]_CH[x]_ECTRL.USE_PREV_TDU_IN** .

The TDU generates an interrupt signal *TIM_TODET_IRQ* [x:x] whenever a timeout is detected for an individual input signal and the **TIM[i]_CH[x]_IRQ_NOTIFY.TODET** bit.

In addition, when the ARU access is enabled with the **TIM[i]_CH[x]_CTRL.ARU_EN** bit inside the **TIM[i]_CH[x]_CTRL** register, the actual values stored inside the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are sent together with the last stored signal level to the ARU if a timeout event *TDU_TIMEOUT_EVT* occurs.

To signal that a timeout occurred, the signal *TO_DET* is set with every *TDU_TIMEOUT_EVT* . The *ARU_OUT* [50:50] signal is driven by *TO_DET* . The bit *ARU_OUT* [48:48] will be updated together with *TO_DET* to the signal level on which the timeout was detected. Timeout signaling with *ARU_OUT* [50:50] is only possible with **TIM[i]_CH[x]_ECTRL.TODET_IRQ_SRC** = 0000.

Thus, a destination could determine if a timeout occurred at the TIM input by evaluating *ARU_OUT* [50:50] .

Since the TIM channel still monitors its input pin although the timeout happened, a valid edge could occur at the input pin while the timeout information is still valid at the ARU. In that case, the new edge associated data is stored inside the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** , the GPR overflow detected bit is set together in the signal (*ARU_OUT* [49:49]) with the timeout bit (*ARU_OUT* [50:50]) and the values are marked as valid to the ARU.

The *TO_DET* signal is cleared, when a successful ARU write access by the TIM channel took place.

The *ARU_OUT* [49:49] bit is cleared, when a successful ARU write access by the TIM channel took place.

When a valid edge initiates an ARU write access which has not ended while a new timeout occurs the GPR overflow detected bit (*ARU_OUT* [49:49]) is set. The bit *ARU_OUT* [48:48] will be updated to the level on which the timeout occurred.

When a timeout occurs and initiates an ARU write access which has not ended while a new timeout occurs the GPR overflow detected bit (*ARU_OUT* [49:49]) is not set.

The following table clarifies the meaning of the *ARU_OUT* [50:48] bits for valid data provided by a TIM channel:

Table 16 ARU_OUT[50:48] Bits for Valid Data Provided by a TIM Channel

ARU_OUT [50:50]	*ARU_OUT* [49:49]	*ARU_OUT* [48:48]	Description
0	0	SL	Valid edge detected
0	1	SL	Input edge overwritten by subsequent edge
1	0	SL	Timeout *TO_DET* detected without valid edge
1	1	SL	Timeout *TO_DET* detected with subsequent valid edge detected

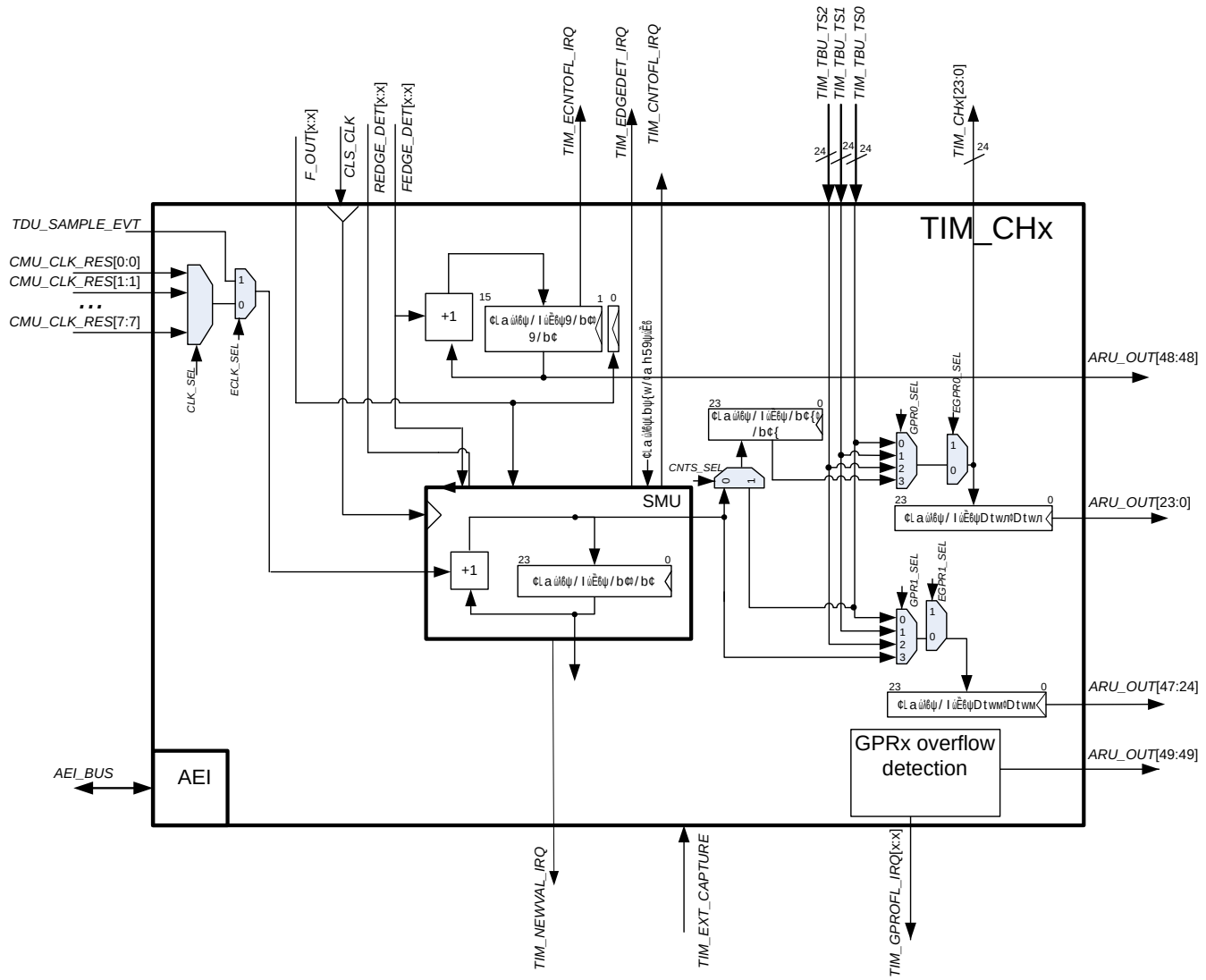
13.4 TIM Channel Architecture

13.4.1 Overview

Each TIM channel consists of an input edge counter **TIM[i]_CH[x]_ECNT** , a Signal Measurement Unit (SMU) with a counter **TIM[i]_CH[x]_CNT** , a counter shadow register **TIM[i]_CH[x]_CNTS** for SMU counter and two general purpose registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** for value storage.

The value **TIM[i]_CH[x]_TDUV.TOV** of the timeout register **TIM[i]_CH[x]_TDUV** is provided to TDU sub−unit of each individual channel for timeout measurement. The architecture of the TIM channel is depicted in Figure *54 "TIM Channel Architecture"* .

Figure 54 TIM Channel Architecture



Each TIM channel receives both input trigger signals *REDGE_DET* [x:x] and *FEDGE_DET* [x:x], generated by the corresponding filter module in order to signalize a detected echo of the input signal *F_IN* [x:x]. The signal *F_OUT* [x:x] shows the filtered signal of the channel's input signal *F_IN* [x:x].

Note:

In the following chapters for the different channel modes – expect the TBCM mode – the used signal names *F_OUT* [x:x], *REDGE_DET* [x:x] and *FEDGE_DET* [x:x] are related to the local channel x or the previous channel x–1, depending on the value of the configuration bit field **TIM[i]_CH[x]_ECTRL.USE_PREV_CH_IN** . In TBCM mode, the signals do not depend on the value of the configuration bit field **TIM[i]_CH[x]_ECTRL.USE_PREV_CH_IN** .

The edge counter **TIM[i]_CH[x]_ECNT** counts every incoming filtered edge (rising and falling). The counter value is uneven in case of detected rising and even in case of detected falling edge. Thus, the input signal level is part of the counter and can be obtained by bit 0 of **TIM[i]_CH[x]_ECNT** . (However, the actual counter implementation counts only falling edges on **TIM[i]_CH[x]_ECNT** [n:1] bits. It generates **TIM[i]_CH[x]_ECNT.ECNT** by composing the **TIM[i]_CH[x]_ECNT.ECNT** [n:1] bits with *F_OUT* [x:x] as bit 0).

Thus, the whole **TIM[i]_CH[x]_ECNT** counter value is always odd, when a positive edge was received and always even, when a negative edge was received.

The current **TIM[i]_CH[x]_ECNT** [7:0] register content is made visible on the bits 31 down to 24 of the registers **TIM[i]_CH[x]_GPR0** , **TIM[i]_CH[x]_GPR1** , and **TIM[i]_CH[x]_CNTS** . This allows the software to detect inconsistent read accesses to registers **TIM[i]_CH[x]_GPR0** , **TIM[i]_CH[x]_GPR1** , and **TIM[i]_CH[x]_CNTS** . However, the update strategy of these registers depends on the selected TIM modes, and thus the consistency check has to be adapted carefully.

It can be chosen with the bit field **TIM[i]_CH[x]_CTRL.FR_ECNT_OFL** when an overflow is signaled on **TIM[i]_CH[x]_IRQ_NOTIFY.ECNTOFL** . An **TIM[i]_CH[x]_ECNT** overflow can be signaled on 8-bit or full range resolution (wrap around 0xFF or 0xFFFF to 0x0 due to increment)

While reading the bit **TIM[i]_CH[x]_ECNT.ECNT** [0:0] shows the input signal value *F_OUT* [x:x] independent of the state (enabled / disabled) of the channel. If a channel gets disabled (OSM mode or resetting **TIM[i]_CH[x]_CTRL.TIM_EN**) the content of **TIM[i]_CH[x]_ECNT** will be frozen until a read of the register takes place. This read will reset the **TIM[i]_CH[x]_ECNT** counter. Continuing reads will show the input signal value in bit **TIM[i]_CH[x]_ECNT.ECNT** [0:0] again.

If a channel gets enabled (**TIM[i]_CH[x]_CTRL.TIM_EN** = 1), the registers **TIM[i]_CH[x]_ECNT.ECNT** [15:1], **TIM[i]_CH[x]_CNT** , **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are set to the initial values. In case an incoming signal edge occurs in the same cluster clock cycle as the channel enabling, the registers **TIM[i]_CH[x]_ECNT.ECNT** [15:1], **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are set to the initial value and the register **TIM[i]_CH[x]_CNT** is increased or a new value is shifted (in TSSM mode) based on the initial value.

For a clear re-enable procedure of the TIM channel it is recommended to re-enable the ARU routing (**TIM[i]_CH[x]_CTRL.ARU_EN**) in parallel.

When new data is written into **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** the **TIM[i]_CH[x]_IRQ_NOTIFY.NEWVAL** bit is set and depending on corresponding enable bit value the *TIM_NEWVAL_IRQ* [x:x] interrupt is raised.

Each TIM input channel has an ARU connection for providing data via the ARU to the other GTM sub-modules. The data provided to the ARU depends on the TIM channel mode and its corresponding adjustments (e.g. multiplexer configuration).

The bit **TIM[i]_CH[x]_CTRL.ARU_EN** of register **TIM[i]_CH[x]_CTRL** decides, whether the measurement results of registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are consumed by another sub-module via ARU (**TIM[i]_CH[x]_CTRL.ARU_EN** = 1) or the CPU / MCS via AEI (**TIM[i]_CH[x]_CTRL.ARU_EN** = 0).

To guarantee a consistent delivery of data from the **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** registers to the ARU or the CPU each TIM channel has to ensure that the data is consumed before it is overwritten with new values.

If new data was produced by the TIM channel (bit **TIM[i]_CH[x]_IRQ_NOTIFY.NEWVAL** is set) while the old data is not consumed by the ARU (**TIM[i]_CH[x]_CTRL.ARU_EN** = 1) or CPU (**TIM[i]_CH[x]_CTRL.ARU_EN** = 0), the TIM channel sets the **TIM[i]_CH[x]_IRQ_NOTIFY.GPROFL** bit and it overwrites the data inside the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** . In addition when **TIM[i]_CH[x]_CTRL.ARU_EN** =1, the bit *ARU_OUT* [49:49] is set to 1 to indicate the overflow in the ARU data.

If the CPU is selected as consumer for the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** (**TIM[i]_CH[x]_CTRL.ARU_EN** = 0), the acknowledge for reading out data is performed by a read access to the register **TIM[i]_CH[x]_GPR0** . Thus, register **TIM[i]_CH[x]_GPR1** should be read always before **TIM[i]_CH[x]_GPR0** .

If the ARU is selected as consumer for the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** (**TIM[i]_CH[x]_CTRL.ARU_EN** = 1), the acknowledge for reading out data is performed by the ARU itself. However, the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** could be read by CPU without giving an acknowledge.

13.4.2 TIM Channel Modes

The TIM provides seven different measurement modes that can be configured with the bit field **TIM[i]_CH[x]_CTRL.TIM_MODE** of register **TIM[i]_CH[x]_CTRL** . The measurement modes are described in the following subsections. Besides these different basic measurement modes, there exist distinct configuration bits in the register **TIM[i]_CH[x]_CTRL** for a more detailed controlling of each mode. The meanings of these bits are as follows:

TIM[i]_CH[x]_CTRL.DSL : control the signal level for the measurement modes (e.g. if a measurement is started with rising edge or falling edge, or if high level pulses or low level pulses are measured.

TIM[i]_CH[x]_CTRL.EGPR0_SEL , **TIM[i]_CH[x]_CTRL.GPR0_SEL** and **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** : control the actual content of the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** after a measurement has finished.

TIM[i]_CH[x]_CTRL.CNTS_SEL : control the content of the registers **TIM[i]_CH[x]_CNTS** . The actual time for updating the **TIM[i]_CH[x]_CNTS** register is mode dependent.

TIM[i]_CH[x]_CTRL.OSM : activate measurement in one-shot mode or continuous mode. In one-shot mode only one measurement cycle is performed and after that the channel is disabled.

TIM[i]_CH[x]_IRQ_NOTIFY.NEWVAL : The NEWVAL IRQ interrupt is triggered at the end of a measurement cycle, signaling that the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are updated.

TIM[i]_CH[x]_CTRL.ARU_EN : enables sending of the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** together with the actual signal level (in bit 48) and the overflow signal **TIM[i]_CH[x]_IRQ_NOTIFY.GPROFL** (in bit 49), and the timeout status information (bit 50) to the ARU.

TIM[i]_CH[x]_CTRL.EXT_CAP_EN : forces an update of the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** and **TIM[i]_CH[x]_CNTS** (TIM channel mode dependent) only on each rising edge of the *TIM_EXT_CAPTURE* [x:x] signal and triggers a *TIM_NEWVAL_IRQ* [x:x] interrupt. If this mode is disabled the *TIM_NEWVAL_IRQ* [x:x] interrupt is triggered at the end of each measurement cycle.

For each channel, the source of the *TIM_EXT_CAPTURE* signal can be configured with the bit fields **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** in the register **TIM[i]_CH[x]_ECTRL** .

13.4.2.1 TIM PWM Measurement Mode (TPWM)

In TIM PWM Measurement Mode, the TIM channel measures duty cycle and period of an incoming PWM signal. The **TIM[i]_CH[x]_CTRL.DSL** bit defines the polarity of the PWM signal to be measured.

When measurement of pulse high time and period is requested (PWM with a high level duty cycle, **TIM[i]_CH[x]_CTRL.DSL** =1) and **TIM[i]_CH[x]_ECTRL.IMM_START** =0, the channel starts measuring after the first rising edge is detected by the filter.

If **TIM[i]_CH[x]_ECTRL.IMM_START** =1 the measurement starts immediately after activating the channel by **TIM[i]_CH[x]_CTRL.TIM_EN** =1.

Measurement is done with the **TIM[i]_CH[x]_CNT** register counting with the configured clock coming from *CCM[i]_CLK_RES* [x:x] until a falling edge is detected.

Assume: **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =0, **TIM[i]_CH[x]_CTRL.ECNT_RESET** =0

Then the counter value is stored inside the shadow register **TIM[i]_CH[x]_CNTS** (if **TIM[i]_CH[x]_CTRL.CNTS_SEL** = 0) and the counter **TIM[i]_CH[x]_CNT** counts continuously until the next rising edge is reached.

On this following rising edge the content of the **TIM[i]_CH[x]_CNTS** register is transferred to **TIM[i]_CH[x]_GPR0** and the content of **TIM[i]_CH[x]_CNT** register is transferred to **TIM[i]_CH[x]_GPR1** , assuming settings for the selectors **TIM[i]_CH[x]_CTRL.EGPR0_SEL** =0, **TIM[i]_CH[x]_CTRL.GPR0_SEL** =0b11 and **TIM[i]_CH[x]_CTRL.EGPR1_SEL** =0, **TIM[i]_CH[x]_CTRL.GPR1_SEL** =0b11. By this, **TIM[i]_CH[x]_GPR0** contains the pulse width length and **TIM[i]_CH[x]_GPR1** contains the period.

Note: The bits **TIM[i]_CH[x]_ECNT** [7:1] may be used to check data consistency of the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** .

In addition, the **TIM[i]_CH[x]_CNT** register is cleared **TIM[i]_CH[x]_IRQ_NOTIFY.NEWVAL** status bit and depending on corresponding interrupt enable condition *TIM_NEWVAL_IRQ* [x:x] interrupt is raised.

The **TIM[i]_CH[x]_CNTS** register update is not performed until the measurement is started. Afterwards each edge leaving the level defined by **TIM[i]_CH[x]_CTRL.DSL** is performing a **TIM[i]_CH[x]_CNTS** register update.

If a PWM with a low level duty cycle should be measured (**TIM[i]_CH[x]_CTRL.DSL** = 0) and **TIM[i]_CH[x]_ECTRL.IMM_START** =0, the channel waits for a falling edge until measurement is started. On this edge the low level duty cycle time is stored first in **TIM[i]_CH[x]_CNTS** and then finally in **TIM[i]_CH[x]_GPR0** and the period is stored in **TIM[i]_CH[x]_GPR1** .

When a PWM period was successfully measured, the data in the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** is marked as valid for reading by the ARU when the **TIM[i]_CH[x]_CTRL.ARU_EN** bit is set inside **TIM[i]_CH[x]_CTRL** register, the **TIM[i]_CH[x]_IRQ_NOTIFY.NEWVAL** bit is set and a new measurement is started.

If the preceding PWM values were not consumed by a reader attached to the ARU (**TIM[i]_CH[x]_CTRL.ARU_EN** bit enabled) or by the CPU the TIM channel set **TIM[i]_CH[x]_IRQ_NOTIFY.GPROFL** status bit and depending on corresponding interrupt enable bit value raises a *TIM_GPROFL_IRQ* [x:x] and overwrites the old values in **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** . A new measurement is started afterwards.

If the register **TIM[i]_CH[x]_CNT** produces an overflow (wrap around 0xFFFFFF to 0x0 due to increment) during the measurement, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.CNTOFL** is set and interrupt *TIM_CNTOFL_IRQ* [x:x] is raised depending on corresponding interrupt enable condition.

If the register **TIM[i]_CH[x]_ECNT** produces an overflow during the measurement, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.ECNTOFL** is set and interrupt *TIM_ECNTOFL_IRQ* [x:x] is raised depending on corresponding interrupt enable condition.

If **TIM[i]_CH[x]_CTRL.ECNT_RESET** =0 the counter **TIM[i]_CH[x]_CNT** will be reset to 0 on active edge (defined by **TIM[i]_CH[x]_CTRL.DSL**) of the input signal. If **TIM[i]_CH[x]_CTRL.ECNT_RESET** =1 the counter **TIM[i]_CH[x]_CNT** will be reset to 0 on each edge of the input signal.

Assume **TIM[i]_CH[x]_CTRL.EXT_CAP_EN** =0 and **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =0:
On every input edge to the active level defined by **TIM[i]_CH[x]_CTRL.DSL** will capture the data selected by **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** to the registers **TIM[i]_CH[x]_GPR1** . Every edge to the inactive level will capture the data selected by **TIM[i]_CH[x]_CTRL.CNTS_SEL** to the registers **TIM[i]_CH[x]_CNTS** .

Assume **TIM[i]_CH[x]_CTRL.EXT_CAP_EN** =0 and **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =1:
On every input edge to the inactive level defined by **TIM[i]_CH[x]_CTRL.DSL** will capture the data selected by **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** to the registers **TIM[i]_CH[x]_GPR1** . Every edge to the active level will capture the data selected by **TIM[i]_CH[x]_CTRL.CNTS_SEL** to the registers **TIM[i]_CH[x]_CNTS** .

13.4.2.1.1 External Capture TIM PWM Measurement Mode (TPWM)

If external capture is enabled **TIM[i]_CH[x]_CTRL.EXT_CAP_EN** =1, the PWM measurement is done continuously. The actual measurement values are captured to **TIM[i]_CH[x]_GPR0** , **TIM[i]_CH[x]_GPR1** if an external capture event occurs.

On every external capture event the data selected by **TIM[i]_CH[x]_CTRL.CNTS_SEL** , **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR0_SEL** will be captured to the registers **TIM[i]_CH[x]_CNTS** , **TIM[i]_CH[x]_GPR0** .

If **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =0 every external capture event will capture the data selected by **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** to the registers **TIM[i]_CH[x]_GPR1** . Every input edge to the level != **TIM[i]_CH[x]_CTRL.DSL** will capture the data selected by **TIM[i]_CH[x]_CTRL.CNTS_SEL** to the registers **TIM[i]_CH[x]_CNTS** .

If **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =1 every input edge to the inactive level != **TIM[i]_CH[x]_CTRL.DSL** will capture the data selected by **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** to the registers **TIM[i]_CH[x]_GPR1** .

Assume **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =0:

Operation is done depending on CMU clock, **TIM[i]_CH[x]_CTRL.ISL** , **TIM[i]_CH[x]_CTRL.DSL** bit and the input signal value defined in next table (Assume **TIM[i]_CH[x]_CTRL.CNTS_SEL** = 0):

Table 17 Operation Depending on CMU clock, TIM[i]_CH[x]_CTRL.ISL, TIM[i]_CH[x]_CTRL.DSL and the Input Signal Value (Assume TIM[i]_CH[x]_CTRL.CNTS_SEL= 0)

Input signal *FOUT* [x:x]	Selected CMU clock	External capture	TIM[i]_CH[x]_CTRL.ISL	TIM[i]_CH[x]_CTRL.DSL	Action description
0	1	0	–	0	**TIM[i]_CH[x]_CNT** ++
1	1	0	–	0	No
Rising edge	–	0	0	0	Capture **TIM[i]_CH[x]_CNT** value in **TIM[i]_CH[x]_CNTS**
Falling edge	–	0	0	0	**TIM[i]_CH[x]_CNT** =0
Rising edge	–	0	1	0	No
Falling edge	–	0	1	0	Capture **TIM[i]_CH[x]_CNT** value in **TIM[i]_CH[x]_CNTS** ; **TIM[i]_CH[x]_CNT** =0
1	1	0	–	1	**TIM[i]_CH[x]_CNT** ++
0	1	0	–	1	No
Falling edge	–	0	0	1	Capture **TIM[i]_CH[x]_CNT** value in **TIM[i]_CH[x]_CNTS**
Rising edge	–	0	0	1	**TIM[i]_CH[x]_CNT** =0
Falling edge	–	0	1	1	No
Rising edge	–	0	1	1	Capture **TIM[i]_CH[x]_CNT** value in **TIM[i]_CH[x]_CNTS** ; **TIM[i]_CH[x]_CNT** =0
–	–	Rising edge	–	–	Do **TIM[i]_CH[x]_GPR0** , **TIM[i]_CH[x]_GPR1** capture ; issue *TIM_NEWVAL_IRQ*
–	0	0	–	–	No

The **TIM[i]_CH[x]_CNTS** register update is not performed until the measurement is started (first edge defined by **TIM[i]_CH[x]_CTRL.DSL** is detected). Afterwards the update of the **TIM[i]_CH[x]_CNTS** register is defined by **TIM[i]_CH[x]_CTRL.ISL** , **TIM[i]_CH[x]_CTRL.DSL** combinations in the table above.

13.4.2.2 TIM Pulse Integration Mode (TPIM)

In TIM Pulse Integration Mode each TIM channel is able to measure a sum of pulse high or low times on an input signal, depending on the selected signal level bit **TIM[i]_CH[x]_CTRL.DSL** of register **TIM[i]_CH[x]_CTRL** register.

If **TIM[i]_CH[x]_ECTRL.IMM_START** =0 the pulse integration measurement is started with occurrence of the first edge defined by **TIM[i]_CH[x]_CTRL.DSL** on the input signal. If **TIM[i]_CH[x]_ECTRL.IMM_START** =1 the measurement starts immediately after activating the channel by **TIM[i]_CH[x]_CTRL.TIM_EN** =1.

The pulse times are measured by incrementing the TIM channel counter **TIM[i]_CH[x]_CNT** until the counter is stopped with occurrence of a input signal edge to the opposite signal level defined by **TIM[i]_CH[x]_CTRL.DSL** .

The counter **TIM[i]_CH[x]_CNT** counts with the *CCM[i]_CLK_RES* [x:x] clock specified by the **TIM[i]_CH[x]_CTRL.CLK_SEL** bit field of the **TIM[i]_CH[x]_CTRL** register.

If the register **TIM[i]_CH[x]_CNT** produces an overflow (wrap around 0xFFFFFF to 0x0 due to increment) during the measurement, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.CNTOFL** is set and interrupt *TIM_CNTOFL_IRQ* [x:x] is raised depending on corresponding interrupt enable condition.

The **TIM[i]_CH[x]_CNT** register is reset at the time the channel is activated (enabling via AEI write access) and it accumulates pulses while the channel is staying enabled.

Assume **TIM[i]_CH[x]_CTRL.EXT_CAP_EN** =0 and **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =0:
After measurement is started, every falling(**TIM[i]_CH[x]_CTRL.DSL** =1) or rising(**TIM[i]_CH[x]_CTRL.DSL** =0) input edge will issue a *TIM_NEWVAL_IRQ* [x:x] interrupt, and the registers **TIM[i]_CH[x]_CNTS** , **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are updated according to settings of its corresponding input multiplexers, using the bits **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** and **TIM[i]_CH[x]_CTRL.CNTS_SEL** .

Note: The bits 1 to 7 of the **TIM[i]_CH[x]_ECNT** may be used to check data consistency of the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** .

Assume **TIM[i]_CH[x]_CTRL.EXT_CAP_EN** =0 and **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =1:
After measurement is started, every falling (**TIM[i]_CH[x]_CTRL.DSL** =1) or rising (**TIM[i]_CH[x]_CTRL.DSL** =0) input edge will issue a *TIM_NEWVAL_IRQ* [x:x] interrupt, and the registers **TIM[i]_CH[x]_CNTS** , **TIM[i]_CH[x]_GPR0** are updated according to settings of its corresponding input multiplexers, using the bits **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR0_SEL** and **TIM[i]_CH[x]_CTRL.CNTS_SEL** .
Every input edge to the active level defined by **TIM[i]_CH[x]_CTRL.DSL** (rising **TIM[i]_CH[x]_CTRL.DSL** =1; falling **TIM[i]_CH[x]_CTRL.DSL** =0) will capture the data selected by **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** to the registers **TIM[i]_CH[x]_GPR1** .

When the **TIM[i]_CH[x]_CTRL.ARU_EN** bit is set inside the **TIM[i]_CH[x]_CTRL** register the measurement results of the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** can be send to subsequent sub−modules attached to the ARU.

13.4.2.2.1 External Capture TIM Pulse Integration Mode (TPIM)

If external capture is enabled **TIM[i]_CH[x]_CTRL.EXT_CAP_EN** =1, the pulse integration is done until next external capture event occurs.

On every external capture event the data selected by **TIM[i]_CH[x]_CTRL.CNTS_SEL** , **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR0_SEL** will be captured to the registers **TIM[i]_CH[x]_CNTS** , **TIM[i]_CH[x]_GPR0** .

If **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =0 every external capture event will capture the data selected by **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** to the registers **TIM[i]_CH[x]_GPR1** .

If **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =1 every input edge to the inactive level != **TIM[i]_CH[x]_CTRL.DSL** will capture the data selected by **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** to the registers **TIM[i]_CH[x]_GPR1** .

Assume **TIM[i]_CH[x]_ECTRL.SWAP_CAPTURE** =0; **TIM[i]_CH[x]_ECTRL.IMM_START** =0:

Operation is done depending on CMU clock, **TIM[i]_CH[x]_CTRL.DSL** bit and the input signal value defined in next table (inc_cnt = false if TIM channel is enabled) :

Table 18 Operation Depending on CMU clock, TIM[i]_CH[x]_CTRL.DSL and the Input Signal Value (inc_cnt = False if TIM Channel is Enabled)

Input signal *F_OUT* [x:x]	Selected CMU Clock	External capture	TIM[i]_CH[x]_CTRL.ISL	TIM[i]_CH[x]_CTRL.DSL	Action description
Falling edge	–	0	–	0	inc_cnt = true
Rising edge	–	0	–	0	inc_cnt = false
Rising edge	–	0	–	1	inc_cnt = true
Falling edge	–	0	–	1	inc_cnt = false
–	1	0	–	–	If inc_cnt == true then **TIM[i]_CH[x]_CNT** ++;
–	–	Rising edge	–	–	Do capture **TIM[i]_CH[x]_GPR0** , **TIM[i]_CH[x]_GPR1** , **TIM[i]_CH[x]_CNTS** ; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** =0
–	0	0	–	–	No

13.4.2.3 TIM Input Event Mode (TIEM)

In TIM Input Event Mode, the TIM channel is able to count edges.

It is configurable if rising, falling or both edges should be counted. This can be done with the bit fields **TIM[i]_CH[x]_CTRL.DSL** and **TIM[i]_CH[x]_CTRL.ISL** in **TIM[i]_CH[x]_CTRL** register.

In addition, a *TIM_NEWVAL_IRQ* [x:x] interrupt is raised when the configured edge was received and this interrupt was enabled.

The counter register **TIM[i]_CH[x]_CNT** is used to count the number of edges, and the bit fields **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** , and **TIM[i]_CH[x]_CTRL.CNTS_SEL** can be used to configure the desired update values for the registers **TIM[i]_CH[x]_GPR0** , **TIM[i]_CH[x]_GPR1** and **TIM[i]_CH[x]_CNTS** . These registers are updated whenever the edge counter **TIM[i]_CH[x]_CNT** is incremented due to the arrival of a desired edge.

If the preceding data was not consumed by a reader attached to the ARU or by the CPU the TIM channel sets **TIM[i]_CH[x]_IRQ_NOTIFY.GPROFL** status bit and raises a *TIM_GPROFL_IRQ* [x:x] if it was enabled in **TIM[i]_CH[x]_IRQ_EN** register and overwrites the old values in **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** with the new ones.

If the register **TIM[i]_CH[x]_CNT** produces an overflow (wrap around 0xFFFFFF to 0x0 due to increment) during the measurement, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.CNTOFL** is set and interrupt *TIM_CNTOFL_IRQ* [x:x] is raised depending on corresponding interrupt enable condition.

If the register **TIM[i]_CH[x]_ECNT** produces an overflow during the measurement, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.ECNTOFL** is set and interrupt *TIM_ECNTOFL_IRQ* [x:x] is raised depending on corresponding interrupt enable condition.

The TIM Input Event Mode does not depend on the bit field **TIM[i]_CH[x]_CTRL.CLK_SEL** of register **TIM[i]_CH[x]_CTRL** .

13.4.2.3.1 External Capture TIM Input Event Mode (TIEM)

If external capture is enabled, capturing is done depending on the **TIM[i]_CH[x]_CTRL.DSL** , **TIM[i]_CH[x]_CTRL.ISL** bit and the input signal value defined in next table:

Table 19 Capturing Depended on the TIM[i]_CH[x]_CTRL.DSL, TIM[i]_CH[x]_CTRL.ISL and the Input Signal Value, If External Capture is Enabled

Input signal *F_OUT* [x:x]	External capture	TIM[i]_CH[x]_CTRL.ISL	TIM[i]_CH[x]_CTRL.DSL	Action description
–	Rising edge	1	–	Do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** ++
–	0	1	–	No
1	Rising edge	0	1	Do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** ++
0	–	0	1	No
0	Rising edge	0	0	Do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** ++
1	–	0	0	No

13.4.2.4 TIM Input Prescaler Mode (TIPM)

In the TIM Input Prescaler Mode, the number of edges which should be detected before a *TIM_NEWVAL_IRQ* [x:x] is raised is programmable. In this mode it must be specified in the **TIM[i]_CH[x]_CNTS** register after how many edges the interrupt has to be raised.

A value of 0 in **TIM[i]_CH[x]_CNTS** means that after one edge an interrupt is raised and a value of 1 means that after two edges an interrupt is raised, and so on.

The edges to be counted can be selected by the bit fields **TIM[i]_CH[x]_CTRL.DSL** and **TIM[i]_CH[x]_CTRL.ISL** .

With each triggered interrupt, the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are updated according to bits **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR0_SEL** and **TIM[i]_CH[x]_CTRL.GPR1_SEL** .

If the register **TIM[i]_CH[x]_ECNT** produces an overflow during the measurement, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.ECNTOFL** is set and interrupt *TIM_ECNTOFL_IRQ* [x:x] is raised depending on corresponding interrupt enable condition.

The register **TIM[i]_CH[x]_CNT** never produces an overflow during this mode, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.CNTOFL** is never set.

The TIM Input Prescaler Mode does not depend on the bit field **TIM[i]_CH[x]_CTRL.CLK_SEL** of register **TIM[i]_CH[x]_CTRL** .

13.4.2.4.1 External Capture TIM Input Prescaler Mode (TIPM)

If external capture is enabled, the external capture events are counted instead of the input signal edges.

Operation is done depending on the external capture signal, **TIM[i]_CH[x]_CTRL.DSL** , **TIM[i]_CH[x]_CTRL.ISL** bit and the input signal value defined in next table:

Table 20 Operation Depending on the External Capture Signal, TIM[i]_CH[x]_CTRL.DSL, TIM[i]_CH[x]_CTRL.ISL and the Input Signal Value

Input signal *F_OUT* [x:x]	External capture	TIM[i]_CH[x]_CTRL.ISL	DSL	Action description

–	Rising edge	1	–	If **TIM[i]_CH[x]_CNT** >= **TIM[i]_CH[x]_CNTS** then do capture ;issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** =0
				else
				TIM[i]_CH[x]_CNT ++
				endif

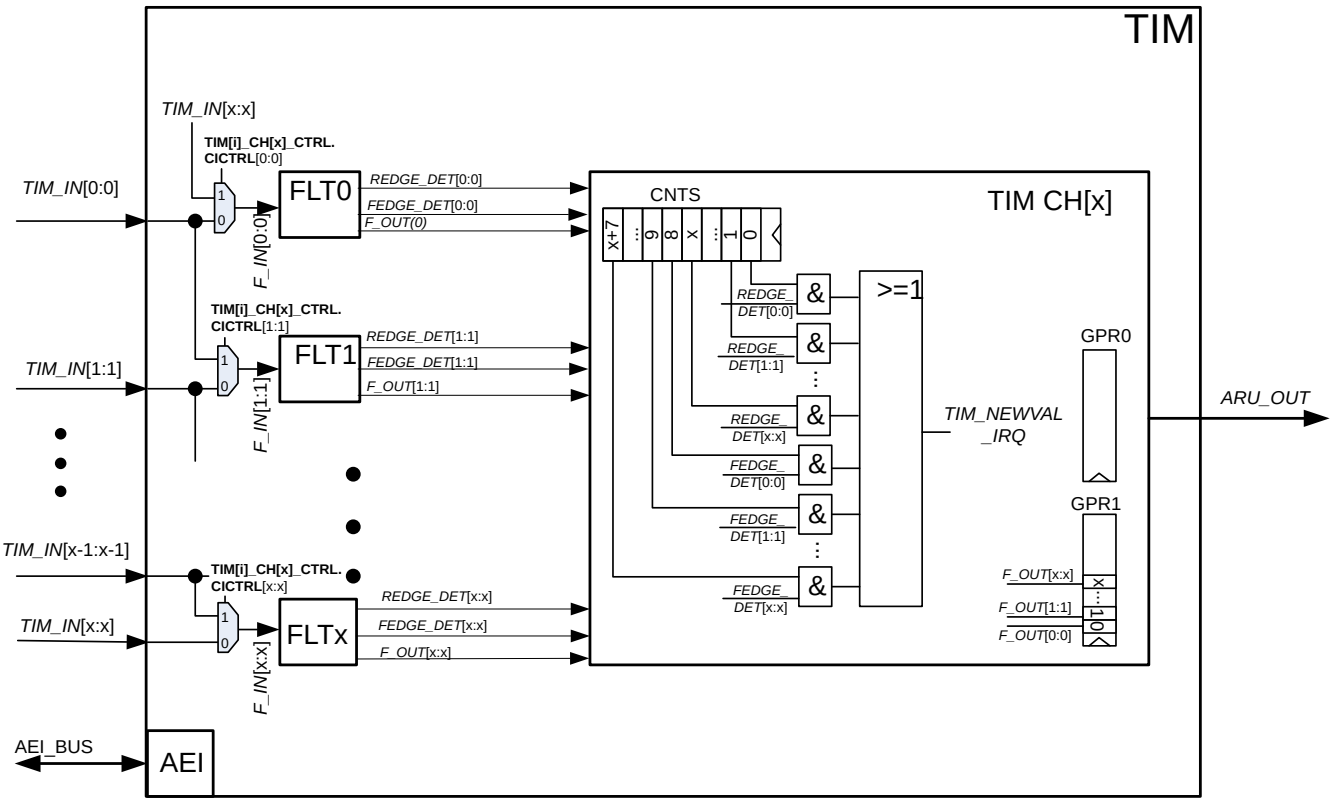
Input signal *F_OUT* [x:x]	External capture	TIM[i]_CH[x]_CTRL.ISL	DSL	Action description
–	0	1	–	No
1	Rising edge	0	1	If **TIM[i]_CH[x]_CNT** >= **TIM[i]_CH[x]_CNTS** then do capture ;issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** =0 else **TIM[i]_CH[x]_CNT** ++ endif
0	–	0	1	No
0	Rising edge	0	0	If **TIM[i]_CH[x]_CNT** >= **TIM[i]_CH[x]_CNTS** then do capture ;issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** =0 else **TIM[i]_CH[x]_CNT** ++ endif
1	–	0	0	No

13.4.2.5 TIM Bit Compression Mode (TBCM)

The TIM Bit Compression Mode can be used to combine all filtered input signals of a TIM sub-module to a parallel m bit data word, which can be routed to the ARU, where m is the number of channels available in the TIM sub-module.

Figure *55 "TIM Bit Compression Mode"* gives an overview of the TIM bit compression mode.

Figure 55 TIM Bit Compression Mode



The register **TIM[i]_CH[x]_CNTS** of a channel is used to configure the event that releases the *TIM_NEWVAL_IRQ* and samples the input signals *F_IN* [0:0] to *F_IN* [7:7] of instance [i-1] in ascending order as a parallel data word in **TIM[i]_CH[x]_GPR1** .

The bits 0 to m-1 of the **TIM[i]_CH[x]_CNTS** register are used to select the *REDGE_DET* signals of the TIM filters 0 to m-1 as a sampling event, and the bits 8 to (7+m) are used to select the *FEDGE_DET* signals of the TIM filters 0 to m-1, respectively. If multiple events are selected, the events are OR-combined (see also Figure *55 "TIM Bit Compression Mode"*).

The value of the bitfields **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR0_SEL** selects the timestamp value, which is routed through the ARU.

If the bit field **TIM[i]_CH[x]_CTRL.ARU_EN** is set, the sampled data of register **TIM[i]_CH[x]_GPR1** is routed together with a time stamp of register **TIM[i]_CH[x]_GPR0** to the ARU, whenever the *TIM_NEWVAL_IRQ* is released.

In TIM Bit compression mode, the register **TIM[i]_CH[x]_ECNT** increments with each *TIM_NEWVAL_IRQ* , which means that the value of **TIM[i]_CH[x]_ECNT** may depend on all m input signals. Consequently, the LSB of **TIM[i]_CH[x]_ECNT** does not reflect the actual level of the input signal **TIM[i]_INP_VAL.TIM_IN[x]** .

If the register **TIM[i]_CH[x]_ECNT** produces an overflow during the measurement, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.ECNTOFL** is set and interrupt *TIM_ECNTOFL_IRQ* [x:x] is raised depending on corresponding interrupt enable condition.

The TIM Bit Compression Mode does not depend on the bit field **TIM[i]_CH[x]_CTRL.CLK_SEL** of register **TIM[i]_CH[x]_CTRL** .

13.4.2.5.1 External Capture Bit Compression Mode (TBCM)

If external capture is enabled, capturing is done for **TIM[i]_CH[x]_CTRL.ISL** =1 as defined in next table. The value 0 for **TIM[i]_CH[x]_CTRL.ISL** is prohibited.

Table 21 Capturing Depended on the TIM[i]_CH[x]_CTRL.DSL, TIM[i]_CH[x]_CTRL.ISL and the Input Signal Value, If External Capture is Enabled

Input signal *F_OUT* [x:x]	External capture	TIM[i]_CH[x]_CTRL.ISL	TIM[i]_CH[x]_CTRL.DSL	Action description
–	Rising edge	1	–	Do capture ;issue *TIM_NEWVAL_IRQ* ;
–	0	1	–	No
–	–	0 – prohibited	–	–

13.4.2.6 TIM Gated Periodic Sampling Mode (TGPS)

In the TIM Gated Periodic Sampling Mode, the number of CMU clock cycles which should elapse before capturing and raising *TIM_NEWVAL_IRQ* [x:x] is programmable. In this mode it must be specified in the **TIM[i]_CH[x]_CNTS** register after how many CMU clock cycles the interrupt has to be raised.

A value of 0 in **TIM[i]_CH[x]_CNTS** means that after one **TIM[i]_CH[x]_CTRL.CLK_SEL** edge a trigger/interrupt is raised, and a value of 1 means that after two edges a trigger/interrupt is raised, and so on.

In the **TIM[i]_CH[x]_CNT** register the elapsed cycles were incremented and compared against **TIM[i]_CH[x]_CNTS** . If **TIM[i]_CH[x]_CNT** is greater or equal to **TIM[i]_CH[x]_CNTS** a trigger will be raised. This allows by writing a value to **TIM[i]_CH[x]_CNTS** that the actual period time can be changed on the fly.

Operation is done depending on CMU clock, **TIM[i]_CH[x]_CTRL.DSL** , **TIM[i]_CH[x]_CTRL.ISL** bit and the input signal value defined in next table:

The register **TIM[i]_CH[x]_CNT** never produces an overflow during this mode, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.CNTOFL** is never set.

Table 22 Operation Depending on CMU Clock, TIM[i]_CH[x]_CTRL.DSL, TIM[i]_CH[x]_CTRL.ISL and the Input Signal Value

Input signal F_OUT[x:x]	Selected CMU clock	External capture	TIM[i]_CH[x]_CTRL.ISL	DSL	Action description

–	1	0	1	–	If **TIM[i]_CH[x]_CNT** >= **TIM[i]_CH[x]_CNTS** then
					do capture ;issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** =0
					else
					TIM[i]_CH[x]_CNT ++
					endif

0	0	0	0	1	No

1	1	0	0	1	If **TIM[i]_CH[x]_CNT** >= **TIM[i]_CH[x]_CNTS** then
					do capture ;issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** =0
					else
					TIM[i]_CH[x]_CNT ++
					endif

0	0	–	0	1	No

Input signal F_OUT-[x:x]	Selected CMU clock	External capture	TIM[i]_CH[x]_CTR-L.ISL	DSL	Action description

0	1	0	0	0	If **TIM[i]_CH[x]_C-NT** >= **TIM[i]_CH[x]_-CNTS** then
					do capture ;issue *T-IM_NEWVAL_IRQ* ; **T-IM[i]_CH[x]_CNT** =0
					else
					TIM[i]_CH[x]_C-NT ++
					endif

1	0	0	0	0	No
–	0	0	–	–	No

In this mode the **TIM[i]_CH[x]_GPR1** operates as a shadow register for **TIM[i]_CH[x]_CNTS** . This would allow that the period for the next sampling period could be specified. The update of **TIM[i]_CH[x]_CNTS** will only take place once on a trigger if the **TIM[i]_CH[x]_GPR1** was written by the CPU. This means that the captured value from the previous trigger can be read by the CPU from **TIM[i]_CH[x]_GPR1** and afterwards the new sampling period for the next sampling period (the one after the actual sampling period) could be written.

With each triggered interrupt, the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are updated according to bits **TIM[i]_CH[x]_CTR-L.GPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** , **TIM[i]_CH[x]_CTRL.EGPR0_SEL** and **TIM[i]_CH[x]_CTRL.EGPR1_SEL** .

When selecting **TIM[i]_CH[x]_ECNT** as a source for the capture registers, **TIM[i]_CH[x]_GPR0** , **TIM[i]_CH[x]_GPR1** will show the edge count and the input signal value at point of capture. Selecting **TIM[i]_CH[x]_CTRL.GPR0_SEL** = '11' and **TIM[i]_CH[x]_CTRL.EGPR0_SEL** = '0' for TIM channel 0 all 8 TIM input signals will be captured to **TIM[i]_CH[x]_GPR0** [7:0].

In the TGPS Mode, the bit field **TIM[i]_CH[x]_CTRL.CLK_SEL** of register **TIM[i]_CH[x]_CTRL** will define the selected CMU clock which will be used.

The behavior of the **TIM[i]_CH[x]_ECNT** counter is configurable by **TIM[i]_CH[x]_CTRL.ECNT_RESET** . If set to 1 on each interrupt (period expired) the **TIM[i]_CH[x]_ECNT** will be reset. Otherwise it operates in wrap around mode.

If the register **TIM[i]_CH[x]_ECNT** produces an overflow during the measurement, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.ECNTOFL** is set and interrupt *TIM_ECNTOFL_IRQ* [x:x] is raised depending on corresponding interrupt enable condition.

13.4.2.6.1 External Capture TIM Gated Periodic Sampling Mode (TGP-S)

If external capture is enabled, the external capture events will capture actual values in the register **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_-GPR1** , reset the counter **TIM[i]_CH[x]_CNT** and issue a *TIM_NEWVAL_IRQ* .

Operation is done depending on the CMU clock, external capture signal, **TIM[i]_CH[x]_CTRL.DSL** , **TIM[i]_CH[x]_CTRL.ISL** bit and the input signal value defined in next table:

Table 23 Operation Depending on the CMU Clock, External Capture Signal, TIM[i]_CH[x]_CTRL.DSL, TIM[i]_CH[x]_CTRL.ISL and the Input Signal Value

Input signal *F_O-UT* [x:x]	Selected CMU clock	External capture	TIM[i]_CH[x]_CTR-L.ISL	TIM[i]_CH[x]_CTR-L.DSL	Action description

–	1	0	1	–	If **TIM[i]_CH[x]_C-NT** >= **TIM[i]_CH[x]_-CNTS** then
					do capture; issue *T-IM_NEWVAL_IRQ* ; **T-IM[i]_CH[x]_CNT** =0
					else
					TIM[i]_CH[x]_C-NT ++
					endif

0	0	0	0	1	No

1	1	0	0	1	If **TIM[i]_CH[x]_C-NT** >= **TIM[i]_CH[x]_-CNTS** then
					do capture; issue *T-IM_NEWVAL_IRQ* ; **T-IM[i]_CH[x]_CNT** =0
					else
					TIM[i]_CH[x]_C-NT ++
					endif

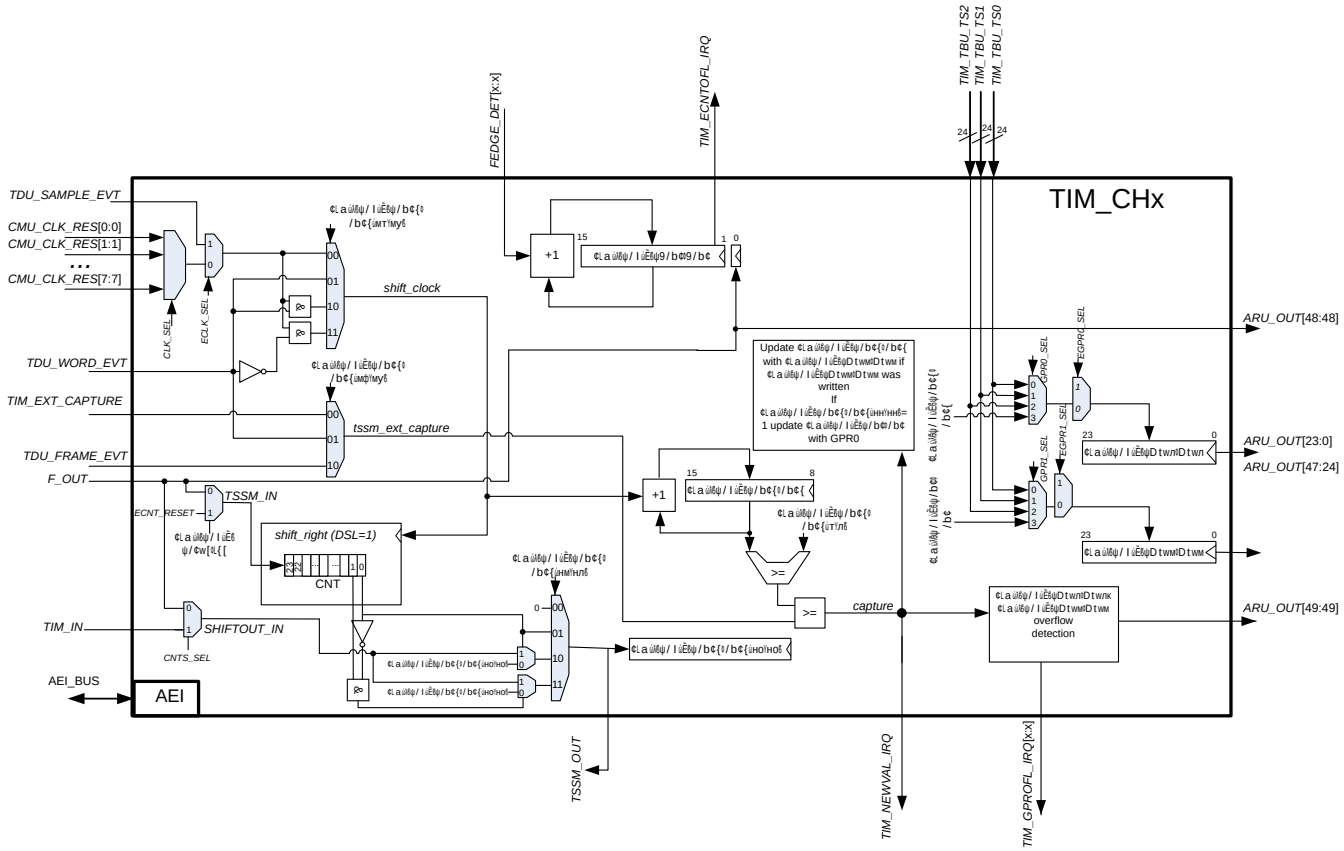
Input signal *F_OUT* [x:x]	Selected CMU clock	External capture	TIM[i]_CH[x]_CTRL.ISL	TIM[i]_CH[x]_CTRL.DSL	Action description
0	0	–	0	1	No

0	1	0	0	0	If **TIM[i]_CH[x]_CNT** >= **TIM[i]_CH[x]_CNTS** then
					do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** =0
					else
					TIM[i]_CH[x]_CNT ++
					endif

1	0	0	0	0	No
–	0	0	–	–	No
–	–	Rising edge	–	–	Do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNT** =0

13.4.2.7 TIM Serial Shift Mode (TSSM)

Figure 56 TIM Serial Shift Mode



In the TIM Serial Shift Mode on each shift clock event, the actual value of the input signal *TSSM_IN* [x:x] will be registered in dependence of **TIM[i]_CH[x]_CTRL.DSL** in the register **TIM[i]_CH[x]_CNT** .
If **TIM[i]_CH[x]_CTRL.ISL** =0 is set *F_OUT* [x:x] will be used as shift in value *TSSM_IN* [x:x] , with **TIM[i]_CH[x]_CTRL.ISL** =1 the bit field **TIM[i]_CH[x]_CTRL.ECNT_RESET** defines the value for *TSSM_IN* [x:x].
With **TIM[i]_CH[x]_CTRL.DSL** =0 *TSSM_IN* [x:x] will be stored in **TIM[i]_CH[x]_CNT** [0:0] and **TIM[i]_CH[x]_CNT** [22:0] will be shifted left.
With **TIM[i]_CH[x]_CTRL.DSL** =1 *TSSM_IN* [x:x] will be stored in **TIM[i]_CH[x]_CNT** [23:23] and **TIM[i]_CH[x]_CNT** [23:1] will be shifted right.
Operation is done depending on the shift clock, external capture signal, **TIM[i]_CH[x]_CTRL.DSL** , **TIM[i]_CH[x]_CTRL.ISL** bit and the input signal value defined in next table:

The register **TIM[i]_CH[x]_CNT** never produces an overflow during this mode, the bit **TIM[i]_CH[x]_IRQ_NOTIFY.CNTOFL** is never set.

Table 24 Operation Depending on the Shift Clock, External Capture Signal, TIM[i]_CH[x]_CTRL.DSL, TIM[i]_CH[x]_CTRL.ISL and the Input Signal Value

Input signal *TSSM_IN* [x:x]	Shift clock	tssm_ext_capture	TIM[i]_CH[x]_CTRL.ISL	DSL	Action description
–	0	0	–	–	No

–	–	1	–	–	If **TIM[i]_CH[x]_CTRL.EXT_CAP_EN** =1 then
					see function table in next chapter
					else
					no
					endif

value	1	0	0	0	**TIM[i]_CH[x]_CNT** [23:1]= **TIM[i]_CH[x]_CNT** [22:0]; **TIM[i]_CH[x]_CNT** [0:0]= value
					if **TIM[i]_CH[x]_CNTS** [15:8] >= **TIM[i]_CH[x]_CNTS** [7:0] then
					do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0
					else
					TIM[i]_CH[x]_CNTS [15:8]++
					endif

value	1	0	0	1	**TIM[i]_CH[x]_CNT** [22:0]= **TIM[i]_CH[x]_CNT** [23:1]; **TIM[i]_CH[x]_CNT** [23:23]= value
					if **TIM[i]_CH[x]_CNTS** [15:8] >= **TIM[i]_CH[x]_CNTS** [7:0] then
					do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0
					else
					TIM[i]_CH[x]_CNTS [15:8]++
					endif

value	1	0	1	0	**TIM[i]_CH[x]_CNT** [23:1]= **TIM[i]_CH[x]_CNT** [22:0]; **TIM[i]_CH[x]_CNT** [0:0]= value
					if **TIM[i]_CH[x]_CNTS** [15:8] >= **TIM[i]_CH[x]_CNTS** [7:0] then
					do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0
					TIM[i]_CH[x]_CNT [23:0]= **TIM[i]_CH[x]_CTRL.ECNT_RESET**
					else
					TIM[i]_CH[x]_CNTS [15:8]++
					endif

BOSCH

value	1	0	1	1	**TIM[i]_CH[x]_CNT** [22:0]= **TIM[i]_CH[x]_CNT** [23:1]; **TIM[i]_CH[x]_CNT** [23:23]= value
					if **TIM[i]_CH[x]_CNTS** [15:8] >= **TIM[i]_CH[x]_CNTS** [7:0] then
					do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0
					TIM[i]_CH[x]_CNT [23:0]= **TIM[i]_CH[x]_CTRL.ECNT_RESET**
					else
					TIM[i]_CH[x]_CNTS [15:8]++
					endif

The register **TIM[i]_CH[x]_CNTS** [7:0] defines the number of bits which will be stored inside **TIM[i]_CH[x]_CNT** .

Each shift clock will increment the register **TIM[i]_CH[x]_CNTS** [15:8]. If the condition **TIM[i]_CH[x]_CNTS** [15:8] >= **TIM[i]_CH[x]_CNTS** [7:0] is met a capture event is raised and *TIM_NEWVAL_IRQ* [x:x] is asserted.

With each capture event the registers **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** are updated according to bits **TIM[i]_CH[x]_CTRL.GPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** , **TIM[i]_CH[x]_CTRL.EGPR0_SEL** and **TIM[i]_CH[x]_CTRL.EGPR1_SEL** .

If the bit field **TIM[i]_CH[x]_CTRL.ISL** is set to 1 the register bits **TIM[i]_CH[x]_CNT** are set to the value defined by **TIM[i]_CH[x]_CTRL.ECNT_RESET** in case of a capture event.

In this mode the **TIM[i]_CH[x]_GPR1** operates as a shadow register for **TIM[i]_CH[x]_CNTS** . This allows that the number of bits to sample can be specified. The update of **TIM[i]_CH[x]_CNTS** will only take place once on a trigger if the **TIM[i]_CH[x]_GPR1** was written by the CPU. This means that the captured value from the previous trigger can be read by the CPU from **TIM[i]_CH[x]_GPR1** and afterwards the new number of bits to sample for the next sampling period (the one after the actual sampling period) could be written.

TSSM using as serial shift in/out: **TIM[i]_CH[x]_CTRL.ISL** =0

- **TIM[i]_CH[x]_CNTS.CNTS** [22:22]=0: Shifted in data stored in **TIM[i]_CH[x]_CNT.CNT** can be captured on a capture event in **TIM[i]_CH[x]_GPR0.GPR0** , all other **TIM[i]_CH[x]_CTRL.GPR0_SEL/TIM[i]_CH[x]_CTRL.EGPR0_SEL** settings can be used too.
- **TIM[i]_CH[x]_CNTS.CNTS** [22:22]=1: The **TIM[i]_CH[x]_GPR0.GPR0** can never be used to capture data.
- The **TIM[i]_CH[x]_GPR0.GPR0** can be used as a shadow register to provide the next data which will be shifted out. An update of **TIM[i]_CH[x]_CNT.CNT** with the content of **TIM[i]_CH[x]_GPR0.GPR0** will happen on every capture event.

TSSM using as serial shift out: **TIM[i]_CH[x]_CTRL.ISL** =1

- TIM_IN data will not be stored in **TIM[i]_CH[x]_CNT.CNT** ; **TIM[i]_CH[x]_CTRL.ECNT_RESET** will be taken instead.
- **TIM[i]_CH[x]_CNTS.CNTS** [22:22]=1: **TIM[i]_CH[x]_GPR0.GPR0** can never be used to capture data.
- **TIM[i]_CH[x]_GPR0.GPR0** can be used as a shadow register to provide the next data which will be shifted out. An update of **TIM[i]_CH[x]_CNT.CNT** with the content of **TIM[i]_CH[x]_GPR0.GPR0** will happen on every capture event.
- **TIM[i]_CH[x]_CNTS.CNTS** [22:22]=0: On a capture event, values can be captured in **TIM[i]_CH[x]_GPR0.GPR0** , all **TIM[i]_CH[x]_CTRL.GPR0_SEL** / **TIM[i]_CH[x]_CTRL.EGPR0_SEL** settings can be used.

- The shift clock which will be in use is selectable by **TIM[i]_CH[x]_CNTS** [17:16]
- 0b00 = *USED_CLK_RES* is in use. It can be set to any CCM[i]_CLK_RES source or to the local TDU sample clock *TDU_SAMPLE_EVT* .
- 0b01 = the *TDU_WORD_EVT* signal will be used as shift clock source.
- 0b10 = *USED_CLK_RES* is gated with *TDU_WORD_EVT* . If *TDU_WORD_EVT* =0 then shift clock will be 0.
- 0b11 = *USED_CLK_RES* is gated with inverted *TDU_WORD_EVT* . If *TDU_WORD_EVT* =1 then shift clock will be 0.

13.4.2.7.1 Signal Generation with TIM Serial Shift Mode

If **TIM[i]_CH[x]_CNTS** [22:22] is 1 the **TIM[i]_CH[x]_GPR0** operates as a shadow register for **TIM[i]_CH[x]_CNT** . This allows that the bits for shifting out can be specified. The update of **TIM[i]_CH[x]_CNT** will only take place once on a trigger if the **TIM[i]_CH[x]_GPR0** was

written by the CPU. This means that the captured value from the previous trigger can be read by the CPU from **TIM[i]_CH[x]_GPR0** and afterwards the new bits to shift out could be written.

In addition, the TIM Serial Shift Mode is able to generate a signal *TSSM_OUT* which can be used internally to the TIM channel. On each system clock the value for *TSSM_OUT* is generated as defined next. The actual value can be read by the register bit **TIM[i]_CH[x]_CNTS** [23:23].

- Following functionality for *TSSM_OUT* [x:x] is selectable by **TIM[i]_CH[x]_CNTS** [21:20]
- 0b00: Constant output; *TSSM_OUT* [x:x] = 0.
- 0b01: Shift output; If **TIM[i]_CH[x]_CTRL.DSL** =0 (shift left) then *TSSM_OUT* [x:x] = **TIM[i]_CH[x]_CNT** [23:23] else (shift right) *TSSM_OUT* [x:x] = **TIM[i]_CH[x]_CNT** [0:0].
- 0b10: Latched output; If **TIM[i]_CH[x]_CTRL.DSL** =0 and **TIM[i]_CH[x]_CNT** [23:23]=1 then *TSSM_OUT* [x:x] = *SHIFTOUT_IN* [x:x] elsif TIM[i]_CH[x]_CTRL.DSL=1 and **TIM[i]_CH[x]_CNT** [0:0]=1 then *TSSM_OUT* [x:x] = *SHIFTOUT_IN* [x:x].
- 0b11: Registered output; If **TIM[i]_CH[x]_CTRL.DSL** =0 and **TIM[i]_CH[x]_CNT** [23:22]=0b01 then *TSSM_OUT* [x:x] = *SHIFTOUT_IN* [x:x] elsif **TIM[i]_CH[x]_CTRL.DSL** =1 and **TIM[i]_CH[x]_CNT** [1:0]=0b10 then *TSSM_OUT* [x:x] = *SHIFTOUT_IN* [x:x].

In case of registered or latched output mode the signal *SHIFTOUT_IN* [x:x] is selectable by **TIM[i]_CH[x]_CTRL.CNTS_SEL** . If **TIM[i]_CH[x]_CTRL.CNTS_SEL** =0 is set *F_OUT* [x:x] will be used for *SHIFTOUT_IN* [x:x], with **TIM[i]_CH[x]_CTRL.CNTS_SEL** =1 the signal *TIM_IN* [x:x] is in us for *SHIFTOUT_IN* [x:x].

13.4.2.7.2 External Capture TIM Serial Shift Mode (TSSM)

If external capture is enabled (**TIM[i]_CH[x]_CTRL.EXT_CAP_EN** =1), the external capture events will capture the actual values in **TIM[i]_CH[x]_GPR0** and **TIM[i]_CH[x]_GPR1** , reset the counter **TIM[i]_CH[x]_CNT** depending on **TIM[i]_CH[x]_CTRL.ISL** and issue a *TIM_NEWVAL_IRQ* . Functionality from previous table will be applied.

- The source which will be used as external capture event for TSSM mode is selectable by **TIM[i]_CH[x]_CNTS** [19:18]
- 0b00 = source selection by **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** is in use.
- 0b01 = *TDU_WORD_EVT* signal will be used as source.
- 0b10 = *TDU_FRAME_EVT* signal will be used as source.
- 0b11 = reserved

Operation is done depending on the shift clock, external capture signal, **TIM[i]_CH[x]_CTRL.DSL** , **TIM[i]_CH[x]_CTRL.ISL** bit and the input signal value defined in next table:

Table 25 Operation Depending on the Shift Clock, External Capture Signal, TIM[i]_CH[x]_CTRL.DSL, TIM[i]_CH[x]_CTRL.ISL and the Input Signal Value

Input signal *F_OUT* [x:x]	Shift clock	tssm_ext_capture	TIM[i]_CH[x]_CTRL.ISL	TIM[i]_CH[x]_CTRL.DSL	Action description
–	0	1	1	–	Do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0 **TIM[i]_CH[x]_CNT** [23:0]= **TIM[i]_CH[x]_CTRL.ECNT_RESET**
–	0	1	0	–	Do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0
value	1	1	1	0	**TIM[i]_CH[x]_CNT** [23:1]= **TIM[i]_CH[x]_CNT** [22:0]; **TIM[i]_CH[x]_CNT** [0:0]= value do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0 **TIM[i]_CH[x]_CNT** [23:0]= **TIM[i]_CH[x]_CTRL.ECNT_RESET**

Input signal *F_OUT* [x:x]	Shift clock	tssm_ext_capture	TIM[i]_CH[x]_CTRL.ISL	TIM[i]_CH[x]_CTRL.DSL	Action description
value	1	1	1	1	**TIM[i]_CH[x]_CNT** [22:0]= **TIM[i]_CH[x]_CNT** [23:1]; **TIM[i]_CH[x]_CNT** [23:23]= value do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0 **TIM[i]_CH[x]_CNT** [23:0]= **TIM[i]_CH[x]_CTRL.ECNT_RESET**
value	1	1	0	0	**TIM[i]_CH[x]_CNT** [23:1]= **TIM[i]_CH[x]_CNT** [22:0]; **TIM[i]_CH[x]_CNT** [0:0]= value do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0
value	1	1	0	1	**TIM[i]_CH[x]_CNT** [22:0]= **TIM[i]_CH[x]_CNT** [23:1]; **TIM[i]_CH[x]_CNT** [23:23]= value do capture; issue *TIM_NEWVAL_IRQ* ; **TIM[i]_CH[x]_CNTS** [15:8]=0

13.5 MAP Submodule Interface

The GTM-IP provides one dedicated TIM module TIM[0] where channel 0 to channel 5 are connected to the MAP sub-module described in chapter *20 "TIM0 Input Mapping Module (MAP)"* . There, the TIM[0] module channels provide the input signal level together with the actual filter value and the annotated time stamp for the edge together in a 49 bit wide signal to the MAP sub-module. This 49 bit wide data signal is marked as valid with a separate valid signal *TIM_MAP_DVAL* [x:x] (x∈{0, 1,..., 5}) (see *13.9.1 "TIM Map Data Interface"*).

Note:

With **TIM[i]_CH[x]_CTRL.TIM_EN** =1 and **TIM[i]_CH[x]_CTRL.GPR0_SEL** =0b00, the MAP interface starts operation, it is not dependent on the setting of the bit fields **TIM[i]_CH[x]_CTRL.TIM_MODE** , **TIM[i]_CH[x]_CTRL.ISL** and **TIM[i]_CH[x]_CTRL.DSL** . The bit field **TIM[i]_CH[x]_CTRL.TBU0_SEL** must also be considered.

Note:

While the MAP interface is in use the following guidelines have to be fulfilled, otherwise inconsistent filter values can be transferred.
Change **TIM[i]_CH[x]_FLT_RE** (with i=0) only between occurrence of rising and falling edge.
Change **TIM[i]_CH[x]_FLT_FE** (with i=0) only between occurrence of falling and rising edge.

13.6 TIM Software Reset of Channels

Each channel x is assigned a unique bit x in the configuration register **TIM[i]_RST** for software reset of the TIM[i] channel block.

Writing via the configuration interface the value 0b1 to the bitfield **TIM[i]_RST.RST_CH[x]** , will immediately reset the content of the following registers to the initial hardware reset state.

- **TIM[i]_CH[x]_CTRL**
- **TIM[i]_CH[x]_FLT_RE**
- **TIM[i]_CH[x]_FLT_FE**
- **TIM[i]_CH[x]_GPR0**
- **TIM[i]_CH[x]_GPR1**
- **TIM[i]_CH[x]_CNT**
- **TIM[i]_CH[x]_CNTS**
- **TIM[i]_CH[x]_IRQ_NOTIFY**

- **TIM[i]_CH[x]_IRQ_EN**
- **TIM[i]_CH[x]_IRQ_MODE**
- **TIM[i]_CH[x]_EIRQ_EN**
- **TIM[i]_CH[x]_TDUV**
- **TIM[i]_CH[x]_TDUC**
- **TIM[i]_CH[x]_ECNT**
- **TIM[i]_CH[x]_ECTRL**

Atomic reset of multiple channels is possible by writing a 1 on each associated bit. E.g: Writing 0x0F to register **TIM[i]_RST** (with i=0) will reset the TIM0 channel x (x∈{0, 1, 2, 3}).

13.7 TIM Configuration Registers Description

13.7.1 TIM[i]_CH[x]_CTRL

Description	TIM[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_CTRL
Address	$0x20000 * i + 0x80 * x + 0x824$
C-Name	GTM.CLS[i].TIM.CH[x].CTRL

Interface: MCS[i]

Name	TIM[i]_CH[x]_CTRL
Address	$0x80 * x + 0x824$
C-Name	

TIM_EN	
Description	TIM channel [x] enable
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Channel disabled 1 : Channel enabled

TIM_EN	
	Note: Enabling of the channel resets the registers **TIM[i]_CH[x]_ECNT** , **TIM[i]_CH[x]_CNT** , **TIM[i]_CH[x]_GPR0** , and **TIM[i]_CH[x]_GPR1** to their reset values. **Note:** After finishing the action in one-shot mode the **TIM[i]_CH[x]_CTRL.TIM_EN** bit is cleared automatically. Otherwise, the bit must be cleared manually.

TIM_MODE	
Description	TIM channel [x] mode
Loop	–
Bit Range	$[3:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0b000 : PWM Measurement Mode (TPWM) 0b001 : Pulse Integration Mode (TPIM) 0b010 : Input Event Mode (TIEM) 0b011 : Input Prescaler Mode (TIPM) 0b100 : Bit Compression Mode (TBCM) 0b101 : Gated Periodic Sampling Mode (TGPS) 0b110 : Serial Shift Mode (TSSM) 0b111 : unused
W-Coding	0b000 : PWM Measurement Mode (TPWM) 0b001 : Pulse Integration Mode (TPIM) 0b010 : Input Event Mode (TIEM) 0b011 : Input Prescaler Mode (TIPM) 0b100 : Bit Compression Mode (TBCM) 0b101 : Gated Periodic Sampling Mode (TGPS) 0b110 : Serial Shift Mode (TSSM) 0b111 : PWM Measurement Mode (TPWM) 0b000 should be used, 0b111 coding might be used for new defined functionality in future devices
	Note: The **TIM[i]_CH[x]_CTRL.TIM_MODE** register should not be changed while the TIM channel is enabled. **Note:** If the TIM channel is enabled and operating in TPWM or TPIM mode after the first valid edge defined by **TIM[i]_CH[x]_CTRL.DSL** has occurred, a reconfiguration of **TIM[i]_CH[x]_CTRL.DSL** , **TIM[i]_CH[x]_CTRL.ISL** , **TIM[i]_CH[x]_CTRL.TIM_MODE** will not change the channel behavior. Reading these bit fields after reconfiguration will show the newly configured settings but the initial channel behavior will not change. Only a disabling of the TIM channel by setting **TIM[i]_CH[x]_CTRL.TIM_EN** = 0 and reenabling with **TIM[i]_CH[x]_CTRL.TIM_EN** = 1 will change the channel operation mode.

OSM	
Description	One-shot mode

OSM	
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Continuous operation mode 1 : One-shot mode
	Note: After finishing the action in one-shot mode the **TIM[i]_CH[x]_CTRL.TIM_EN** bit is cleared automatically.

ARU_EN	
Description	TIM[i]_CH[x]_GPR0 and TIM[i]_CH[x]_GPR1 register values routed to ARU.
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	NARU == 1 0 : Registers content not routed 1 : Registers content routed
RW-Coding	NARU == 0 0 : Registers content not routed 1 : prohibited

CICTRL	
Description	Channel Input Control
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False

CICTRL	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Use signal *TIM_IN* [x:x] as input for channel x 1 : Use signal *TIM_IN* [x-1:x-1] as input for channel x (or *TIM_IN* [7:7] of instance [i-1])

TBU0_SEL	
Description	TIM_TBU_TS0 bits input select for TIM0_CH[x]_GPR0 and TIM0_CH[x]_GPR1
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$i == 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Use *TIM_TBU_TS0* [23:0] to store in **TIM[0]_CH[x]_GPR0** / **TIM[0]_CH[x]_GPR1** 1 : Use *TIM_TBU_TS0* [26:3] to store in **TIM[0]_CH[x]_GPR0** / **TIM[0]_CH[x]_GPR1**
	Note: This bit is only applicable for TIM0

GPR0_SEL	
Description	Selection for TIM[i]_CH[x]_GPR0 register
Loop	–
Bit Range	[9 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync

GPR0_SEL	
RW-Coding	TIM[i]_CH[x]_CTRL.EGPR0_SEL == 0 0b00 : Use *TIM_TBU_TS0* as input 0b01 : Use *TIM_TBU_TS1* as input 0b10 : Use *TIM_TBU_TS2* as input 0b11 : Use **TIM[i]_CH[x]_CNTS** as input; if TGPS mode in channel 0 is selected use Filter *F_OUT* as input
RW-Coding	TIM[i]_CH[x]_CTRL.EGPR0_SEL == 1 0b00 : Use **TIM[i]_CH[x]_ECNT** as input 0b01 : Use **TIM[i]_INP_VAL** as input 0b10 : Prohibited 0b11 : Prohibited

GPR1_SEL	
Description	Selection for TIM[i]_CH[x]_GPR1 register
Loop	–
Bit Range	[11 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	TIM[i]_CH[x]_CTRL.TIM_MODE != 4 && TIM[i]_CH[x]_CTRL.EGPR1_SEL == 0 0b00 : Use *TIM_TBU_TS0* as input 0b01 : Use *TIM_TBU_TS1* as input 0b10 : Use *TIM_TBU_TS2* as input 0b11 : Use **TIM[i]_CH[x]_CNT** as input
RW-Coding	TIM[i]_CH[x]_CTRL.TIM_MODE != 4 && TIM[i]_CH[x]_CTRL.EGPR1_SEL == 1 0b00 : Use **TIM[i]_CH[x]_ECNT** as input 0b01 : Use **TIM[i]_INP_VAL** as input 0b10 : Prohibited 0b11 : Prohibited
RW-Coding	TIM[i]_CH[x]_CTRL.TIM_MODE == 4 && TIM[i]_CH[x]_CTRL.EGPR1_SEL == 0 0b00 : Use Filter *F_OUT* as input 0b01 : Use Filter *F_OUT* as input 0b10 : Use Filter *F_OUT* as input 0b11 : Use Filter *F_OUT* as input
RW-Coding	TIM[i]_CH[x]_CTRL.TIM_MODE == 4 && TIM[i]_CH[x]_CTRL.EGPR1_SEL == 1 0b00 : Use Filter *F_OUT* as input 0b01 : Use **TIM[i]_INP_VAL** as input 0b10 : Prohibited 0b11 : Prohibited

CNTS_SEL	
Description	Selection for TIM[i]_CH[x]_CNTS register
Loop	–

CNTS_SEL	
Bit Range	[12 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Use **TIM[i]_CH[x]_CNT** register as input 1 : Use *TIM_TBU_TS0* as input
	Note: **TIM[i]_CH[x]_CTRL.CNTS_SEL** in TSSM mode selects the source signal for registered or latched shift out operation. 0 = Use *F_OUT* [x:x] 1 = Use *TIM_IN* [x:x]

DSL	
Description	Signal level control
Loop	–
Bit Range	[13 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Measurement starts with falling edge (low level measurement) 1 : Measurement starts with rising edge (high level measurement)
	Note: In **TIM[i]_CH[x]_CTRL.TIM_MODE** =0b110 (TSSM) the bit field **TIM[i]_CH[x]_CTRL.DSL** defines the shift direction. 0 = Shift left 1 = Shift right

ISL	
Description	Ignore signal level
Loop	–
Bit Range	[14 : 14]
Access Type	RW

ISL	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Use **TIM[i]_CH[x]_CTRL.DSL** bit for selecting active signal level (TIEM) 1 : Ignore **TIM[i]_CH[x]_CTRL.DSL** and treat both edges as active edge (TIEM)
	This bit is mode dependent and will have different meanings (see details in the TIM Channel mode description).

ECNT_RESET	
Description	Enables resetting of counter in certain modes
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	TIM[i]_CH[x]_CTRL.TIM_MODE == 0 0 : **TIM[i]_CH[x]_ECNT.ECNT** counter operating in wrap around mode, **TIM[i]_CH[x]_CNT.CNT** is reset on active input edge defined by **TIM[i]_CH[x]_CTRL.DSL** 1 : **TIM[i]_CH[x]_ECNT.ECNT** counter operating in wrap around mode, **TIM[i]_CH[x]_CNT.CNT** is reset on active and inactive input edge
RW-Coding	TIM[i]_CH[x]_CTRL.TIM_MODE == 5 0 : **TIM[i]_CH[x]_ECNT.ECNT** counter operating in wrap around mode 1 : **TIM[i]_CH[x]_ECNT.ECNT** counter is reset with periodic sampling
RW-Coding	TIM[i]_CH[x]_CTRL.TIM_MODE == 6 0 : Input shift value *TSSM_IN* [x:x]=0 is in use with **TIM[i]_CH[x]_CTRL.ISL** =1 1 : Input shift value *TSSM_IN* [x:x]=1 is in use with **TIM[i]_CH[x]_CTRL.ISL** =1
RW-Coding	TIM[i]_CH[x]_CTRL.TIM_MODE != 0 && TIM[i]_CH[x]_CTRL.TIM_MODE != 5 && TIM[i]_CH[x]_CTRL.TIM_MODE != 6 0 : **TIM[i]_CH[x]_ECNT.ECNT** counter operating in wrap around mode; 1 : **TIM[i]_CH[x]_ECNT.ECNT** counter operating in wrap around mode;

FLT_EN	
Description	Filter enable for channel [x]
Loop	–
Bit Range	[16 : 16]

FLT_EN	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Filter disabled and internal states are reset 1 : Filter enabled
	Note: If the filter is disabled all filter related units (including CSU) are bypassed, which means that the signal *F_IN* is directly routed to signal *F_OUT* .

FLT_CNT_FRQ	
Description	Filter counter frequency select
Loop	–
Bit Range	[18 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : *CCM[i]_CLK_RES* [0:0] resolution in use 0b01 : *CCM[i]_CLK_RES* [1:1] resolution in use 0b10 : *CCM[i]_CLK_RES* [6:6] resolution in use 0b11 : *CCM[i]_CLK_RES* [7:7] resolution in use

EXT_CAP_EN	
Description	Enables external capture mode. The selected TIM mode is only sensitive to external capture pulses the input event changes are ignored.
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

EXT_CAP_EN	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : External capture disabled 1 : External capture enabled

FLT_MODE_RE	
Description	Filter mode for rising edge
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Immediate edge propagation mode 1 : Individual deglitch mode

FLT_CTR_RE	
Description	Filter counter mode for rising edge
Loop	–
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	TIM[i]_CH[x]_CTRL.FLT_MODE_RE == 1 && TIM[i]_CH[x]_ECTRL.EFLT_CTR_RE == 0 0 : Up-Down Counter individual deglitch mode 1 : Hold Counter individual deglitch mode
RW-Coding	TIM[i]_CH[x]_CTRL.FLT_MODE_RE == 0 && TIM[i]_CH[x]_ECTRL.EFLT_CTR_RE == 0 0 : Immediate edge propagation mode 1 : Immediate edge propagation mode
RW-Coding	TIM[i]_CH[x]_CTRL.FLT_MODE_RE == 1 && TIM[i]_CH[x]_ECTRL.EFLT_CTR_RE == 1 0 : Reset Counter individual deglitch mode 1 : Prohibited

FLT_CTR_RE	
RW-Coding	TIM[i]_CH[x]_CTRL.FLT_MODE_RE == 0 && TIM[i]_CH[x]_ECTRL.EFLT_CTR_RE == 1 0 : Not applicable, initial value 1 : Prohibited

FLT_MODE_FE	
Description	Filter mode for falling edge
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Immediate edge propagation mode 1 : Individual deglitch mode

FLT_CTR_FE	
Description	Filter counter mode for falling edge
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	TIM[i]_CH[x]_CTRL.FLT_MODE_FE == 1 && TIM[i]_CH[x]_ECTRL.EFLT_CTR_FE == 0 0 : Up-Down Counter individual deglitch mode 1 : Hold Counter individual deglitch mode
RW-Coding	TIM[i]_CH[x]_CTRL.FLT_MODE_FE == 0 && TIM[i]_CH[x]_ECTRL.EFLT_CTR_FE == 0 0 : Immediate edge propagation mode 1 : Immediate edge propagation mode
RW-Coding	TIM[i]_CH[x]_CTRL.FLT_MODE_FE == 1 && TIM[i]_CH[x]_ECTRL.EFLT_CTR_FE == 1 0 : Reset Counter individual deglitch mode 1 : Prohibited
RW-Coding	TIM[i]_CH[x]_CTRL.FLT_MODE_FE == 0 && TIM[i]_CH[x]_ECTRL.EFLT_CTR_FE == 1 0 : Not applicable, initial value 1 : Prohibited

CLK_SEL	
Description	CMU clock source select for channel
Loop	–
Bit Range	[26 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	TIM[i]_CH[x]_ECTRL.ECLK_SEL == 0 0b000 : *CCM[i]_CLK_RES* [0:0] resolution in use 0b001 : *CCM[i]_CLK_RES* [1:1] resolution in use 0b010 : *CCM[i]_CLK_RES* [2:2] resolution in use 0b011 : *CCM[i]_CLK_RES* [3:3] resolution in use 0b100 : *CCM[i]_CLK_RES* [4:4] resolution in use 0b101 : *CCM[i]_CLK_RES* [5:5] resolution in use 0b110 : *CCM[i]_CLK_RES* [6:6] resolution in use 0b111 : *CCM[i]_CLK_RES* [7:7] resolution in use
RW-Coding	TIM[i]_CH[x]_ECTRL.ECLK_SEL == 1 0b000 : *TDU_SAMPLE_EVT* of TDU selected 0b001 : Prohibited 0b010 : Prohibited 0b011 : Prohibited 0b100 : Prohibited 0b101 : Prohibited 0b110 : Prohibited 0b111 : Prohibited

FR_ECNT_OFL	
Description	Extended Edge counter overflow behavior
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Overflow will be signaled on **TIM[i]_CH[x]_ECNT** bit width = 8 1 : Overflow will be signaled on **TIM[i]_CH[x]_ECNT** bit width (full range)

EGPR0_SEL	
Description	Extension of TIM[i]_CH[x]_CTRL.GPR0_SEL bit field
Loop	–
Bit Range	[28 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Details described in **TIM[i]_CH[x]_CTRL.GPR0_SEL** bit field.

EGPR1_SEL	
Description	Extension of TIM[i]_CH[x]_CTRL.GPR1_SEL bit field
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Details described in **TIM[i]_CH[x]_CTRL.GPR1_SEL** bit field.

TOCTRL	
Description	Timeout control
Loop	–
Bit Range	[31 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

TOCTRL	
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Timeout feature disabled 0b11 : Timeout feature enabled for both edges 0b01 : Timeout feature enabled for rising edge only 0b10 : Timeout feature enabled for falling edge only
	Note: It has to mention that writing of **TIM[i]_CH[x]_CTRL.TOCTRL** = 0 will every time stop the TDU, independent of the previous state of **TIM[i]_CH[x]_CTRL.TOCTRL** .

13.7.2 TIM[i]_CH[x]_FLT_RE

Description	TIM[i] channel [x] filter parameter 0 register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_FLT_RE
Address	$0x20000 * i + 0x80 * x + 0x81C$
C-Name	GTM.CLS[i].TIM.CH[x].FLT_RE

Interface: MCS[i]

Name	TIM[i]_CH[x]_FLT_RE
Address	$0x80 * x + 0x81C$
C-Name	

FLT_RE	
Description	Filter parameter for rising edge
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync

Note:

TIM[i]_CH[x]_FLT_RE has different meanings in the various filter modes.
Immediate edge propagation mode = acceptance time for rising edge
Individual deglitch time mode = deglitch time for rising edge

13.7.3 TIM[i]_CH[x]_FLT_FE

Description	TIM[i] channel [x] filter parameter 1 register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_FLT_FE
Address	$0x20000 * i + 0x80 * x + 0x820$
C-Name	GTM.CLS[i].TIM.CH[x].FLT_FE

Interface: MCS[i]

Name	TIM[i]_CH[x]_FLT_FE
Address	$0x80 * x + 0x820$
C-Name	

FLT_FE	
Description	Filter parameter for falling edge
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync

Note:
TIM[i]_CH[x]_FLT_FE has different meanings in the various filter modes.
Immediate edge propagation mode = acceptance time for falling edge
Individual deglitch time mode = deglitch time for falling edge

13.7.4 TIM[i]_CH[x]_GPR0

Description	TIM[i] channel [x] general purpose 0 register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	TIM[i]_CLK_ENABLE == 1

Interface: CPU

Name	TIM[i]_CH[x]_GPR0
Address	$0x20000 * i + 0x80 * x + 0x800$
C-Name	GTM.CLS[i].TIM.CH[x].GPR0

Interface: MCS[i]

Name	TIM[i]_CH[x]_GPR0
Address	$0x80 * x + 0x800$
C-Name	

GPR0	
Description	Input signal characteristic parameter 0
Loop	-
Bit Range	[23 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TIM[i]_CH[x]_CTRL.TIM_MODE != 6
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Note: The content of this register has different meaning for the TIM channels modes. The content directly depends on the bit fields **TIM[i]_CH[x]_CTRL.EGPR0_SEL** , **TIM[i]_CH[x]_CTRL.GPR0_SEL** . **Note:** The content of this register can only be written in TIM channel mode TSSM.

ECNT	
Description	Edge counter, value refers to TIM[i]_CH[x]_ECNT.ECNT[7:0].
Loop	-
Bit Range	[31 : 24]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

ECNT	
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Note: The **TIM[i]_CH[x]_GPR0.ECNT** counts every incoming filtered edge (rising and falling). The counter value is uneven in case of detected rising, and even in case of detected falling edge. Thus, the input signal level is part of the counter and can be obtained by bit 0 of **TIM[i]_CH[x]_GPR0.ECNT** . **Note:** The **TIM[i]_CH[x]_GPR0.ECNT** register is reset to its initial value when the channel is enabled. **Note:** Bit 0 depends on the input level coming from the filter unit and defines this value.

13.7.5 TIM[i]_CH[x]_GPR1

Description	TIM[i] channel [x] general purpose 1 register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_GPR1
Address	$0x20000 * i + 0x80 * x + 0x804$
C-Name	GTM.CLS[i].TIM.CH[x].GPR1

Interface: MCS[i]

Name	TIM[i]_CH[x]_GPR1
Address	$0x80 * x + 0x804$
C-Name	

GPR1	
Description	Input signal characteristic parameter 1
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TIM[i]_CH[x]_CTRL.TIM_MODE != 6 && TIM[i]_CH[x]_CTRL.TIM_MODE != 5
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync

GPR1	
	Note: The content of this register has different meaning for the TIM channels modes. The content directly depends on the bit fields **TIM[i]_CH[x]_CTRL.EGPR1_SEL** , **TIM[i]_CH[x]_CTRL.GPR1_SEL** . **Note:** In TBCM mode if **TIM[i]_CH[x]_CTRL.EGPR1_SEL** =1, **TIM[i]_CH[x]_CTRL.GPR1_SEL** = 0b01 then **TIM[i]_INP_VAL** is used as input in all other cases TIM Filter *F_OUT* is used as input and Bits **TIM[i]_CH[x]_GPR1** [2-3:8] = 0 **Note:** The content of this register can only be written in TIM channel mode TGPS and TSSM.

ECNT	
Description	Edge counter, value refers to TIM[i]_CH[x]_ECNT.ECNT[7:0].
Loop	–
Bit Range	$[31 : 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Note: The **TIM[i]_CH[x]_GPR1.ECNT** counts every incoming filtered edge (rising and falling). The counter value is uneven in case of detected rising, and even in case of detected falling edge. Thus, the input signal level is part of the counter and can be obtained by bit 0 of **TIM[i]_CH[x]_GPR1.ECNT** . **Note:** The **TIM[i]_CH[x]_GPR1.ECNT** register is reset to its initial value when the channel is enabled. **Note:** Bit 0 depends on the input level coming from the filter unit and defines this value.

13.7.6 TIM[i]_CH[x]_CNT

Description	TIM[i] channel [x] SMU counter register
Loop	$i = \{n : 0 <= n <= \mathrm{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_CNT
Address	$0x20000 * i + 0x80 * x + 0x808$

C–Name	GTM.CLS[i].TIM.CH[x].CNT

Interface: MCS[i]

Name	TIM[i]_CH[x]_CNT
Address	$0x80 * x + 0x808$
C–Name	

CNT	
Description	Actual SMU counter value
Loop	–
Bit Range	$[23 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Note: The meaning of this value depends on the configured mode: TPWM = actual duration of PWM signal. TPIM = actual duration of all pulses (sum of pulses). TIEM = actual number of received edges. TIPM = actual number of received edges. TGPS = elapsed time for periodic sampling. TSSM = shift data.

13.7.7 TIM[i]_CH[x]_CNTS

Description	TIM[i] channel [x] SMU shadow counter register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_CNTS
Address	$0x20000 * i + 0x80 * x + 0x810$
C–Name	GTM.CLS[i].TIM.CH[x].CNTS

Interface: MCS[i]

Name	TIM[i]_CH[x]_CNTS
Address	$0x80 * x + 0x810$

C-Name	

CNTS	
Description	Counter shadow register
Loop	-
Bit Range	[23 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TIM[i]_CH[x]_CTRL.TIM_MODE < 3
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Note: The content of this register has different meaning for the TIM channels modes. The content depends directly on the bit field **TIM[i]_CH[x]_CTRL.CNTS_SEL** . **Note:** The register **TIM[i]_CH[x]_CNTS** is only writable in TIPM,TBCM ,TGPS and TSSM mode.

ECNT	
Description	Edge counter
Loop	-
Bit Range	[31 : 24]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Note: The **TIM[i]_CH[x]_CNTS.ECNT** counts every incoming filtered edge (rising and falling). The counter value is uneven in case of detected rising, and even in case of detected falling edge. Thus, the input signal level is part of the counter and can be obtained by bit 0 of **TIM[i]_CH[x]_CNTS.ECNT** . **Note:** The **TIM[i]_CH[x]_CNTS.ECNT** register is reset to its initial value when the channel is enabled. **Note:** Bit 0 depends on the input level coming from the filter unit and defines the reset value immediately.

13.7.8 TIM[i]_CH[x]_IRQ_NOTIFY

Description	TIM[i] channel [x] interrupt notification register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_IRQ_NOTIFY
Address	$0x20000 * i + 0x80 * x + 0x82C$
C-Name	GTM.CLS[i].TIM.CH[x].IRQ_NOTIFY

Interface: MCS[i]

Name	TIM[i]_CH[x]_IRQ_NOTIFY
Address	$0x80 * x + 0x82C$
C-Name	

NEWVAL	
Description	New measurement value detected by the channel [x] ([x]:0...m-1)
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

ECNTOFL	
Description	ECNT counter overflow of channel [x] ([x]:0...m-1)
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	True

ECNTOFL	
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R–Coding	0 : No interrupt raised 1 : Interrupt was raised
W–Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

CNTOFL	
Description	SMU TIM[i]_CH[x]_CNT counter overflow of channel [x], ([x]:0...m−1)
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R–Coding	0 : No interrupt raised 1 : Interrupt was raised
W–Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

GPROFL	
Description	TIM[i]_CH[x]_GPR0 and TIM[i]_CH[x]_GPR1 data overflow, old data not read out before new data has arrived at input pin, ([x]:0...m−1)
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–

GPROFL	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R–Coding	0 : No interrupt raised 1 : Interrupt was raised
W–Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

TODET	
Description	Timeout reached for input signal of channel [x], ([x]:0...m–1)
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R–Coding	0 : No interrupt raised 1 : Interrupt was raised
W–Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

GLITCHDET	
Description	Glitch detected on channel [x], ([x]:0...m–1)
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–

GLITCHDET	
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

13.7.9 TIM[i]_CH[x]_IRQ_EN

Description	TIM[i] channel [x] interrupt enable register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_IRQ_EN
Address	$0x20000 * i + 0x80 * x + 0x830$
C-Name	GTM.CLS[i].TIM.CH[x].IRQ_EN

Interface: MCS[i]

Name	TIM[i]_CH[x]_IRQ_EN
Address	$0x80 * x + 0x830$
C-Name	

NEWVAL_IRQ_EN	
Description	TIM_NEWVAL_IRQ[x:x] interrupt enable
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync

NEWVAL_IRQ_EN	
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

ECNTOFL_IRQ_EN	
Description	TIM_ECNTOFL_IRQ[x:x] interrupt enable
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

CNTOFL_IRQ_EN	
Description	TIM_CNTOFL_IRQ[x:x] interrupt enable
Loop	-
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

GPROFL_IRQ_EN	
Description	TIM_GPROFL_IRQ[x:x] interrupt enable
Loop	-
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

GPROFL_IRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

TODET_IRQ_EN	
Description	TIM_TODET_IRQ[x:x] interrupt enable
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

GLITCHDET_IRQ_EN	
Description	TIM_GLITCHDET_IRQ[x:x] interrupt enable
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP 1 : Enable interrupt, interrupt is visible outside GTM-IP

13.7.10 TIM[i]_CH[x]_IRQ_FORCINT

Description	TIM[i] channel [x] force interrupt register

Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_IRQ_FORCINT
Address	$0x20000 * i + 0x80 * x + 0x834$
C-Name	GTM.CLS[i].TIM.CH[x].IRQ_FORCINT

Interface: MCS[i]

Name	TIM[i]_CH[x]_IRQ_FORCINT
Address	$0x80 * x + 0x834$
C-Name	

TRG_NEWVAL	
Description	Trigger the bit TIM_CH[x]_IRQ_NOTIFY.NEWVAL by software
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 5, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No interrupt triggering 1 : Force setting of **TIM_CH[x]_IRQ_NOTIFY.NEWVAL**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_ECNTOFL	
Description	Trigger the bit TIM_CHx_IRQ_NOTIFY.ECNTOFL by software
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	False
Multithread	False

TRG_ECNTOFL	
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 5, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No interrupt triggering 1 : Force setting of **TIM_CH[x]_IRQ_NOTIFY.ECNTOFL**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_CNTOFL	
Description	Trigger the bit TIM_CH[x]_IRQ_NOTIFY.CNTOFL by software
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 5, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No interrupt triggering 1 : Force setting of **TIM_CH[x]_IRQ_NOTIFY.CNTOFL**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_GPROFL	
Description	Trigger the bit TIM_CH[x]_IRQ_NOTIFY.GPROFL by software
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear

TRG_GPROFL	
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 5, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No interrupt triggering 1 : Force setting of **TIM_CH[x]_IRQ_NOTIFY.GPROFL**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_TODET	
Description	Trigger the bit TIM_CH[x]_IRQ_NOTIFY.TODET by software
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 5, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No interrupt triggering 1 : Force setting of **TIM_CH[x]_IRQ_NOTIFY.TODET**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_GLITCHDET	
Description	Trigger the bit TIM_CH[x]_IRQ_NOTIFY.GLITCHDET by software
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–

TRG_GLITCHDET	
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 5, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No interrupt triggering 1 : Force setting of **TIM_CH[x]_IRQ_NOTIFY.GLITCHDET**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

13.7.11 TIM[i]_CH[x]_IRQ_MODE

Description	TIM[i] channel [x] interrupt mode configuration register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_IRQ_MODE
Address	$0x20000 * i + 0x80 * x + 0x838$
C-Name	GTM.CLS[i].TIM.CH[x].IRQ_MODE

Interface: MCS[i]

Name	TIM[i]_CH[x]_IRQ_MODE
Address	$0x80 * x + 0x838$
C-Name	

IRQ_MODE	
Description	IRQ mode selection of channel [x]
Loop	-
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	-

IRQ_MODE	
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

13.7.12 TIM[i]_RST

Description	TIM[i] global software reset register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_RST
Address	$0x20000 * i + 0xC08$
C-Name	GTM.CLS[i].TIM.RST

Interface: MCS[i]

Name	TIM[i]_RST
Address	$0xC08$
C-Name	

RST_CH[x]	
Description	Software reset of channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0 : No action 1 : Reset channel [x]

RST_CH[x]	
R-Coding	0 : No Status
	Note: This bit is cleared automatically after write by CPU. The channel [x] registers are set to their reset values and channel [x] operation is stopped immediately.

13.7.13 TIM[i]_IN_SRC

Description	TIM[i] AUX IN source selection register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_IN_SRC
Address	$0x20000 * i + 0xC04$
C-Name	GTM.CLS[i].TIM.IN_SRC

Interface: MCS[i]

Name	TIM[i]_IN_SRC
Address	$0xC04$
C-Name	

VAL_[x]	
Description	Value to be fed to channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x * 4 + 1 : x * 4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : don't care, bits will not be changed 0b01 : Change state to 0 0b10 : Change state to 1 0b11 : don't care, bits will not be changed
R-Coding	0b00 : State is 0 0b01 : unused 0b10 : unused 0b11 : State is 1

VAL_[x]	
	Multithread encoding in use (**TIM[i]_IN_SRC.VAL_[x]** [1:1] defines the state of the signal) Function depends on the combination of **TIM[i]_IN_SRC.VAL_[x]** [1:1] and **TIM[i]_IN_SRC.MODE_[x]** [1:1] see **TIM[i]_IN_SRC.MODE_[x]** description. **Note:** Any read access to a **TIM[i]_IN_SRC.VAL_[x]** bit field will always result in a value 00 or 11 indicating current state. A modification of the state is only performed with the values 01 and 10. Writing the values 00 and 11 is always ignored.

MODE_[x]	
Description	Input source to channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x * 4 + 3 : x * 4 + 2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W–Coding	0b00 : don't care, bits will not be changed 0b01 : Change state to 0 0b10 : Change state to 1 0b11 : don't care, bits will not be changed
R–Coding	0b00 : State is 0 0b01 : unused 0b10 : unused 0b11 : State is 1
	Multithread encoding in use (**TIM[i]_IN_SRC.MODE_[x]** [1:1] defines the state of the signal) Function table: **TIM[i]_IN_SRC.MODE_[x]** [1:1] = 0, **TIM[i]_IN_SRC.VAL_[x]** [1:1] = 0: The input signal defined by bit field **TIM[i]_CH[x]_CTRL.CICTRL** of the TIM channel is used as input source. **TIM[i]_IN_SRC.MODE_[x]** [1:1] = 0, **TIM[i]_IN_SRC.VAL_[x]** [1:1] = 1: The signal *AUX_IN* [x:x] of the TIM channel is used as input source. **TIM[i]_IN_SRC.MODE_[x]** [1:1] = 1: The state **TIM[i]_IN_SRC.VAL_[x]** [1:1] defines the input level for the TIM channel. **Note:** Any read access to a **TIM[i]_IN_SRC.MODE_[x]** bit field will always result in a value 00 or 11 indicating current state. A modification of the state is only performed with the values 01 and 10. Writing the values 00 and 11 is always ignored.

13.7.14 TIM[i]_CH[x]_EIRQ_EN

Description	TIM[i] channel [x] error interrupt enable register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	TIM[i]_CLK_ENABLE == 1

Interface: CPU

Name	TIM[i]_CH[x]_EIRQ_EN
Address	$0x20000 * i + 0x80 * x + 0x83C$
C-Name	GTM.CLS[i].TIM.CH[x].EIRQ_EN

Interface: MCS[i]

Name	TIM[i]_CH[x]_EIRQ_EN
Address	$0x80 * x + 0x83C$
C-Name	

NEWVAL_EIRQ_EN	
Description	TIM_NEWVAL[x]_EIRQ error interrupt enable
Loop	-
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM-IP 1 : Enable error interrupt, error interrupt is visible outside GTM-IP

ECNTOFL_EIRQ_EN	
Description	TIM_ECNTOFL_IRQ[x:x] interrupt enable
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync

ECNTOFL_EIRQ_EN	
RW-Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM-IP 1 : Enable error interrupt, error interrupt is visible outside GTM-IP

CNTOFL_EIRQ_EN	
Description	TIM_CNTOFL_IRQ[x:x] interrupt enable
Loop	-
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM-IP 1 : Enable error interrupt, error interrupt is visible outside GTM-IP

GPROFL_EIRQ_EN	
Description	TIM_GPROFL_IRQ[x:x] interrupt enable
Loop	-
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM-IP 1 : Enable error interrupt, error interrupt is visible outside GTM-IP

TODET_EIRQ_EN	
Description	TIM_TODET_IRQ[x:x] interrupt enable
Loop	-
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

TODET_EIRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP 1 : Enable error interrupt, error interrupt is visible outside GTM–IP

GLITCHDET_EIRQ_EN	
Description	TIM_GLITCHDET_IRQ[x:x] interrupt enable
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP 1 : Enable error interrupt, error interrupt is visible outside GTM–IP

13.7.15 TIM[i]_CH[x]_TDUV

Description	TIM[i] channel [x] TDU control register
Loop	$i = \{n : 0 <= n <= \mathrm{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_TDUV
Address	$0x20000 * i + 0x80 * x + 0x818$
C–Name	GTM.CLS[i].TIM.CH[x].TDUV

Interface: MCS[i]

Name	TIM[i]_CH[x]_TDUV
Address	$0x80 * x + 0x818$
C–Name	

TOV	
Description	Time out compare value slice0 for channel [y]
Loop	–
Bit Range	[7 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Compare value for **TIM[i]_CH[x]_TDUC.TO_CNT**

TOV1	
Description	Time out compare value slice1 for channel [y]
Loop	–
Bit Range	[15 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Compare value for **TIM[i]_CH[x]_TDUC.TO_CNT1**

TOV2	
Description	Time out compare value slice2 for channel [y]
Loop	–
Bit Range	[23 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

TOV2	
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	If bitfield TIM[i]_CH[x]_TDUV.SLICING ˜= 3: Compare value for **TIM[i]_CH[x]_TDUC.TO_CNT2** If bitfield TIM[i]_CH[x]_TDUV.SLICING == 3: **TIM[i]_CH[x]_TDUV.TOV2** operates as a shadow register for **TIM[i]_CH[x]_TDUC.TO_CNT**

SLICING	
Description	Cascading of counter slices
Loop	–
Bit Range	[25 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW–Coding	TIM[i]_CH[x]_ECTRL.USE_LUT == 0 0b00 : Combine slice2,slice1,slice0 to 1x24 bit counter 0b01 : Combine slice1,slice0 to 1x16bit counter use slice2 as 1x8 bit counter 0b10 : Use slice2,slice1,slice0 as 3x8 bit counter 0b11 : Use slice1,slice0 as 2x8 bit counter
RW–Coding	TIM[i]_CH[x]_ECTRL.USE_LUT != 0 0b01 : Combine slice1,slice0 to 1x16bit slice2 not usable 0b10 : Use slice1,slice0 as 2x8 bit counter slice2 not usable 0b11 : Use slice1,slice0 as 2x8 bit counter

TCS_USE_SAMPLE_EVT	
Description	Use TDU_SAMPLE_EVT as Timeout Clock
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync

TCS_USE_SAMPLE_EVT	
RW-Coding	0 : *CCM[i]_CLK_RES* selected by **TIM[i]_CH[x]_TDUV.TCS** is in use by **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT2** 1 : *CCM[i]_CLK_RES* selected by **TIM[i]_CH[x]_TDUV.TCS** is in use by **TIM[i]_CH[x]_TDUC.TO_CNT2** ; *TDU_SAMPLE_EVT* is in use by **TIM[i]_CH[x]_TDUC.TO_CNT**

TDU_SAME_CNT_CLK	
Description	Define clocking of TO_CNT, TO_CNT1
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : **TIM[i]_CH[x]_TDUC.TO_CNT** clock selected by (TCS, **TIM[i]_CH[x]_TDUV.TCS_USE_SAMPLE_EVT**); **TIM[i]_CH[x]_TDUC.TO_CNT1** clocked on tdu_word_event 1 : **TIM[i]_CH[x]_TDUC.TO_CNT1** uses same clock as **TIM[i]_CH[x]_TDUC.TO_CNT**

TCS	
Description	Timeout clock selection
Loop	–
Bit Range	[30 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0b000 : *CCM[i]_CLK_RES* [0:0] resolution in use 0b001 : *CCM[i]_CLK_RES* [1:1] resolution in use 0b010 : *CCM[i]_CLK_RES* [2:2] resolution in use 0b011 : *CCM[i]_CLK_RES* [3:3] resolution in use 0b100 : *CCM[i]_CLK_RES* [4:4] resolution in use 0b101 : *CCM[i]_CLK_RES* [5:5] resolution in use 0b110 : *CCM[i]_CLK_RES* [6:6] resolution in use 0b111 : *CCM[i]_CLK_RES* [7:7] resolution in use

13.7.16 TIM[i]_CH[x]_TDUC

Description	TIM[i] channel [x] TDU counter register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_TDUC
Address	$0x20000 * i + 0x80 * x + 0x814$
C-Name	GTM.CLS[i].TIM.CH[x].TDUC

Interface: MCS[i]

Name	TIM[i]_CH[x]_TDUC
Address	$0x80 * x + 0x814$
C-Name	

TO_CNT	
Description	Current Timeout value slice0 for channel [y]
Loop	-
Bit Range	$[7 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	TIM[i]_CH[x]_CTRL.TOCTRL > 0
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	If bitfield TIM[i]_CH[x]_TDUV.SLICING ˜= 3: counter will be reset to 0x00 on **TIM[i]_CH[x]_ECTRL.TDU_RESYNC** condition If bitfield TIM[i]_CH[x]_TDUV.SLICING == 3: counter will be loaded with **TIM[i]_CH[x]_TDUV.TOV2** on **TIM[i]_CH[x]_ECTRL.TDU_RESYNC** condition

TO_CNT1	
Description	Current Timeout value slice1 for channel [y]
Loop	-
Bit Range	$[15 : 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-

TO_CNT1	
Condition	–
Initial value	0
Protect Enable Cond	TIM[i]_CH[x]_CTRL.TOCTRL > 0
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Counter will be reset to 0x00 on **TIM[i]_CH[x]_ECTRL.TDU_RESYNC** condition

TO_CNT2	
Description	Current Timeout value slice2 for channel [y]
Loop	–
Bit Range	$[23 : 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	TIM[i]_CH[x]_CTRL.TOCTRL > 0 && TIM[i]_CH[x]_ECTRL.USE_LUT == 0
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Counter will be reset to 0x00 on **TIM[i]_CH[x]_ECTRL.TDU_RESYNC** condition

Note:
The register **TIM[i]_CH[x]_TDUC** is writable if Timeout unit is disabled (**TIM[i]_CH[x]_CTRL.TOCTRL** =0b00).

Note:
If **TIM[i]_CH[x]_ECTRL.USE_LUT** != 0b00 (input signal generation by lookup table) the bit field **TIM[i]_CH[x]_TDUC.TO_CNT2** is writable at any time, **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** will not be changed.

13.7.17 TIM[i]_CH[x]_ECNT

Description	TIM[i] channel [x] SMU edge counter register
Loop	$i = \{n : 0 <= n <= \mathrm{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_ECNT
Address	$0x20000 * i + 0x80 * x + 0x80C$
C–Name	GTM.CLS[i].TIM.CH[x].ECNT

Interface: MCS[i]

Name	TIM[i]_CH[x]_ECNT
Address	$0x80 * x + 0x80C$
C-Name	

ECNT	
Description	Edge counter
Loop	-
Bit Range	$[15 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Note: The **TIM[i]_CH[x]_ECNT** register is reset to its initial value when the channel is enabled. Bit 0 depends on the input level coming from the filter unit and defines this value. **Note:** If TIM channel is disabled the content of **TIM[i]_CH[x]_ECNT.ECNT** gets frozen and a read will auto clear the bits [15:1]. Further read accesses to **TIM[i]_CH[x]_ECNT.ECNT** will show on Bit 0 the actual input signal value of the channel.

13.7.18 TIM[i]_CH[x]_ECTRL

Description	TIM[i] channel [x] extended control register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_CH[x]_ECTRL
Address	$0x20000 * i + 0x80 * x + 0x828$
C-Name	GTM.CLS[i].TIM.CH[x].ECTRL

Interface: MCS[i]

Name	TIM[i]_CH[x]_ECTRL
Address	$0x80 * x + 0x828$
C-Name	

EXT_CAP_SRC	
Description	Defines selected source for triggering the TIM_EXT_CAPTURE functionality.
Loop	–
Bit Range	[3 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0b0000 : *NEWVAL_EVT* of following channel selected 0b0001 : *AUX_IN* selected 0b0010 : *CNTOFL_EVT* of following channel selected 0b0011 : If **TIM[i]_CH[x]_CTRL.CICTRL** = 1 : use signal *TIM_IN* [x:x] as input for channel x; if **TIM[i]_CH[x]_CTRL.CICTRL** = 0 : use signal *TIM_IN* [x−1:x−1] as input for channel x (or *TIM_IN* [7:7] of instance [i−1] if x is 0) 0b0100 : *ECNTOFL_EVT* of following channel selected 0b0101 : *TODET_EVT* of following channel selected 0b0110 : *GLITCHDET_EVT* of following channel selected 0b0111 : *GPROFL_EVT* of following channel selected 0b1000 : *CCM[i]_CLK_RES* selected by **TIM[i]_CH[x]_CTRL.CLK_SEL** of following channel 0b1001 : *REDGE_DET* of following channel selected 0b1010 : *FEDGE_DET* of following channel selected 0b1011 : Logical or of (*FEDGE_DET* , *REDGE_DET*) of following channel selected 0b1100 : *TDU_SAMPLE_EVT* of local TDU selected 0b1101 : *TDU_WORD_EVT* of local TDU selected 0b1110 : *TDU_FRAME_EVT* of local TDU selected

USE_PREV_TDU_IN	
Description	Select input data source for TDU
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Use input data of local filter for TDU 1 : Use input data of previous channel (after filter unit) for TDU

TODET_IRQ_SRC	
Description	Selection of source for TIM_TODET_IRQ
Loop	–
Bit Range	[7 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Use *TDU_TIMEOUT_EVT* 0b01 : Use *TDU_WORD_EVT* 0b10 : Use *TDU_FRAME_EVT* 0b11 : Use *TDU_SAMPLE_EVT*
	Note: With **TIM[i]_CH[x]_ECTRL.TODET_IRQ_SRC** =0b00 the bit *ARU_OUT* [50:50] will be driven by channel x signal *TO_DET* , if **TIM[i]_CH[x]_ECTRL.TODET_IRQ_SRC** !=0b00 *ARU_OUT* [50:50] will be 0.

TDU_START	
Description	Defines condition which will start the TDU unit.
Loop	–
Bit Range	[10 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0b000 : Start once immediate on *TDU_START_000_EVENT* (see Note) 0b001 : Start once with occurrence of first cmu_clk selected by **TIM[i]_CH[x]_CTRL.CLK_SEL** when measure unit is enabled by **TIM[i]_CH[x]_CTRL.TIM_EN** =1 0b010 : Start once with occurrence of first active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; restart on *TDU_FRAME_EVT* if TDU is stopped 0b011 : Start once with occurrence of first active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** 0b100 : Start/restart with occurrence of external capture event (if TDU is stopped, restart again) 0b101 : Start/restart with occurrence of first cmu_clk selected by **TIM[i]_CH[x]_CTRL.CLK_SEL** when measure unit is enabled by **TIM[i]_CH[x]_CTRL.TIM_EN** =1 (if TDU is stopped, restart again) 0b110 : Start once with occurrence of external capture event; restart on *TDU_FRAME_EVT* if TDU is stopped 0b111 : Start/restart with occurrence of first active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** (if TDU is stopped, restart again)

TDU_START	
	Note: *TDU_START_000_EVENT* is defined as: Each writing of **TIM[i]_CH[x]_CTRL.TOCTRL** != 0 (independent of current **TIM[i]_CH[x]_CTRL.TOCTRL**) while **TIM[i]_CH[x]_ECTRL.TDU_START** =0b000 and TDU is stopped (initially or stopped by **TIM[i]_CH[x]_ECTRL.TDU_STOP** event). This event will last 1 system clock cycle. **Note:** In mode **TIM[i]_CH[x]_TDUV.SLICING** =0b11 every start/restart will load the **TIM[i]_CH[x]_TDUC.TO_CNT** with value **TIM[i]_CH[x]_TDUV.TOV2** .

TDU_STOP	
Description	Defines condition which will stop the TDU unit.
Loop	–
Bit Range	[14 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	TIM[i]_CH[x]_TDUV.SLICING == 2 || TIM[i]_CH[x]_TDUV.SLICING == 3 0b000 : Immediate stop counting of TDU on *TDU_TOCTRL_0_EVENT* (see Note) 0b001 : Stop counting of TDU on *TDU_WORD_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note) 0b010 : Stop counting of TDU on *TDU_FRAME_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note) 0b011 : Stop counting of TDU on *TDU_TIMEOUT_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note) 0b100 : Stop counting of TDU on external capture event or on *TDU_TOCTRL_0_EVENT* (see Note) 0b101 : Stop counting of **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note); stop counting of **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note); stop counting of **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_TOCTRL_0_EVENT* (see Note); 0b110 : Increase **TIM[i]_CH[x]_TDUC.TO_CNT** only if *F_OUT* [x:x] = 0; stop counting of **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note); increase **TIM[i]_CH[x]_TDUC.TO_CNT1** only if *F_OUT* [x:x] = 1; stop counting of **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note); stop counting of **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_TOCTRL_0_EVENT* (see Note);
RW-Coding	TIM[i]_CH[x]_TDUV.SLICING == 0 || TIM[i]_CH[x]_TDUV.SLICING == 1 0b000 : Immediate stop counting of TDU on *TDU_TOCTRL_0_EVENT* (see Note) 0b001 : Stop counting of TDU on *TDU_WORD_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note) 0b010 : Stop counting of TDU on *TDU_FRAME_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note) 0b011 : Stop counting of TDU on *TDU_TIMEOUT_EVT* or on *TDU_TOCTRL_0_EVENT* (see Note) 0b100 : Stop counting of TDU on external capture event or on *TDU_TOCTRL_0_EVENT* (see Note)
	Note: *TDU_TOCTRL_0_EVENT* is defined as: Each writing of **TIM[i]_CH[x]_CTRL.TOCTRL** = 0 (independent of current **TIM[i]_CH[x]_CTRL.TOCTRL**) while TDU is started. This event will last 1 system clock cycle.

TDU_RESYNC	
Description	Defines condition which will resynchronize the TDU unit.
Loop	–
Bit Range	[19 : 16]

TDU_RESYNC	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	TIM[i]_CH[x]_TDUV.SLICING == 0 0b0000 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** or *TDU_TIMEOUT_EVT* or on *TDU_START_000_EVENT* (see Note); reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_TIMEOUT_EVT* or on *TDU_START_000_EVENT* (see Note); reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; 0b0001 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; 0b0010 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b0011 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b0100 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b0101 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b0110 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b0111 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b1000 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** , **TIM[i]_CH[x]_TDUC.TO_CNT2** on event selected by **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** ; 0b1001 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; 0b1010 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b1011 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on TDU_WORD_EVT; 0b1100 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b1101 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b1110 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b1111 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ;

TDU_RESYNC	
RW-Coding	TIM[i]_CH[x]_TDUV.SLICING == 1 0b0000 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** or *TDU_TIMEOUT_EVT* or on *TDU_START_000_EVENT* (see Note); reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_TIMEOUT_EVT* or on *TDU_START_000_EVENT* (see Note); reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; 0b0001 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; 0b0010 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b0011 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b0100 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_FRAME_EVT* ; 0b0101 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_FRAME_EVT* ; 0b0110 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_FRAME_EVT* ; 0b0111 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_FRAME_EVT* ; 0b1000 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** , **TIM[i]_CH[x]_TDUC.TO_CNT2** on event selected by **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; 0b1001 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; 0b1010 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b1011 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on TDU_WORD_EVT; 0b1100 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_FRAME_EVT* ; 0b1101 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_FRAME_EVT* ; 0b1110 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_FRAME_EVT* ; 0b1111 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_FRAME_EVT* ;
RW-Coding	TIM[i]_CH[x]_TDUV.SLICING == 2 0b0000 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** or *TDU_TIMEOUT_EVT* or on *TDU_START_000_EVENT* (see Note); reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_TIMEOUT_EVT* or on *TDU_START_000_EVENT* (see Note); if **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; 0b0001 : If **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; 0b0010 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b0011 : If **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b0100 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; ▽

TDU_RESYNC	
	△ 0b0101 : If **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b0110 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b0111 : If **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b1000 : Reset counters **TIM[i]_CH[x]_TDUC.TO_CNT** , **TIM[i]_CH[x]_TDUC.TO_CNT1** , **TIM[i]_CH[x]_TDUC.TO_CNT2** on event selected by **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; 0b1001 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; if **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; 0b1010 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b1011 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; if **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; 0b1100 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b1101 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; if **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b1110 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ; 0b1111 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on *TDU_SAMPLE_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; if **TIM[i]_CH[x]_CTRL.TOCTRL** [0:0] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on rising input edge; if **TIM[i]_CH[x]_CTRL.TOCTRL** [1:1] = 1 then reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on falling input edge; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* ;
RW-Coding	TIM[i]_CH[x]_TDUV.SLICING == 3 0b0000 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** or *TDU_WORD_EVT* or on *TDU_START_000_EVENT* (see Note); reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** or TDU_WORD_EVT or on *TDU_START_000_EVENT* (see Note) 0b0001 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** 0b0010 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on *TDU_WORD_EVT* 0b0011 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on *TDU_WORD_EVT* or on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** 0b0100 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* 0b0101 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* or each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; 0b0110 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* 0b0111 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on *TDU_WORD_EVT* or on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* or each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; 0b1000 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on event selected by **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on event selected by **TIM[i]_CH[x]_ECTRL.EXT_CAP_SRC** ▽

TDU_RESYNC	
	△ 0b1001 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** 0b1010 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* 0b1011 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** else reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** 0b1110 : Reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* 0b1111 : Load counter **TIM[i]_CH[x]_TDUC.TO_CNT** with **TIM[i]_CH[x]_TDUV.TOV2** on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** else reset counter **TIM[i]_CH[x]_TDUC.TO_CNT** on *TDU_WORD_EVT* ; reset counter **TIM[i]_CH[x]_TDUC.TO_CNT1** on *TDU_FRAME_EVT* or on each active edge selected by **TIM[i]_CH[x]_CTRL.TOCTRL** ;
	Note: *TDU_START_000_EVENT* is defined as: Each writing of **TIM[i]_CH[x]_CTRL.TOCTRL** != 0 (independent of current **TIM[i]_CH[x]_CTRL.TOCTRL**) while **TIM[i]_CH[x]_ECTRL.TDU_START** =0b000 and TDU is stopped (initially or stopped by **TIM[i]_CH[x]_ECTRL.TDU_STOP** event). This event will last 1 system clock cycle.

USE_LUT	
Description	Generate filter input by lookup table
Loop	–
Bit Range	[23 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Lookup table not in use, *LUT_IN0* [x:x] used as filter input 0b01 : Use 3 bit lookup table with index = *TIM_EXT_CAPTURE* [x:x] & *LUT_IN1* [x:x] & *LUT_IN0* [x:x]. Filter input is defined by **TIM[i]_CH[x]_TDUC.TO_CNT2** . 0b10 : Use 3 bit lookup table with index = *FOUT_PREV* [x:x] & *LUT_IN1* [x:x] & *LUT_IN0* [x:x]. Filter input is defined by **TIM[i]_CH[x]_TDUC.TO_CNT2** . 0b11 : Use 3 bit lookup table with index = *TSSM_OUT* [x:x] & *LUT_IN1* [x:x] & *LUT_IN0* [x:x]. Filter input is defined by **TIM[i]_CH[x]_TDUC.TO_CNT2** .

EFLT_CTR_RE	
Description	Extension of bit field TIM[i]_CH[x]_CTRL.FLT_CTR_RE
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

EFLT_CTR_RE	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Details described in **TIM[i]_CH[x]_CTRL.FLT_CTR_RE** bit field.

EFLT_CTR_FE	
Description	Extension of bit field TIM[i]_CH[x]_CTRL.FLT_CTR_FE
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Details described in **TIM[i]_CH[x]_CTRL.FLT_CTR_FE** bit field.

SWAP_CAPTURE	
Description	Swap point in time of capturing TIM[i]_CH[x]_CNTS and TIM[i]_CH[x]_GPR1
Loop	–
Bit Range	[28 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Inactive edge will capture data in **TIM[i]_CH[x]_CNTS** ; *TIM_NEWVAL_IRQ* event will capture data in **TIM[i]_CH[x]_GPR1** 1 : Swap time of capture: inactive edge will capture data in **TIM[i]_CH[x]_GPR1** ; *TIM_NEWVAL_IRQ* event will capture data in **TIM[i]_CH[x]_CNTS**
	This bit is only applicable in TPWM and TPIM mode. Set to 0 in all other modes.

IMM_START	
Description	Start the measurement immediately
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Start with first active edge the measurement 1 : Start immediately after enable (**TIM[i]_CH[x]_CTRL.TIM_EN** =1) the measurement
	This bit is only applicable in TPWM and TPIM mode. Set to 0 in all other modes.

ECLK_SEL	
Description	Extension of bit field TIM[i]_CH[x]_CTRL.CLK_SEL
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
	Details described in **TIM[i]_CH[x]_CTRL.CLK_SEL** bit field.

USE_PREV_CH_IN	
Description	Select input data source for TIM channel
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

USE_PREV_CH_IN	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIM[i]_RESET_CH[x]: sync
RW-Coding	0 : Use input data of local filter unit for channel measurements 1 : Use input data of previous channel (after filter unit) for channel measurements

13.7.19 TIM[i]_INP_VAL

Description	TIM[i] input value observation register
Loop	$i = \{n : 0 <= n <= \text{NTIM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TIM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIM[i]_INP_VAL
Address	$0x20000 * i + 0xC00$
C-Name	GTM.CLS[i].TIM.INP_VAL

Interface: MCS[i]

Name	TIM[i]_INP_VAL
Address	$0xC00$
C-Name	

F_OUT[x]	
Description	Signal channel [x] after TIM FLT unit
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Input data is 0 1 : Input data is 1

F_IN[x]	
Description	Signal channel [x] after INPSRC selection, before TIM FLT unit

F_IN[x]	
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Input data is 0 1 : Input data is 1

TIM_IN[x]	
Description	Signal channel [x] after TIM input signal synchronization
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Input data is 0 1 : Input data is 1

13.8 TIM Signal Description

13.8.1 TIM Reset

Loop	$j = \{n : 0 <= n <= \mathrm{NTIM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	–
Format	–

TIM[j]_RESET_CH[x]	
Description	TIM[j] channel [x] reset active signal
Loop	–
Condition	–
Signal Type	ARRAY[NTIM][8]

TIM[j]_RESET_CH[x]	
Assignment	TIM[j]_RST.RST_CH[x] == 1

13.8.2 TIM Interrupt Signals

Loop	–
Condition	–
Format	–

TIM_GPROFL_IRQ	
Description	TIM[i]_CH[x]_GPR0 and TIM[i]_CH[x]_GPR1 data overflow, old data was not read out before new data has arrived at input pin of channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TIM_TODET_IRQ	
Description	Time out reached for input signal of channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TIM_NEWVAL_IRQ	
Description	New measurement value detected by SMU of channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TIM_ECNTOFL_IRQ	
Description	ECNT counter overflow of channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TIM_CNTOFL_IRQ	
Description	SMU TIM[i]_CH[x]_CNT counter overflow of channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TIM_GLITCHDET_IRQ	
Description	A glitch was detected by the TIM filter of channel x
Loop	–
Condition	–

TIM_GLITCHDET_IRQ	
Signal Type	INT
Assignment	–

13.8.3 TIM Signals

Loop	–
Condition	–
Format	–

LUT_IN0	
Description	INPUTSRC unit: Input signal 0 for lookuptable processing
Loop	–
Condition	–
Signal Type	INT
Assignment	–

LUT_IN1	
Description	INPUTSRC unit: Input signal 1 for lookuptable processing
Loop	–
Condition	–
Signal Type	INT
Assignment	–

LUT_IN2	
Description	INPUTSRC unit: Input signal 2 for lookuptable processing
Loop	–
Condition	–
Signal Type	INT
Assignment	–

F_IN	
Description	Output signal of the INPUTSRC units of each channels
Loop	–
Condition	–
Signal Type	INT
Assignment	–

FLT_CLK_RES	
Description	FLT unit: Clock resolution in use
Loop	–
Condition	–
Signal Type	INT
Assignment	–

F_IN_SYNC	
Description	FLT unit: FLT input signal F_IN synchronized based on the chosen clock resolution
Loop	–

F_IN_SYNC	
Condition	–
Signal Type	INT
Assignment	–

FEDGE_DET	
Description	FLT unit: Falling edge detection signal
Loop	–
Condition	–
Signal Type	INT
Assignment	–

REDGE_DET	
Description	FLT unit: Rising edge detection signal
Loop	–
Condition	–
Signal Type	INT
Assignment	–

INC	
Description	Input signal to TDU_SLICE; on 1 the counter will increase
Loop	–
Condition	–
Signal Type	INT
Assignment	–

LOAD	
Description	Input signal to TDU_SLICE; on 1 the counter will be loaded with LOAD_VAL
Loop	–
Condition	–
Signal Type	INT
Assignment	–

LOAD_VAL	
Description	Input signal to TDU_SLICE; defines the value which will be loaded into the counter in case LOAD=1
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EQ_EVT	
Description	Output signal of TDU_SLICE; 1 in case counter and compare values are identical
Loop	–
Condition	–
Signal Type	INT
Assignment	–

GT_EVT	
Description	Output signal of TDU_SLICE; 1 in case counter value is greater than compare value
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TDU_TIMEOUT_EVT	
Description	Output signal of TDU; signal behavior depends on TDU configuration
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TDU_WORD_EVT	
Description	Output signal of TDU; signal behavior depends on TDU configuration
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TDU_FRAME_EVT	
Description	Output signal of TDU; signal behavior depends on TDU configuration
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TDU_SAMPLE_EVT	
Description	Output signal of TDU; signal behavior depends on TDU configuration
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TDU_TOCTRL_0_EVENT	
Description	TDU unit: single cycle pulse indicating that the timeout functionality was disabled
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TDU_START_000_EVENT	
Description	TDU unit: single cycle pulse indicating that the timeout functionality was enabled
Loop	–
Condition	–
Signal Type	INT
Assignment	–

USED_CLK_RES	
Description	TIM_CH unit: Internal clock resolution signal selected depending on the configuration in TSSM mode;
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TSSM_IN	
Description	TIM_CH unit: Internal input signal selected depending on the configuration in TSSM mode;
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TSSM_OUT	
Description	TIM_CH unit: Output signal in use by TSSM mode
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SHIFTOUT_IN	
Description	TIM_CH unit: Internal input signal selected depending on configuration in TSSM mode;
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ARU_OUT	
Description	TIM_CH unit: Output signal defining the values for the ARU transfer of this channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TO_DET	
Description	TIM_CH unit: Internal signal stores any TDU_TIMEOUT_EVT events until the actual ARU transfer is served
Loop	–
Condition	–
Signal Type	INT
Assignment	–

FOUT_NEXT	
Description	TIM_CH unit: Output signal the next channel instance
Loop	–
Condition	–
Signal Type	INT
Assignment	–

FOUT_PREV	
Description	TIM_CH unit: Input signal from the previous channel instance
Loop	–
Condition	–
Signal Type	INT
Assignment	–

NEWVAL	
Description	TIM_CH unit: Signal which indicates that a capture event of the defined channel mode has occured
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CNTOFL	
Description	TIM_CH unit: Signal which indicates that an overflow of the channel counter has occured
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ECNTOFL	
Description	TIM_CH unit: Signal which indicates that an overflow of the edge counter has occured
Loop	–
Condition	–
Signal Type	INT
Assignment	–

GPROFL	
Description	TIM_CH unit: Signal which indicates that captured data was overwritten by new capture event
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EXT_CAP_SRC_IN	
Description	TIM_CH unit: Input port vector 9 bits , providing all local capture source events from following channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EXT_CAP_SRC_OUT	
Description	TIM_CH unit: Output port vector 9 bits, providing all local capture source events from this channel for the previous channel
Loop	–
Condition	–
Signal Type	INT

EXT_CAP_SRC_OUT	
Assignment	–

NEWVAL_EVT	
Description	Interrupt event – triggered when a new measurement value is detected by SMU of channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CNTOFL_EVT	
Description	Interrupt event – triggered when SMU TIM[i]_CH[x]_CNT counter of channel x overflows
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ECNTOFL_EVT	
Description	Interrupt event – triggered when ECNT counter of channel x overflows
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TODET_EVT	
Description	Interrupt event – triggered when input signal of channel x times out
Loop	–
Condition	–
Signal Type	INT
Assignment	–

GLITCHDET_EVT	
Description	Interrupt event – triggered when a glitch is detected by the TIM filter in channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

GPROFL_EVT	
Description	Interrupt event – triggered when TIM[i]_CH[x]_GPR0 and TIM[i]_CH[x]_GPR1 data overflow, old data was not read out before new data has arrived at input pin of channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

13.9 TIM Port Description

13.9.1 TIM Map Data Interface

Loop	–
Condition	–
Format	–

TIM_MAP_DATA[x]	
Description	[48:48]: Signal level bit from TIM instance channel x.[47:24]: Actual filter value TIM[i]_CH[x]_FLT_RE (with i=0)/ TIM[i]_CH[x]_FLT_FE (with i=0) if corresponding channel x bit field TIM[i]_CH[x]_CTRL.FLT_MODE_RE/ TIM[i]_CH[x]_CTRL.FLT_MODE_FE is 1 else 0 is assigned.[23:0]: Time stamp value is selected by TIM[i]_CH[x]_CTRL.TBU0_SEL and TIM[i]_CH[x]_CTRL.GPR0_SEL of channel x, if bit field TIM[i]_CH[x]_CTRL.TIM_EN= 1.
Loop	$x = \{n : 0 <= n <= 5\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	48 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_MAP_DVAL	
Description	Mark TIM_MAP_DATA[x] valid for one clock cycle
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	5 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_MAP_IN6	
Description	Signal level of filtered output of TIM[i] instance channel 6.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

13.9.2 TIM Ports

Loop	–
Condition	–
Format	–

AUX_IN	
Description	Auxiliary input for TIM channels (wired to TIM[i]_AUX_IN)
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_EXT_CAPTURE	
Description	External capture signal of each channel
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIMCH0_EXT_CAP_SRC_-OUT	
Description	External capture sources signal of channel 0
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	8 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_TBU_TS0	
Description	Time base channel 0 signal
Loop	–
Condition	–

TIM_TBU_TS0	
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	26 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_TBU_TS1	
Description	Time base channel 1 signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_TBU_TS2	
Description	Time base channel 2 signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_IN	
Description	External input for TIM channels
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	isAsyncInput
Operational Reset	GTM_RESET
Reset Value	–

F_OUT	
Description	FLT unit: Output signal of each filter channel
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

14 Timer Output Module (TOM)

14.1 Overview

The Timer Output Module (TOM) offers up to 16 independent channels (index x) to generate simple PWM signals at each output port *TOM_OUT* [x:x].

Additionally, at TOM output *TOM_OUT* [15:15] a pulse count modulated signal can be generated.

The architecture of the TOM sub-module is depicted in figure *57 "TOM Block Diagram"* .

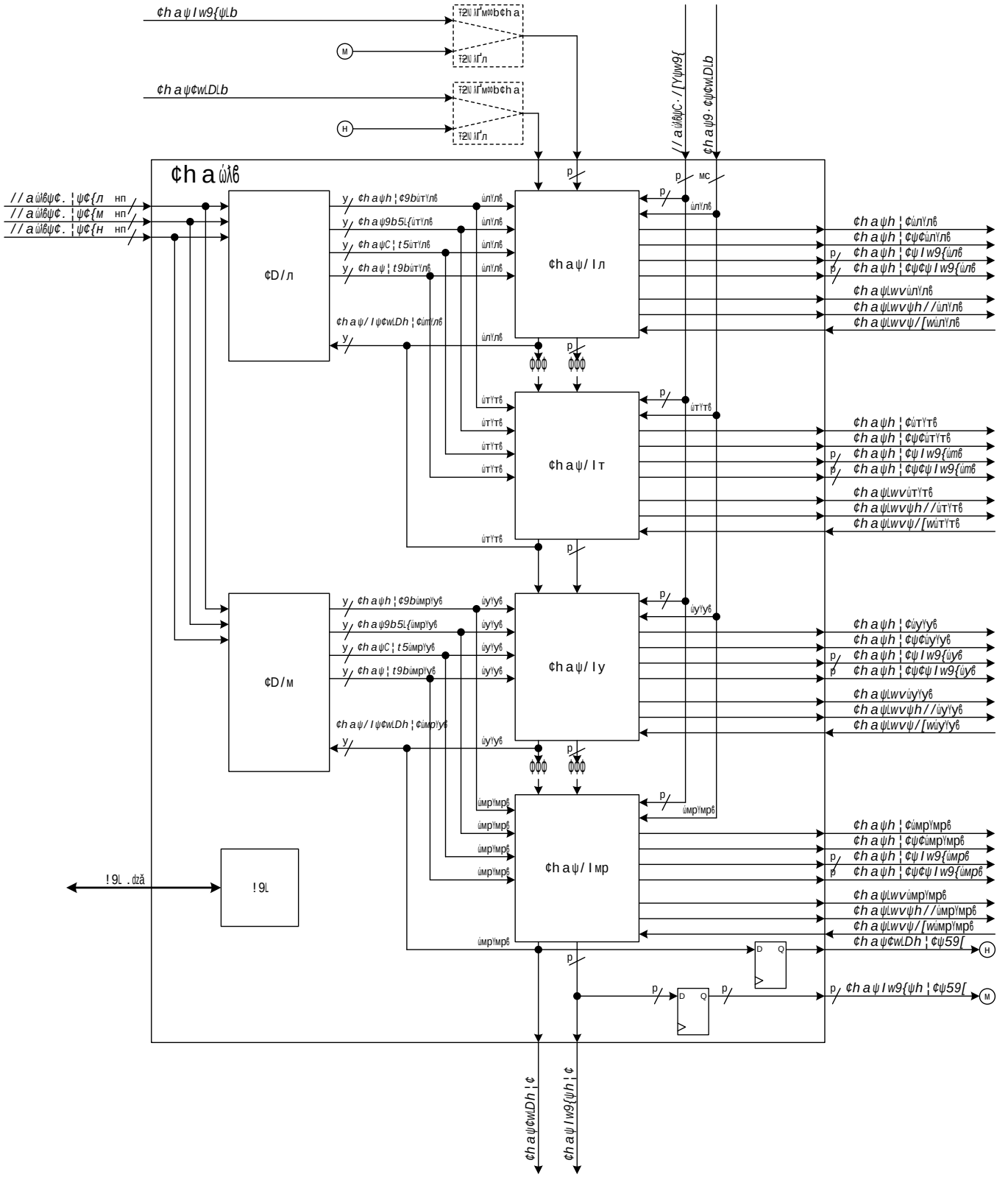
Indices and their range as used inside this chapter are:

- NTOM= number of TOM instances in the device
- i:={0, 1,..., NTOM-1} instance index of cluster / TOM module
- x:={0, 1,..., 15} index of channels
- g:={0, 1} index of global control unit
- c:={0, 1,..., 7} index of channel in the context of a global control unit
- y:={0, 1, 2} index of used time base

The register and bitfield names used inside this chapter have the following meaning:

Note:
All figures in this chapter, which show block diagrams or gate schematics are drawn in principle to show specific functionality. They cannot completely represent the real implementation of the design.

Figure 57 TOM Block Diagram



As shown in figure *57 "TOM Block Diagram"* , each TOM module integrates 16 channels with outputs *TOM_OUT* and *TOM_OUT_T* . The two sub−modules TGC0 and TGC1 are global channel control units that control the enabling/disabling of the channels and their outputs as well as the update of their period and pulse width register. Unlike ATOM, TOM has no communication interface with ARU and it can only be configured via AEI interface.

The input clocks for the TOM channels come from the cluster clock *CLS[i]_CLK* with the configurable clock resolution *CCM[i]_FXCLK_RES* signals of the CCM sub−module. This gives the freedom to select a programmable input clock for the TOM channel counters.

The module TOM receives two (three) timestamp values *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* (and *CCM[i]_TBU_TS2*) in order to realize synchronized output behavior on behalf of a common time base.

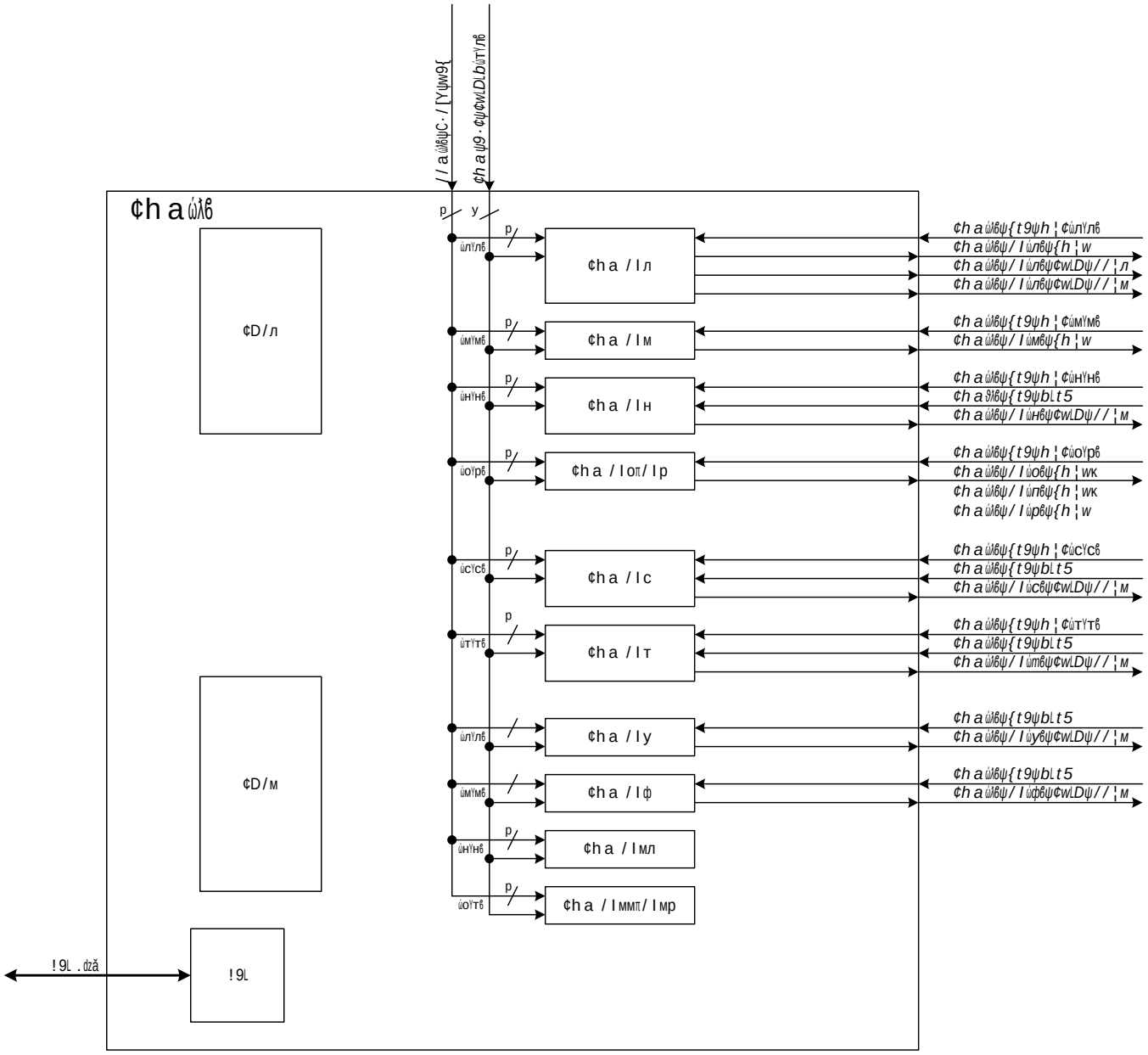
As shown in figure *57 "TOM Block Diagram"* , NTOM−1 TOM sub−modules are connected together through a trigger chain. The trigger signal *TOM_TRIGIN* of TOM instance i is coming from *TOM_TRIGOUT* of TOM instance i−1 and the trigger signal *TOM_TRIGOUT* of TOM instance i is routed to *TOM_TRIGIN* of TOM instance i+1. The trigger chain goes across all channels one by one. The trigger input signal *TOM_TRIGIN* of TOM instance i is connected to the trigger input signal *TOM_CH_TRIGIN* [0:0] of channel 0. The channel 0 can generate a trigger output signal *TOM_CH_TRIGOUT* [0:0] that is connected to *TOM_CH_TRIGIN* [1:1] to trigger channel 1. In this way, each channel x is triggered from the preceding channel and then can trigger the next channel (please refer to figure *60 "TOM Channel Architecture"* and chapter *14.3.2 "Internal Trigger Interface"* for the internal trigger chain details). Channel 15 of TOM instance i is routed to trigger channel 0 of the next instance i+1.
Similarly, there is also a HRES value chain that goes across all channels of each instance along with the trigger chain. The high−resolution value input *TOM_HRES_IN* of TOM instance i is connected to *TOM_HRES_OUT* of TOM instance i−1 and the high−resolution value output *TOM_HRES_OUT* of TOM instance i is routed to *TOM_HRES_IN* of TOM instance i+1. Please refer to chapter *14.12 "High Resolution PWM Support (HRES Mode)"* for more details.

The trigger input signal of TOM instance 0 is connected to its own output *TOM_TRIGOUT_DEL* , i.e. the last channel of TOM instance 0 can trigger the first channel of TOM instance 0 (this path is registered, which means delayed by one cluster clock period).
Similarly, the high−resolution value input *TOM_HRES_IN* of TOM instance 0 is connected to its own output *TOM_HRES_OUT_DEL* , i.e. the high−resolution value output of the last channel of TOM instance 0 is also fed to the first channel of TOM instance 0, also with one pipeline register delay.

The function of the interrupt signals *TOM_IRQ* , *TOM_IRQ_OCC* and *TOM_IRQ_CLR* is described in more detail in chapter *3.12 "GTM−IP Interrupt Concept"* .

In figure *58 "TOM−SPE Interface Signals"* the TOM SPE interface signals are depicted. Please refer to chapter *23 "Sensor Pattern Evaluation (SPE)"* for more details about the TOM SPE interface signals (*TOM[i]_SPE_OUT* , *TOM[i]_SPE_NIPD* , *TOM[i]_CH[0]_SOUR* , *TOM[i]_CH[1]_SOUR* , *TOM[i]_CH[0]_TRIG_CCU0* , *TOM[i]_CH[0]_TRIG_CCU1* , *TOM[i]_CH[2]_TRIG_CCU1* , *TOM[i]_CH[6]_TRIG_CCU1* , *TOM[i]_CH[7]_TRIG_CCU1* , *TOM[i]_CH[8]_TRIG_CCU1* , *TOM[i]_CH[9]_TRIG_CCU1*).

Figure 58 TOM–SPE Interface Signals



14.2 TOM Global Channel Control (TGC[0], TGC[1])

14.2.1 Overview

Two global channel control units (TGC[0] and TGC[1]) exist to drive a number of individual TOM channels synchronously by external or internal events.

Each TGC[g] can drive up to eight TOM channels where TGC[0] controls TOM channels x (x∈{0, 1,..., 7}) and TGC[1] controls TOM channels x (x∈{8, 9,..., 15}).

The TOM sub–module supports four different kinds of signaling mechanisms:

- Global enable/disable mechanism for each TOM channel with control register **TOM[i]_TGC[g]_ENDIS_CTRL** and status register **TOM[i]_-TGC[g]_ENDIS_STAT** .
- Global output enable mechanism for each TOM channel x with control register **TOM[i]_TGC[g]_OUTEN_CTRL** and status register **T-OM[i]_TGC[g]_OUTEN_STAT** .
- Global force update mechanism for each TOM channel and optionally reset the counter **TOM[i]_CH[x]_CN0** with control register **T-OM[i]_TGC[g]_FUPD_CTRL** .
- Global update enable of the register **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** and **TOM[i]_CH[x]_CTRL.CLK_SRC** for each TOM channel x with the control bit field **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** (x= c+8*g).

14.2.2 TGC Sub–unit

Each of the first three individual mechanisms (enable/disable of the channel, output enable and force update) can be driven by three different trigger sources.
The three trigger sources are :

1. Over the configuration interface (bit **TOM[i]_TGC[g]_GLB_CTRL.HOST_TRIG**
2. The TBU time stamp (signal *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , *CCM[i]_TBU_TS2* if available)
3. The trigger signal *TOM_CH_TRIGOUT* [7:0] for TGC[0] or *TOM_CH_TRIGOUT* [15:8] for TCG[1] (bunch of trigger signals *TOM_CH_TRIGOUT* [x:x]) which can be the trigger *TRIG_CCU0* of channel x, the trigger of preceding channel x–1 (i.e. signal *TOM_CH_TRIGOUT* [x–1:x–1]) or the external trigger *TOM_EXT_TRIGIN* [x:x] from TIM.

The first way (1.) is to trigger the control mechanism by a direct register write access via configuration interface (bit **TOM[i]_TGC[g]_GLB_CTRL.HOST_TRIG**).

The second way (2.) is provided by a compare match trigger on behalf of a specified time base coming from the module TBU (selected by bitfield **TOM[i]_TGC[g]_ACT_TB.TBU_SEL**) and the time stamp compare value defined in the bit field **TOM[i]_TGC[g]_ACT_TB.ACT_TB** .

In this case, the selected TBU time base *CCM[i]_TBU_TS0* / *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* should be configured to be up–counting. A cyclic event compare of **TOM[i]_TGC[g]_ACT_TB.ACT_TB** and selected *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , *CCM[i]_TBU_TS2* is performed. Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more details about cyclic event compare strategy.

The third way (3.) is the input *TOM_CH_TRIGOUT* (bunch of trigger signals *TOM_CH_TRIGOUT* [x:x]) coming from the TOM channel x, x∈{0, 1,..., 7} for TGC[0] or x∈{8, 9,..., 15} for TGC[1].

The corresponding trigger signal *TOM_CH_TRIGOUT* [x:x] coming from channel x can be masked by the register **TOM[i]_TGC[g]_INT_TRIG** .

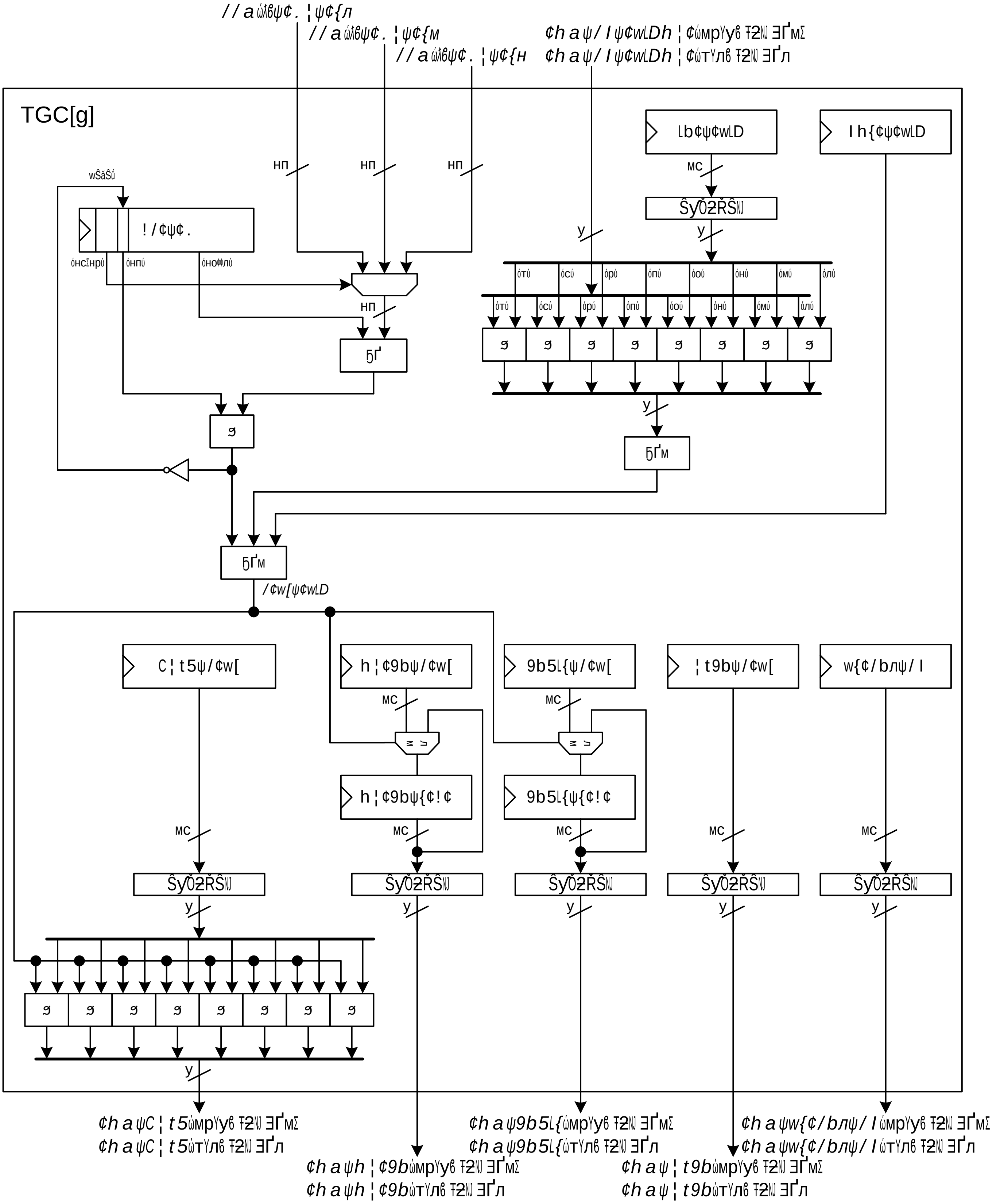
To enable or disable each individual TOM channel, the register **TOM[i]_TGC[g]_ENDIS_CTRL** and/or **TOM[i]_TGC[g]_ENDIS_STAT** have to be used.

The register **TOM[i]_TGC[g]_ENDIS_STAT** controls directly the signals *TOM_ENDIS* [7:0] if g=0 or *TOM_ENDIS* [15:8] if g=1 (see figure *59 "TOM Global Channel Control Mechanism"*). A write access to this register is possible.

The register **TOM[i]_TGC[g]_ENDIS_CTRL** is a shadow register that overwrites the value of register **TOM[i]_TGC[g]_ENDIS_STAT** if one of the three trigger conditions matches.

To ensure a clean restart of a TOM channel x, it is recommended to raise a channel reset by setting of bit **TOM[i]_TGC[g]_GLB_CTRL.RST_CH[c]** (x = c + 8*g) after a channel is disabled (*TOM_ENDIS* [x:x]=0) and before it is enabled again (*TOM_ENDIS* [x:x]=1).

Figure 59 TOM Global Channel Control Mechanism



The output of the individual TOM channels can be controlled using the register **TOM[i]_TGC[g]_OUTEN_CTRL** and **TOM[i]_TGC[g]_OUTEN_STAT** .

The register **TOM[i]_TGC[g]_OUTEN_STAT** controls directly the signals *TOM_OUTEN* [7:0] if g=0 or *TOM_OUTEN* [15:8] if g=1. A write access to this register is possible.

TOM[i]_TGC[g]_OUTEN_CTRL is a shadow register that overwrites the value of register **TOM[i]_TGC[g]_OUTEN_STAT** if one of the three trigger conditions matches.

The trigger condition has always priority over the bus write access to the **TOM[i]_TGC[g]_OUTEN_STAT** and **TOM[i]_TGC[g]_ENDIS_STAT** registers, even if **TOM[i]_TGC[g]_OUTEN_CTRL.OUTEN_CTRL[k]** / **TOM[i]_TGC[g]_OUTEN_CTRL.ENDIS_CTRL[k]** is set to 0b00. This means that the bus write access to **TOM[i]_TGC[g]_OUTEN_STAT** and **TOM[i]_TGC[g]_ENDIS_STAT** register is ignored in the clock cycle when the trigger condition is active.

The register **TOM[i]_TGC[g]_FUPD_CTRL** defines which of the TOM channels receive a force update event if the trigger signal *CTRL_TRIG* is raised while **TOM[i]_TGC[g]_FUPD_CTRL.RSTCN0_CH[c]** determines whether the force update event also reset counter **TOM[i]_CH[x]_CN0.CN0** of the channel. If in continuous counting up mode with **TOM[i]_CH[x]_CTRL.RST_CCU0** =0, the force update mechanism is configured with **TOM[i]_TGC[g]_FUPD_CTRL.RSTCN0_CH[c]** =1, the force update event will simultaneously update the operation registers, reset the counter and also set the output to be **TOM[i]_CH[x]_CTRL.SL** . That means the force update mechanism can be used to stop the current PWM signal generation and restart a new PWM signal generation with new PWM parameters.

The force update request is stored and executed synchronized to the selected *CCM[i]_FXCLK_RES* as shown in figure *60 "TOM Channel Architecture"* . And the falling edge of the synchronized pulse will trigger the force update functionalities as mentioned above.

The register bit **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** defines for which TOM channel the update of the working register **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** by the corresponding shadow register **TOM[i]_CH[x]_SR0.SR0** , **TOM[i]_CH[x]_SR1.SR1** , **TOM[i]_CH[x]_CTRL_SR.SL_SR** and **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** is enabled. If update is enabled, the registers **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** and **TOM[i]_CH[x]_CTRL.CLK_SRC** will be updated on reset of counter **TOM[i]_CH[x]_CN0.CN0** (see figure *60 "TOM Channel Architecture"*). An exception is the configuration of **TOM[i]_CH[x]_CTRL.SR0_TRIG** =1 which enables the trigger generation defined by **TOM[i]_CH[x]_SR0.SR0** then **TOM[i]_CH[x]_CM0** is not updated with **TOM[i]_CH[x]_SR0** .

14.3 TOM Channel Architecture

Each individual TOM channel comprises a Counter Compare Unit 0 (CCU0) which contains the counter **TOM[i]_CH[x]_CN0.CN0** and the period register **TOM[i]_CH[x]_CM0.CM0** , a Counter Compare Unit 1 (CCU1) which contains the pulse width register **TOM[i]_CH[x]_CM1.CM1** and the Signal Output Generation Unit (SOU) which contains the output register SOUR (see figure *60 "TOM Channel Architecture"*). The architecture is depicted in figure *60 "TOM Channel Architecture"* .

Note:

Figure *60 "TOM Channel Architecture"* doesn't reflect the real implementation of the circuit but is intended to explain the functionality of the channel in a better manner.

Figure 60 TOM Channel Architecture

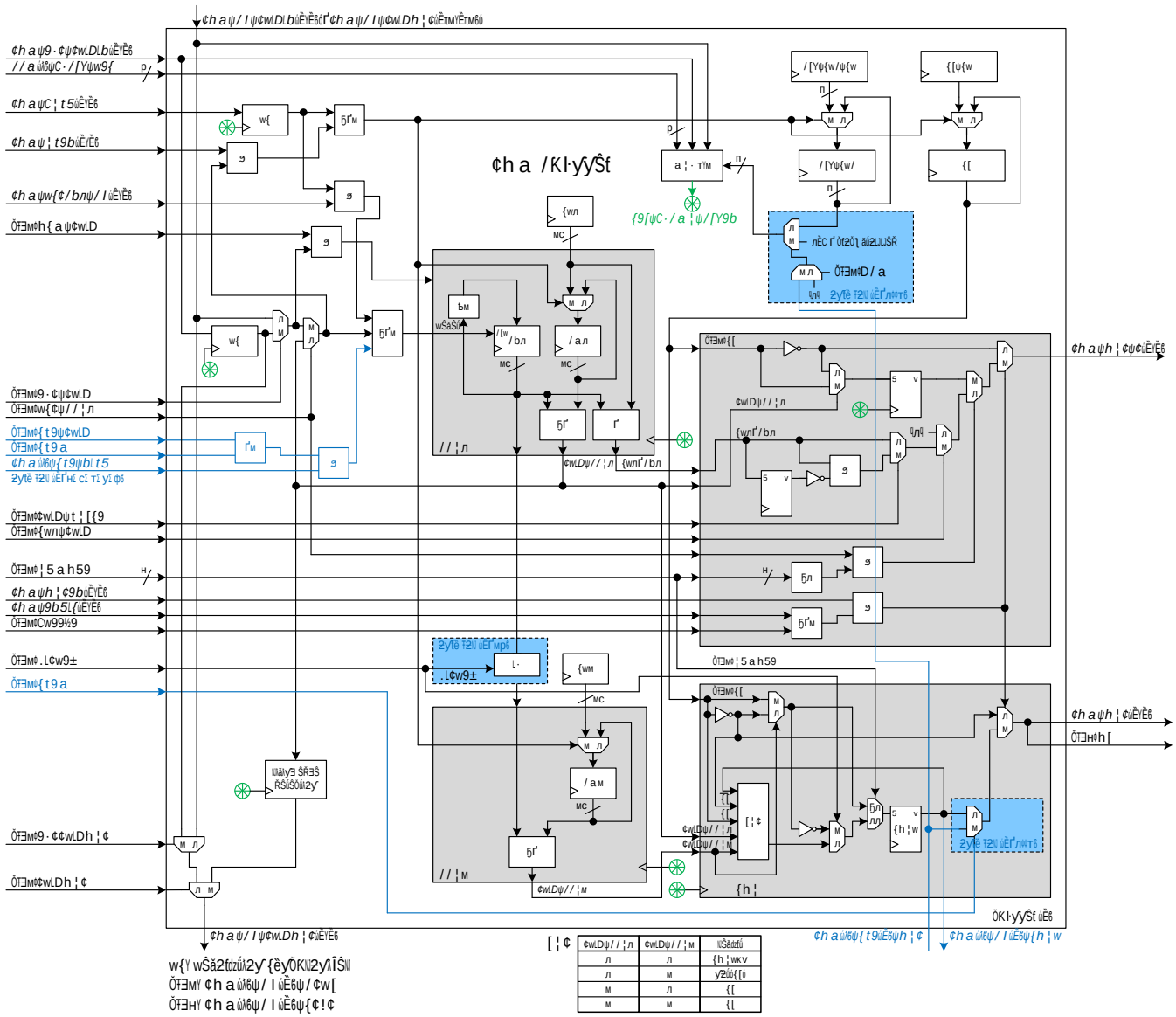
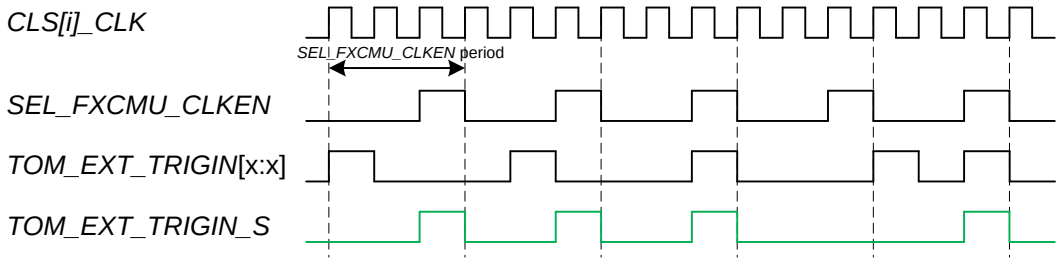


Figure 61 Timing Diagram of RS Unit



Most functionalities (e.g. the counter **TOM[i]_CH[x]_CN0.CN0** in CCU0 unit) are synchronized with the selected clock enable signal *SEL_FXCMU_CLKEN* that is configured by **TOM[i]_CH[x]_CTRL.CLK_SRC** . This configuration register can select the clock source from fixed clock enable signals *CCM[i]_FXCLK_RES* that are provided from CCM sub-module. The signal *SEL_FXCMU_CLKEN* looks like clock enable signal that is shown in figure *61 "Timing Diagram of RS Unit"* . Most functionality modes are only active when *SEL_FXCMU_CLKEN* = 1 and the consequent operation registers and outputs are always synchronously updated at the falling edge of *SEL_FXCMU_CLKEN* pulse. The functionalities synchronous to the selected clock resolution *SEL_FXCLK_CLKEN* are depicted with the green round symbol in figure *60 "TOM Channel Architecture"* .

In all TOM channels the operation registers **TOM[i]_CH[x]_CN0.CN0** , **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM1.CM1** and the shadow registers **TOM[i]_CH[x]_SR0.SR0** and **TOM[i]_CH[x]_SR1.SR1** are 16-bit wide. As shown in figure *60 "TOM Channel Architecture"* , the unit CCU0 provides different counting up/down functionalities in various modes. The comparators inside CCU0 and CCU1 perform selectable signed greater-equal or less-equal comparison against different parameters in different modes and then generate TRIG_CCU0=1 when the compare match event in CCU0 occurs or generate TRIG_CCU1=1 when the compare match event in CCU1 occurs.

The Signal Output Unit (SOU) generates the output signal for each TOM channel. The output signal level depends on the bit **TOM[i]_CH[x]_CTRL.SL** in combination with the counter behavior and its compare events in CCU0 and CCU1. The initial signal output level for the channel is the reverse value of the bit **TOM[i]_CH[x]_CTRL.SL** .

In continuous counting up mode, **TOM[i]_CH[x]_CM0.CM0** represents PWM period while **TOM[i]_CH[x]_CM1.CM1** is PWM pulse width. In order to generate various PWM signals, CCU0 counts up **TOM[i]_CH[x]_CN0.CN0** and compares it with **TOM[i]_CH[x]_CM0.CM0** while CCU1 compares it with **TOM[i]_CH[x]_CM1.CM1** . Depending on the configuration bit **TOM[i]_CH[x]_CTRL.RST_CCU0** , the counter **TOM[i]_CH[x]_CN0.CN0** can be reset, when the counter value is equal to the compare value **TOM[i]_CH[x]_CM0.CM0** (i.e. **TOM[i]_CH[x]_CN0.CN0** counts only from 0 to **ATOM[i]_CH[x]_CM0.CM0** −1 and is then reset to 0) or when it is triggered by the TOM[i] trigger signal *TOM_CH_TRIGIN* [x:x] (*TOM_CH_TRIGOUT* [x−1:x−1]) of the preceding channel x−1 which can also be the last channel of preceding instance TOM[i−1]) or when it is triggered by the external trigger signal *TOM_EXT_TRIGIN* [x:x] from TIM module i.

As an exception, the input *TOM_TRIGIN* of instance TOM0 is triggered by its own last channel 15 via signal *TOM_CH_TRIGOUT* [15:15].

The external trigger signal *TOM_EXT_TRIGIN* [x:x] is synchronous to the clock source selected by RS (Resolution Synchronizer) unit and then taken into use.
With such a RS unit, in each *SEL_FXCLK_CLKEN* period (i.e. selected clock resolution signal), any event (or pulses) of *TOM_EXT_TRIGIN* will be synchronized by RS unit on *TOM_EXT_TRIGIN_S* that looks exactly the same as *SEL_FXCLK_CLKEN* . Figure *61 "Timing Diagram of RS Unit"* shows the timing diagram of RS unit. If *TOM_EXT_TRIGIN* has one or more pulses in a *SEL_FXCLK_CLKEN* period, the synchronized output *TOM_EXT_TRIGIN_S* will have 1 pulse at the end of this period. Otherwise, *TOM_EXT_TRIGIN_S* is always 0.

The trigger signal *TOM_CH_TRIGIN* [x:x] coming from the preceding channel may not be synchronous to the selected clock resolution of channel x. If the counter **TOM[i]_CH[x]_CN0.CN0** is reset by the trigger signal *TOM_CH_TRIGIN* [x:x] of a preceding channel with **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, the preceding triggering channel should be configured with the same *CCM[i]_FXCLK_RES* as the triggered channel.

In continuous counting up mode, once the comparison event of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** −1 in unit CCU0 occurs, *TRIG_CCU0* = 1 will be set to trigger the output generation of *TOM_OUT* [x:x] in the SOU sub−unit together with the corresponding interrupt signals. In order to trigger the next channel, the rising edge detection unit can detect each rising edge of *TRIG_CCU0* (i.e. each comparison event) and generate a single cluster clock pulse on *TOM_CH_TRIGOUT* [x:x] when the selected clock resolution signal *SEL_FXCLK_CLKEN* =1 (i.e. synchronous to the selected clock resolution). In addition, *TOM_CH_TRIGOUT* [x:x] can also be configured to be *TOM_CH_TRIGIN* [x:x] (*TOM_CH_TRIGOUT* [x−1:x−1] of the preceding channel x−1) or the external trigger signal *TOM_EXT_TRIGIN* [x:x] from TIM.

Similarly, once the comparison event of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1** −1 in unit CCU1 is occurs, *TRIG_CCU1* = 1 will be set to trigger the output generation of *TOM_OUT* [x:x] in the SOU sub−unit together with the corresponding interrupt signals.

When **TOM[i]_CH[x]_CN0.CN0** < **TOM[i]_CH[x]_CM0.CM0** −1, *TRIG_CCU0* will be reset as 0. Similarly, when **TOM[i]_CH[x]_CN0.CN0** < **TOM[i]_CH[x]_CM1.CM1** −1, *TRIG_CCU1* will be reset to 0.

As shown in figure *60 "TOM Channel Architecture"* , in continuous counting up mode according to the LUT with *TRIG_CCU0* , *TRIG_CCU1* and the signal level configuration **TOM[i]_CH[x]_CTRL.SL** , SOU unit can generate *TOM_OUT* [x:x] with various PWM timing characteristics when the channel and the output are both enabled (*TOM_OUTEN* [x:x]=1 and *TOM_ENDIS* [x:x]=1). Please refer to the chapter of different modes for more details on PWM signal generation.

The output level on the TOM channel output pin *TOM_OUT* [x:x] is captured in bit **TOM[i]_CH[x]_STAT.OL** (see figure *60 "TOM Channel Architecture"*)

If a TOM channel is disabled by the register **TOM[i]_TGC[g]_OUTEN_STAT** , the actual value of the channel output at *TOM_OUT* [x:x] is defined by the signal level bit (**TOM[i]_CH[x]_CTRL.SL**).
If the output is enabled, the output at *TOM_OUT* [x:x] depends on value of register SOUR (see figure *60 "TOM Channel Architecture"*).

In addition to the bit **TOM[i]_CH[x]_CTRL.SL** , a shadow register exists in **TOM[i]_CH[x]_CTRL_SR.SL_SR** . With this shadow register it is possible to change the signal level value synchronous to the update of the operation registers **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM1.CM1** from its shadow registers **TOM[i]_CH[x]_SR0.SR0** and **TOM[i]_CH[x]_SR1.SR1** .

The operating clock resolution from CMU is specified in the bitfield **TOM[i]_CH[x]_CTRL.CLK_SRC** . In addition, a shadow register **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** exists. With this shadow register it is possible to change the selected clock resolution from CMU synchronously to the update of the operation registers **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM1.CM1** from its shadow registers **TOM[i]_CH[x]_SR0.SR0** and **TOM[i]_CH[x]_SR1.SR1** .

Due to the fixed clock resolution scheme from CMU, the first clock resolution period may differ from expected value, depending on the point of time, the **TOM[i]_CH[x]_CTRL.CLK_SRC** bitfield is updated from its shadow register bitfield **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** .

When the output is disabled (*TOM_OUTEN* [x:x]=0), the output signals *TOM_OUT* [x:x] and *TOM_OUT_T* [x:x] are the inverse value of the **TOM[i]_CH[x]_CTRL.SL** bit. When the output is enabled (*TOM_OUTEN* [x:x]=1), the output signals *TOM_OUT* [x:x] and *TOM_OUT_T* [x:x] are dependent on different operation modes, the channel enable signal *TOM_ENDIS* [x:x] and the freeze mode enable register **TOM[i]_CH[x]_CTRL.FREEZE** . (Please refer to register description of **TOM[i]_CH[x]_CTRL.SL** for more details.) Especially, in some modes the output signal *TOM_OUT_T* [x:x] is not supported. So it is always clamped to 0 if the output and the channel are both enabled (i.e. *TOM_OUTEN* [x:x]=1 and *TOM_ENDIS* [x:x]=1). When the output is enabled (*TOM_OUTEN* [x:x]=1) but the channel is disabled (*TOM_ENDIS* [x:x]=0), if **TOM[i]_CH[x]_CTRL.FREEZE** =0, *TOM_OUT_T* [x:x] is ! **TOM[i]_CH[x]_CTRL.SL** . When the output is enabled (*TOM_OUTEN* [x:x]=1) but the channel is disabled (*TOM_ENDIS* [x:x]=0), if **TOM[i]_CH[x]_CTRL.FREEZE** =1, *TOM_OUT_T* [x:x] is clamped to 0.

When the channel is disabled (*TOM_ENDIS* [x:x]=0), the counting and comparing functionalities in CCU0 and CCU1 are deactivated in different modes and thus *TRIG_CCU0* = 0, *TRIG_CCU1* = 0. Therefore, *TOM_CH_TRIGOUT* [x:x] is not generated if **TOM[i]_CH[x]_CTRL.TRIGOUT** = 1, regardless of *TOM_OUTEN* [x:x].

14.3.1 External Trigger Interface

Each channel x of TOM instance i has an input signal *TOM_EXT_TRIGIN* [x:x] that is connected to *TIM_EXT_CAPTURE* [x:x] of TIM instance i if x ∈ {0, 1, ..., 7} or *TIM_EXT_CAPTURE* [x−7:x−7] of TIM instance i if x ∈ {8, 9, ..., 15}. Via this signal, events (high pulses with the length of 1 *CLS[i]_CLK* period) generated by the TIM module i can be used to control the functionality of the TOM channel in multiple ways:

- *TOM_EXT_TRIGIN* [x:x] events can be used to define the clock source of the selected channel mode (**TOM[i]_CH[x]_CTRL.CLK_SRC** = 0b1110).
- *TOM_EXT_TRIGIN* [x:x] events can be used to reset the counter register **TOM[i]_CH[x]_CN0** in continuous counting up mode or change the counting direction in counting up−down mode or trigger the start of counting in one−shot mode (activated by **TOM[i]_CH[x]_CTRL.EXT_TRIG** = 1 && **TOM[i]_CH[x]_CTRL.RST_CCU0** = 1). In this case, if **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** = 0b10 is configured, *TOM_EXT_TRIGIN* [x:x] events can also trigger the update of the operation registers from their shadow registers. Any *TOM_EXT_TRIGIN* [x:x] event is synchronized in the TOM channel as specified in figure *61 "Timing Diagram of RS Unit"* .
- *TOM_EXT_TRIGIN* [x:x] can be used as a trigger event for the following channels (activated by **TOM[i]_CH[x]_CTRL.EXTTRIGOUT** = 1 && **TOM[i]_CH[x]_CTRL.TRIGOUT** = 0). Any *TOM_EXT_TRIGIN* [x:x] event is synchronized in the TOM channel as specified in figure *61 "Timing Diagram of RS Unit"* .

In certain configurations of the TIM module, the *TOM_EXT_TRIGIN* [x:x] signal pulses can have a length of more than 1 cluster clock cycle. In this case, the functionality of the ATOM is not specified and it is highly recommended to prevent this configuration in applications.

Before triggering the functionalities mentioned above, the *TOM_EXT_TRIGIN* [x:x] signal is always synchronized with the current selected clock enable signal in RS unit as shown in figure *61 "Timing Diagram of RS Unit"* .

14.3.1.1 Internal Trigger Interface

Each TOM channel x has an input signal *TOM_CH_TRIGIN* [x:x] that is coming from *TOM_CH_TRIGOUT* [x−1:x−1] from the preceding channel. Via this signal, events (high pulses with the length of 1 *CLS[i]_CLK* period) generated by the preceding TOM channel, can be used to control the functionality of TOM channel in multiple ways:

- *TOM_CH_TRIGIN* [x:x] events can be used to define the clock source of the selected channel mode (**TOM[i]_CH[x]_CTRL.CLK_SRC** = 0b1101).
- *TOM_CH_TRIGIN* [x:x] events can be used to reset the counter register **TOM[i]_CH[x]_CN0** in continuous counting up mode or change the counting direction in counting up−down mode or trigger the start of counting in one−shot mode (activated by **TOM[i]_CH[x]_CTRL.EXT_TRIG** = 0 && **ATOM[i]_CH[x]_CTRL.RST_CCU0** = 1). In this case, if **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** = 0b10 is configured, *TOM_CH_TRIGIN* [x:x] events can also trigger the update of the operation registers from their shadow registers.
- *TOM_CH_TRIGIN* [x:x] can be used as a trigger event for following channels (activated by **TOM[i]_CH[x]_CTRL.EXTTRIGOUT** = 0 && **TOM[i]_CH[x]_CTRL.TRIGOUT** = 0).

Especially, for channel 0 of TOM module i, i ∈ {1, 2, ..., NTOM−1}, *TOM_CH_TRIGIN* [0:0] comes from *TOM_CH_TRIGOUT* [15:15] of the preceding TOM instance [i−1] and may be delayed by a register with the cluster clock *CLS[i]_CLK* in channel 0 before triggering the functionalities as mentioned above. This delay is controlled by the device configuration variable TOM_TRIG_CHAIN (indicated by **CCM[i]_HW_CONF.TOM_TRIG_CHAIN**). The delay exists after every N cluster (N = TOM_TRIG_CHAIN). For TOM0, *TOM_CH_TRIGIN* [0:0] is connected to *TOM_TRIGOUT_DEL* via the signal *TOM_CH_TRIGOUT* [15:15] of TOM0 itself as shown in figure *57 "TOM Block Diagram"* . Furthermore, the cluster i can operate on faster GTM−IP clock *CLK* , if the device is defined with FAST_CLS_CLK[i] = 1. In this case, it is allowed to have one more register that is clocked with *CLK* to delay *TOM_CH_TRIGIN* [8:8] in channel 8 before triggering the functionalities. This is controlled by another ATOM device configuration parameter TOM_TRIG_INTCHAIN (it can only be valid if FAST_CLS_CLK[i] = 1 and indicated by **CCM[i]_HW_CONF.TOM_TRIG_INTCHAIN**). The registers for these special cases are not shown in figure *57 "TOM Block Diagram"* .

Unlike *TOM_EXT_TRIGIN* [x:x], *TOM_CH_TRIGIN* [x:x] is not synchronized with the selected clock enable signal of channel x. For the application of 2 channels generating PWM signals synchronously, **TOM[i]_CH[x]_CTRL.CLK_SRC** of the triggering channel and the triggered channel should be configured with the same clock resolution and the same cluster clock.

14.3.2 Internal Trigger Interface

As shown in figure *60 "TOM Channel Architecture"* , each TOM channel x has an input signal *TOM_CH_TRIGIN* [x:x] that is coming from *TOM_CH_TRIGOUT* [x−1:x−1] from the preceding channel. Via this signal, events (high pulses with the length of 1 *CLS[i]_CLK* period) generated by the preceding TOM channel, can be used to control the functionality of TOM channel in multiple ways:

- *TOM_AGC_TRIGIN* [x:x] events can be used to define the clock source of the selected channel mode (**TOM[i]_CH[x]_CTRL.CLK_SRC** = 0b1101).
- *TOM_AGC_TRIGIN* [x:x] events can be used to reset the counter register **TOM[i]_CH[x]_CN0** in continuous counting up mode or change the counting direction in counting up−down mode or trigger the start of counting in one−shot mode (activated by **TOM[i]_CH[x]_CTRL.EXT_TRIG** = 0 && **TOM[i]_CH[x]_CTRL.RST_CCU0** = 1). In this case, if **TOM[i]_TGC[g]_GLB_CTRL.UPEN[c]** = 0b10 is configured, *TOM_CH_TRIGIN* [x:x] events can also trigger the update of the operation registers from their shadow registers.
- *TOM_CH_TRIGIN* [x:x] can be used as a trigger event for following channels (activated by **TOM[i]_CH[x]_CTRL.EXTTRIGOUT** = 0 && **TOM[i]_CH[x]_CTRL.TRIGOUT** = 0).

Especially, for channel 0 of TOM module i, i ∈ {1, 2, ..., NTOM−1}, *TOM_CH_TRIGIN* [0:0] is connected to *TOM_TRIGIN* of instance [i] that comes from *TOM_CH_TRIGOUT* [15:15] of the preceding TOM instance [i−1] and may be delayed by a register with the cluster clock *CLS[i]_CLK* in channel 0 before triggering the functionalities as mentioned above. This delay is controlled by the device configuration variable TOM_TRIG_CHAIN (indicated by **CCM[i]_HW_CONF.TOM_TRIG_CHAIN**). The delay exists after every N cluster (N = TOM_TRIG_CHAIN). For TOM0, *TOM_CH_TRIGIN* [0:0] is connected to *TOM_TRIGOUT_DEL* via the signal *TOM_CH_TRIGOUT* [15:15] of TOM0 itself as shown in figure *57 "TOM Block Diagram"* . Furthermore, the cluster i can operate on faster GTM−IP clock *CLK* , if the device is defined with FAST_CLS_CLK[i] = 1. In this case, it is allowed to have one more register that is clocked with *CLK* to delay *TOM_CH_TRIGIN* [8:8] in channel 8 before triggering the functionalities. This is controlled by another TOM device configuration parameter TOM_TRIG_INTCHAIN (it can only be valid if FAST_CLS_CLK[i] = 1 and indicated by **CCM[i]_HW_CONF.TOM_TRIG_INTCHAIN**). The registers for these special cases are not shown in figure *60 "TOM Channel Architecture"* .

Unlike *TOM_EXT_TRIGIN* [x:x], *TOM_CH_TRIGIN* [x:x] is not synchronized with the selected clock enable signal of channel x. For the application of 2 channels generating PWM signals synchronously, **TOM[i]_CH[x]_CTRL.CLK_SRC** of the triggering channel and the triggered channel should be configured with the same clock resolution and the same cluster clock.

14.3.3 Pulse Width, Period, Signal Level and Clock Frequency Update Mechanisms

The two action registers **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM1.CM1** can be reloaded with the content of the shadow registers **TOM[i]_CH[x]_SR0.SR0** and **TOM[i]_CH[x]_SR1.SR1** . The bit **TOM[i]_CH[x]_CTRL.CLK_SRC** that determines the clock frequency of the counter **TOM[i]_CH[x]_CN0.CN0** can be reloaded with its shadow register **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** . The signal level defined in bitfield **TOM[i]_CH[x]_CTRL.SL** can be reloaded with its shadow register **TOM[i]_CH[x]_CTRL_SR.SL_SR** .

With the update of the register **TOM[i]_CH[x]_CTRL.CLK_SRC** at the end of a period a new counter **TOM[i]_CH[x]_CN0.CN0** clock frequency can be easily adjusted.

The update of the configuration registers **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** with the content of their shadow registers can be performed under three conditions. Firstly, when **TOM[i]_CH[x]_CN0.CN0** is reset by comparing **TOM[i]_CH[x]_CN0.CN0** greater than or equal to **TOM[i]_CH[x]_CM0.CM0** −1, the update will be performed. Secondly, when the channel receives the trigger signal *TOM_CH_TRIGIN* [x:x] (= *TOM_CH_TRIGOUT* [x−1:x−1]) from another TOM channel [x−1] or the external trigger signal *TOM_EXT_TRIGIN* [x:x] of the TIM instance i after it is synchronized to the selected *CCM[i]_FXCLK_RES* of channel x, the update will also be performed. These two update mechanisms must be enabled by configuring **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** = 0b10. Thirdly, these configuration registers can be force updated via a trigger signal via *TOM_FUPD* or via *TOM_EXT_TRIGIN* .

Based on the update mechanisms above, these registers **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** , **TOM[i]_CH[x]_CTRL.CLK_SRC** can be updated synchronously and asynchronously. Synchronous update means that the pulse width, period duration, signal level and the counter operating clock frequency can be changed at the end of the running PWM period. Asynchronous update means that the pulse width, period duration, signal level and the counter operating clock frequency can also be changed during the actual running PWM period. The following two sections *14.3.3.1 "Synchronous Update Of Pulse Width Only"* and *14.3.3.2 "Asynchronous Update Of Pulse Width Only"* provide examples of synchronous update and asynchronous update of pulse width in continuous up mode with **TOM[i]_CH[x]_CTRL.RST_CCU0** = 0.

An update of pulse width, period and counter **TOM[i]_CH[x]_CN0.CN0** clock frequency that takes effect synchronous to the start of a new period can be easily reached by performing following steps:

1. Disable the update of the action register with the content of the corresponding shadow register by setting the channel specific configuration bit **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** = 0b01. (Configuring **TOM[i]_CH[x]_CTRL.UDMODE** =0b01 if in counting up−down mode.)
2. Write new desired values to **TOM[i]_CH[x]_SR0** , **TOM[i]_CH[x]_SR1** , **TOM[i]_CH[x]_CTRL.SL** , **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** .
3. Enable update of the action register by setting the channel specific configuration bit **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** = 0b01.

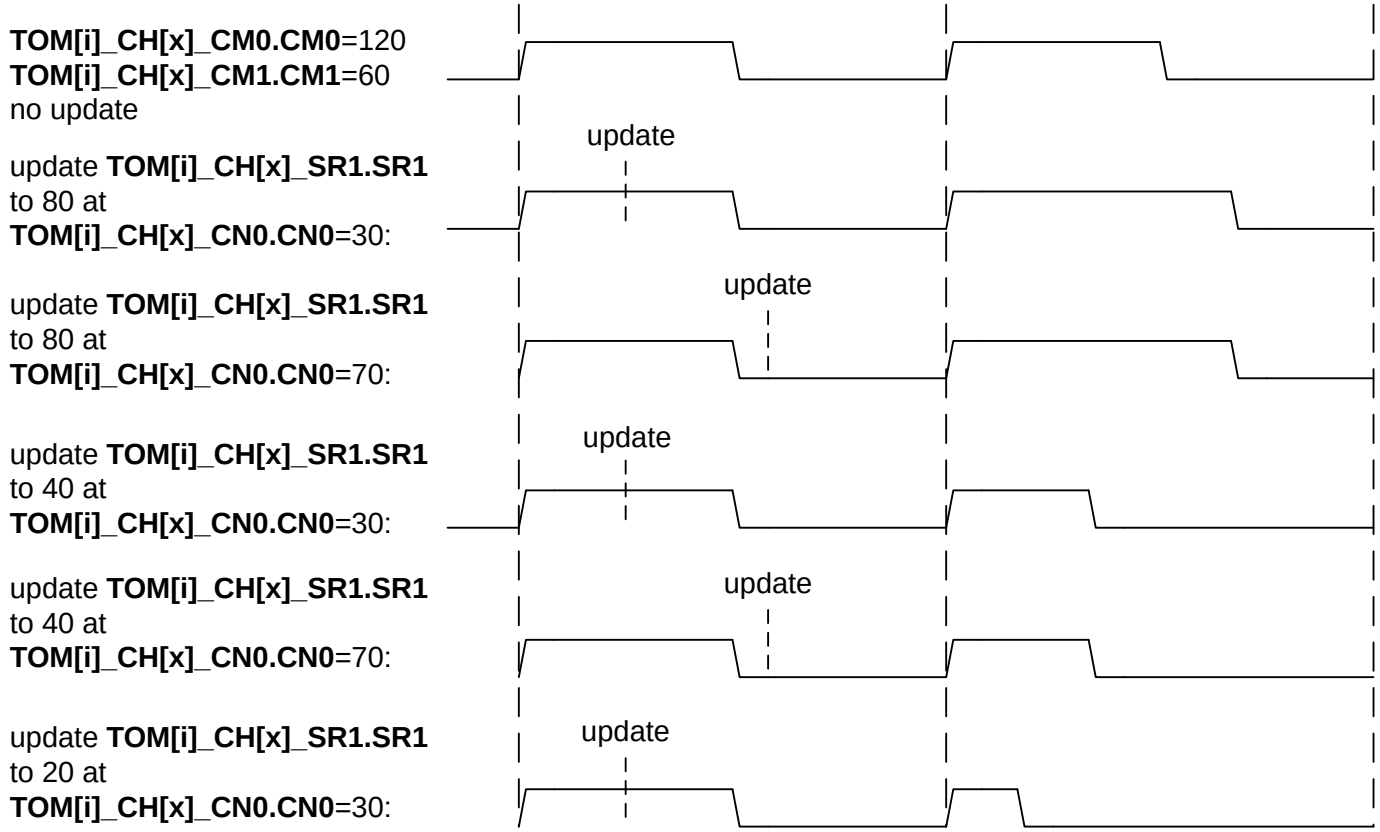
Asynchronous update of the operation registers can be realized by hot reconfiguring directly the registers **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** , **TOM[i]_CH[x]_CTRL.CLK_SRC** or hot reconfiguring their shadow registers and then using the force update mechanism (configuring **TOM[i]_TGC[g]_FUPD_CTRL.FUPD_CTRL[c]** = 0b01, **TOM[i]_TGC[g]_FUPD_CTRL.RSTCN0_CH[c]** = 0b01 and then writing bit **TOM[i]_TGC[g]_GLB_CTRL.HOST_TRIG** = 1). Such asynchronous update mechanism is sometimes helpful to change the PWM parameters of the current running PWM signal. For example, in continuous counting up mode, the current running PWM period can be reduced by asynchronously updating the value of **TOM[i]_CH[x]_CM0.CM0** with a smaller value before the end of the current period. However, since the behavior depends on the access or update time of registers and the current running operation phase of the channel, which is often difficult to control, then the behavior after an asynchronous update can be unpredictable and unexpected. Therefore, such an asynchronous update must be carefully performed in real application. The following steps are recommended to apply the force update mechanism on channel x:

1. Disable the update of the action register with the content of the corresponding shadow register by setting the channel specific configuration bit field **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** = 0b01.
2. Write new desired values to **TOM[i]_CH[x]_SR0.SR0** , **TOM[i]_CH[x]_SR1.SR1** , **TOM[i]_CH[x]_CTRL_SR.SL_SR** , **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** .
3. Enable the force update of the operation registers by setting the channel specific configuration bit field **TOM[i]_TGC[g]_FUPD_CTRL.FUPD_CTRL[c]** = 0b10, **TOM[i]_TGC[g]_FUPD_CTRL.RSTCN0_CH[c]** = 0b01.
4. Write **TOM[i]_TGC[g]_GLB_CTRL.HOST_TRIG** = 1 to trigger the force update.

14.3.3.1 Synchronous Update Of Pulse Width Only

A synchronous update of the pulse width only can be done by simply writing the desired new value to register **TOM[i]_CH[x]_SR1.SR1** without prior disable of the update mechanism (as described in the chapter above). The new pulse width is then applied in the period following the period where the update of register **TOM[i]_CH[x]_SR1.SR1** is done.

Figure 62 Synchronous Update of Pulse Width Only

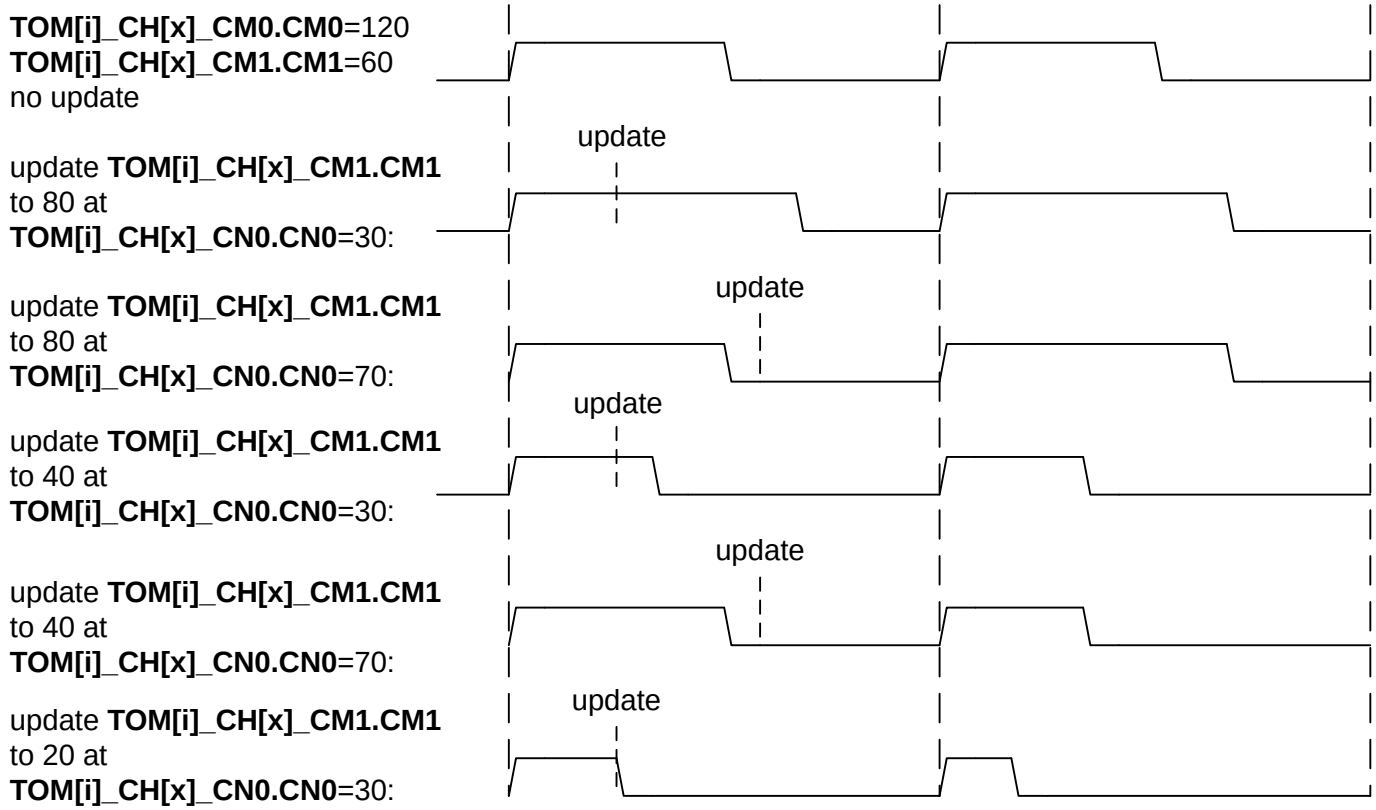


14.3.3.2 Asynchronous Update Of Pulse Width Only

If the update of the pulse width should be performed independent of the start of a new period (asynchronous), it is mandatory to additionally disable the synchronous update mechanism as a whole (i.e. configuring bits **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** =0b01 of corresponding channel x) and then update the operation register. On the one hand, the desired new value can be written directly to register **TOM[i]_CH[x]_CM1.CM1** . On the other hand, it can be realized by writing the new value to **TOM[i]_CH[x]_SR1.SR1** and afterwards updating the value of **TOM[i]_CH[x]_CM1.CM1** by using the asynchronous force update mechanism (configuring **TOM[i]_TGC[g]_FUPD_CTRL.FUPD_CTRL[c]** = 0b10, **ATOM[i]_TGC[g]_FUPD_CTRL.RSTCN0_CH[c]** = 0b01 and then writing bit **TOM[i]_TGC[g]_GLB_CTRL.HOST_TRIG** = 1).

Depending on the time of update of **TOM[i]_CH[x]_CM1.CM1** in relation to the actual value of **TOM[i]_CH[x]_CN0.CN0** and **TOM[i]_CH[x]_CM1.CM1** , the new pulse width is applied in the current period or the following period. The behavior of the output signal due to the different possibilities of an asynchronous update during a PWM period is shown in figure *63 "Asynchronous Update of Pulse Width Only"* . The new pulse width may jitter from update to update by maximum one period (given by **TOM[i]_CH[x]_CM0.CM0**). However, the period remains unchanged. In any case the creation of glitches is avoided.

Figure 63 Asynchronous Update of Pulse Width Only



14.4 Continuous Counting Up Mode

In continuous counting up mode, the TOM channel starts incrementing the counter **TOM[i]_CH[x]_CN0.CN0** once it is enabled by setting the corresponding bits in register **TOM[i]_TGC[g]_ENDIS_STAT** (refer to chapter *14.2.2 "TGC Sub-unit"* for details of enabling a TOM channel). In this mode with **TOM[i]_CH[x]_CTRL.UDMODE** =0b00 (i.e. **TOM[i]_CH[x]_CN0.CN0** counts only up), depending on configuration bits **TOM[i]_CH[x]_CTRL.RST_CCU0** the counter register **TOM[i]_CH[x]_CN0.CN0** can be reset either when the counter value is equal to the compare value **TOM[i]_CH[x]_CM0.CM0** (i.e. **TOM[i]_CH[x]_CN0.CN0** counts only from 0 to **TOM[i]_CH[x]_CM0.CM0** −1 and is then reset to 0) or when the trigger signal *TOM_CH_TRIGOUT* [x−1:x−1] of the preceding channel x−1 (which can also be the last channel of preceding instance TOM[i−1]) is captured by the TOM[i] or when there is trigger signal *TOM_EXT_TRIGIN* [x:x] from TIM[i] after it was synchronized to the selected clock resolution signal.
In this case, if **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** =0b10, also the operation register **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** are updated from their shadow registers together with **TOM[i]_CH[x]_CN0.CN0** reset.

The duration of the pulse high or low time and period is measured with the counter in sub-unit CCU0. The trigger of the counter is one of the eight fixed CMU clock signals configurable in the channel control register **TOM[i]_CH[x]_CTRL.CLK_SRC** . The register **TOM[i]_CH[x]_CM0.CM0** holds the duration of the period and the register **TOM[i]_CH[x]_CM1.CM1** holds the duration of the pulse width in clock ticks of the selected CMU clock *SEL_FXCMU_CLKEN* . The signal level of the generated output signal can be configured with the configuration bit **TOM[i]_CH[x]_CTRL.SL** .

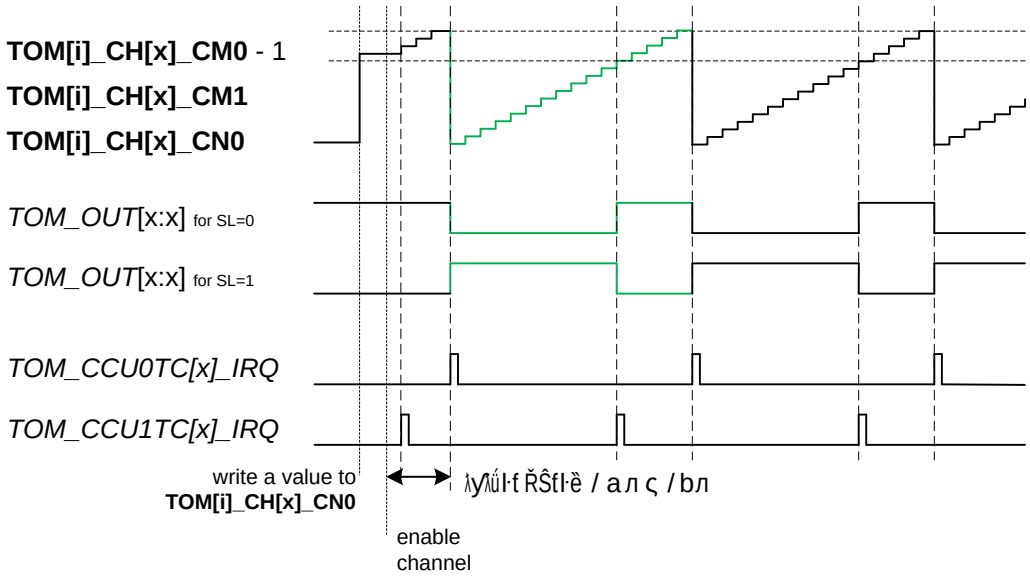
If counter **TOM[i]_CH[x]_CN0.CN0** of channel x is reset by its own CCU0 unit (i.e. the compare match of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** −1 configured by **TOM[i]_CH[x]_CTRL.RST_CCU0** =0), the PWM output behavior is shown in figure *64 "PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=0"* and the following statements are valid:

- After TOM i channel x is enabled, **TOM[i]_CH[x]_CN0.CN0** counts firstly from its initial value to **TOM[i]_CH[x]_CM0.CM0** −1 and is then reset to 0. This phase is defined as initial delay that can be flexibly configured as by the value of (**TOM[i]_CH[x]_CM0.CM0** − **TOM[i]_CH[x]_CN0.CN0**) multiplied with the selected fixed CMU clock period.
- When **TOM[i]_CH[x]_CN0.CN0** is reset from **TOM[i]_CH[x]_CM0.CM0** −1 to 0, an edge to **TOM[i]_CH[x]_CTRL.SL** is generated.
- When **TOM[i]_CH[x]_CN0.CN0** is incrementing and reaches **TOM[i]_CH[x]_CM1.CM1** (**TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1**), an edge to ! **TOM[i]_CH[x]_CTRL.SL** is generated.
- **TOM[i]_CH[x]_CN0.CN0** always counts from 0 to **TOM[i]_CH[x]_CM0.CM0** −1 and is then reset to 0.
- If **TOM[i]_CH[x]_CM0.CM0** =0 or **TOM[i]_CH[x]_CM0.CM0** =1, the counter **TOM[i]_CH[x]_CN0.CN0** is constant 0. It is highly recommended to configure **TOM[i]_CH[x]_CM0.CM0** >1.
- If **TOM[i]_CH[x]_CM1.CM1** =0 and **TOM[i]_CH[x]_CM0.CM0** >1, the output is ! **TOM[i]_CH[x]_CTRL.SL** (i.e. 0% duty cycle PWM signal).

- If **TOM[i]_CH[x]_CM1.CM1** >= **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM0.CM0** >1, the output is **TOM[i]_CH[x]_CTRL.SL** (i.e. 100% duty cycle PWM signal).

Since the *TOM_OUT* [x:x] signal level is defined as ! **TOM[i]_CH[x]_CTRL.SL** when the TOM channel is disabled without setting freeze mode, the first PWM period can be shifted earlier by writing an initial offset value to **TOM[i]_CH[x]_CN0.CN0** register. By doing this, the TOM channel first counts until **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** and then it toggles the output signal at *TOM_OUT* [x:x].

Figure 64 PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=0



In this case, as shown in figure *64 "PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=0"* , 2 interrupt signals *TOM_CCU0TC[x]_IRQ* and *TOM_CCU1TC[x]_IRQ* of channel x can be generated together with output generation *TOM_OUT* [x:x] and both of them are shown on *TOM_IRQ[x:x]* if **TOM[i]_CH[x]_IRQ_EN** =0b11. When **TOM[i]_CH[x]_CN0.CN0** is reset from **TOM[i]_CH[x]_CM0.CM0** −1 to 0, *TOM_CCU0TC[x]_IRQ* of channel x is generated. When **TOM[i]_CH[x]_CN0.CN0** is incrementing to reach **TOM[i]_CH[x]_CM1.CM1** , *TOM_CCU1TC[x]_IRQ* of channel x is generated. Especially, if the initial configuration before enabling the channel fulfills the condition **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1** − 1, an interrupt is generated after 1 fixed CMU clock period on *TOM_CCU1TC[x]_IRQ* . Similarly but not shown in the figure, if the initial configuration before enabling the channel fulfills the condition **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** − 1, an interrupt is generated after 1 selected CMU clock period on *TOM_CCU0TC[x]_IRQ* . The interrupt signals are always aligned with the edges of the output *TOM_OUT* [x:x].

If the counter register **TOM[i]_CH[x]_CN0.CN0** of channel x is reset by the trigger signal coming from another channel or the assigned TIM module (configured by **TOM[i]_CH[x]_CTRL.RST_CCU0** =1), the PWM output behavior is shown in figure *65 "PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=1"* and the following statements are valid (with MAX=number of selected *CCM[i]_FXCLK_RES* cycle between two trigger signals):

- **TOM[i]_CH[x]_CN0.CN0** counts from 0 to MAX−1 and is then reset to 0 with the falling edge of *TOM_CH_TRIGIN* [x:x] or falling edge of *TOM_EXT_TRIGIN_S* that is synchronized signal *TOM_EXT_TRIGIN* [x:x] (see chapter *14.3.1 "External Trigger Interface"*).
- If **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM1.CM1** , an edge to ! **TOM[i]_CH[x]_CTRL.SL** is generated.
- If **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** , an edge to **TOM[i]_CH[x]_CTRL.SL** is generated.
- As soon as **TOM[i]_CH[x]_CN0.CN0** reaches the value of **TOM[i]_CH[x]_CM0.CM0** while **TOM[i]_CH[x]_CM1.CM1** is equal to **TOM[i]_CH[x]_CM0.CM0** , an edge to **TOM[i]_CH[x]_CTRL.SL** is generated at the output or the output remains at **TOM[i]_CH[x]_CTRL.SL** level depending on the former level of the output (**TOM[i]_CH[x]_CM0.CM0** has higher priority). Please note that this configuration is not suitable for generating 100% duty cycle.
- If **TOM[i]_CH[x]_CM0.CM0** =0 and **TOM[i]_CH[x]_CM1.CM1** >MAX, the output is a pulse of **TOM[i]_CH[x]_CTRL.SL** for the period MAX (i.e. 100% duty cycle PWM signal).
- If **TOM[i]_CH[x]_CM0.CM0** > MAX and **TOM[i]_CH[x]_CM1.CM1** =0, the output is ! **TOM[i]_CH[x]_CTRL.SL** (i.e. 0% duty cycle PWM signal).

In this case, as shown in figure *65 "PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=1"* , 2 interrupt signals *TOM_CCU0TC[x]_IRQ* and *TOM_CCU1TC[x]_IRQ* of channel x can be generated together with output generation *TOM_OUT* [x:x] and both of them can be shown on *TOM_IRQ* [x:x] if **TOM[i]_CH[x]_IRQ_EN** =0b11. *TOM_CCU0TC[x]_IRQ* of channel x is generated when **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** while *TOM_CCU1TC[x]_IRQ* of channel x is generated when **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM1.CM1** .

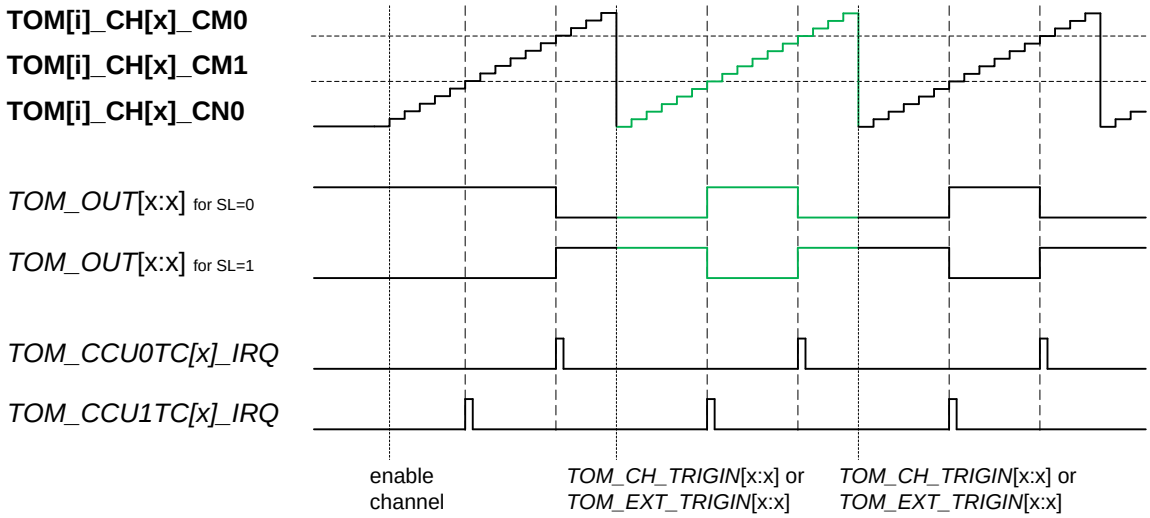
The hardware ensures that for both 0% and 100% duty cycle no glitch occurs at the output of the TOM channel.

In case of **TOM[i]_TGC[g]_GLB_CTRL.UPEN[c]=0b10** , when the counter value **TOM[i]_CH[x]_CN0.CN0** reaches the compare value in register **TOM[i]_CH[x]_CM0.CM0** (in fact **TOM[i]_CH[x]_CM0.CM0** −1) if **TOM[i]_CH[x]_CTRL.RST_CCU0** =0 or the channel captures the falling edge of the trigger signal if **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, the operation registers **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_-**

CH[x]_CM1.CM1 , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** are updated with the content of their shadow registers **TOM[i]_CH[x]_SR0.SR0** , **TOM[i]_CH[x]_SR1.SR1** , **TOM[i]_CH[x]_CTRL_SR.SL_SR** and **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** .

The operation registers **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** can also be asynchronously updated from their shadow registers with the force update mechanism. Please refer to chapter *14.3.3 "Pulse Width, Period, Signal Level and Clock Frequency Update Mechanisms"* for the force update configuration steps and more details. In continuous counting up mode, the force update mechanism can be also used to stop the current running PWM signal generation and restart a new PWM period with new PWM parameters by additionally setting **TOM[i]_TGC[g]_FUPD_CTRL.RSTCN0_CH[c]** = 0b10 together with **TOM[i]_TGC[g]_FUPD_CTRL.FUPD[c]** = 0b10. That means the force update event will not only update the operation registers from their shadow registers, but also simultaneously reset the counter to restart the next new PWM period with setting the output as **TOM[i]_CH[x]_CTRL.SL** if **TOM[i]_CH[x]_CTRL.RST_CCU0** =0 or without changing the output if **TOM[i]_CH[x]_CTRL.RST_CCU0** =1.

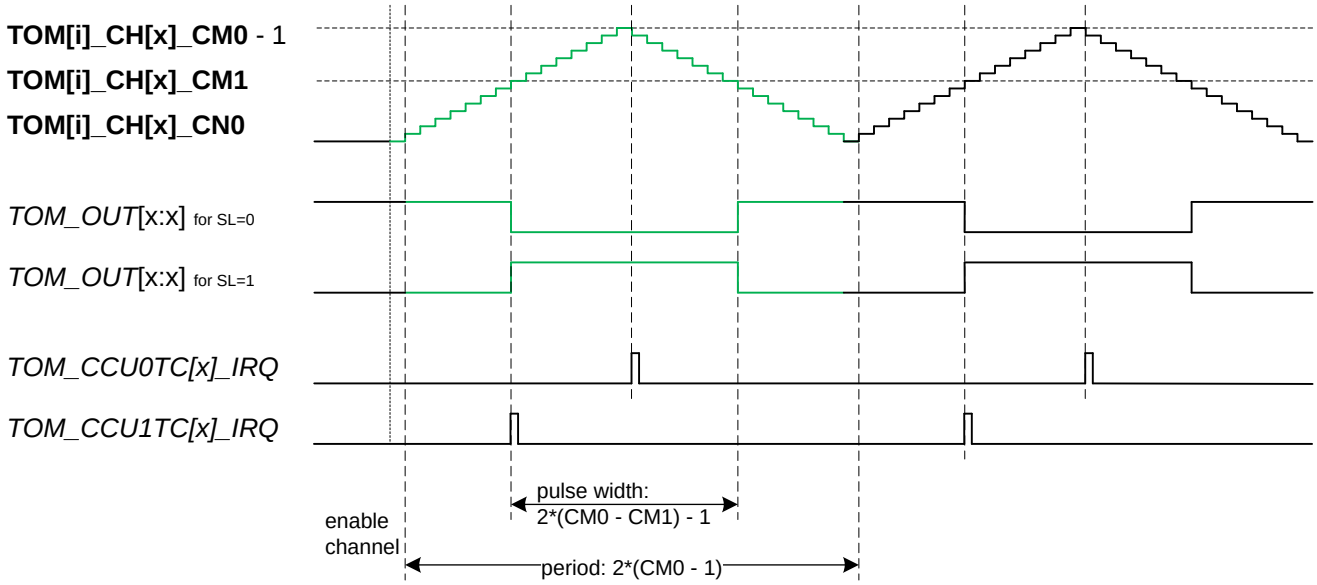
Figure 65 PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=1



14.5 Continuous Counting Up–Down Mode

In continuous counting up–down mode, if **TOM[i]_CH[x]_CN0.CN0** counts up and down (**TOM[i]_CH[x]_CTRL.UDMODE** != 0b00). Depending on configuration bits **TOM[i]_CH[x]_CTRL.RST_CCU0** , the counter **TOM[i]_CH[x]_CN0.CN0** changes the direction either when the counter value is equal to the compare value **TOM[i]_CH[x]_CM0.CM0** –1, or when has counted down to 0 or when triggered by the internal trigger signal *TOM_CH_TRIGIN* [x:x] (i.e. *TOM_CH_TRIGOUT* [x–1:x–1] from the preceding channel [x–1]) or the external trigger signal *TOM_EXT_TRIGIN* [x:x] from TIM module [i].

Figure 66 PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up–Down Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=0



The clock of the counter **TOM[i]_CH[x]_CN0.CN0** can be one of the CMU clocks *CCM[i]_FXCLK_RES* .
The clock for **TOM[i]_CH[x]_CN0.CN0** is defined by **TOM[i]_CH[x]_CTRL.CLK_SRC** value in register **TOM[i]_CH[x]_CTRL** .

If **TOM[i]_CH[x]_CTRL.RST_CCU0** =0, the counter register **TOM[i]_CH[x]_CN0.CN0** of channel x will change its counting direction by the compare match of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** –1 in its own CCU0 unit and the following statements are valid:

- In each PWM period, **TOM[i]_CH[x]_CN0.CN0** firstly counts continuously up from 1 to **TOM[i]_CH[x]_CM0.CM0** −1 and then down to 0. So period = 2*(**TOM[i]_CH[x]_CM0.CM0** −1).
- If **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1** , the output is *TOM_OUT* [x:x] set to **TOM[i]_CH[x]_CTRL.SL** .
- If **TOM[i]_CH[x]_CN0.CN0** < **TOM[i]_CH[x]_CM1.CM1** , the output is *TOM_OUT* [x:x] set to ! **TOM[i]_CH[x]_CTRL.SL** . So pulse width = 2*(**TOM[i]_CH[x]_CM0.CM0** − **TOM[i]_CH[x]_CM1.CM1**)−1.
- If **TOM[i]_CH[x]_CM0.CM0** <= 1, the counter behavior of **TOM[i]_CH[x]_CN0.CN0** is unexpected (not recommended).
- If **TOM[i]_CH[x]_CM1.CM1** = 0 and also **TOM[i]_CH[x]_CM0.CM0** > 1, the output *TOM_OUT* [x:x] is **TOM[i]_CH[x]_CTRL.SL** (i.e. 100% duty cycle).
- If **TOM[i]_CH[x]_CM1.CM1** >= **TOM[i]_CH[x]_CM0.CM0** and also **TOM[i]_CH[x]_CM0.CM0** > 1, the output *TOM_OUT* [x:x] is ! **TOM[i]_CH[x]_CTRL.SL** (i.e. 0% duty cycle).

This behavior is depicted in figure *66 "PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up-Down Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=0"* . In this case, if **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[x]** =1, the operation registers **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** can be updated when **TOM[i]_CH[x]_CN0.CN0** = 0 (i.e. counting direction changes from down to up) or **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** −1 (i.e. counting direction changes from up to down) depending on various configurations of **TOM[i]_CH[x]_CTRL.UDMODE** (see the register description for more details). It is particularly not recommended to change the value of **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** when **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** −1. Otherwise, the behavior may be unexpected.

Like continuous counting up mode, interrupt signals *TOM_CCU0TC[x]_IRQ* and *TOM_CCU1TC[x]_IRQ* of channel x are generated together with PWM signal output edges. When **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM1.CM1** at the first time and the first edge is generated, an interrupt on *TOM_CCU1TC[x]_IRQ* is generated. When **TOM[i]_CH[x]_CN0.CN0** changes its counting direction from up to down, an interrupt on *TOM_CCU0TC[x]_IRQ* is generated.

If **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, the counter register **TOM[i]_CH[x]_CN0.CN0** of channel x will change its counting direction by the trigger signal coming from another channel or the assigned TIM module and the following statements are valid:

- **TOM[i]_CH[x]_CN0.CN0** firstly counts continuously up. At the falling edge of the trigger signal pulse *TOM_CH_TRIGIN* [x:x] or *TOM_EXT_TRIGIN_S* (the synchronized signal *TOM_EXT_TRIGIN* [x:x], see chapter *14.3.1 "External Trigger Interface"*), the counter switches to count down mode. If **TOM[i]_CH[x]_CN0.CN0** reaches 0, it will count up again.
- If **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1** , the output *TOM_OUT* [x:x] is set to **TOM[i]_CH[x]_CTRL.SL** .
- If **TOM[i]_CH[x]_CN0.CN0** < **TOM[i]_CH[x]_CM1.CM1** , the output is *TOM_OUT* [x:x] set to ! **TOM[i]_CH[x]_CTRL.SL** .
- If **TOM[i]_CH[x]_CM1.CM1** = 0, the output *TOM_OUT* [x:x] is **TOM[i]_CH[x]_CTRL.SL** (i.e. 100% duty cycle).
- If **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** , the output *TOM_OUT_T* [x:x] is set to **TOM[i]_CH[x]_CTRL.SL** .
- If **TOM[i]_CH[x]_CN0.CN0** < **TOM[i]_CH[x]_CM0.CM0** , the output is *TOM_OUT_T* [x:x] set to ! **TOM[i]_CH[x]_CTRL.SL** .
- If **TOM[i]_CH[x]_CM0.CM0** = 0, the output *TOM_OUT_T* [x:x] is **TOM[i]_CH[x]_CTRL.SL** (i.e. 100% duty cycle).

This behavior is depicted in figure *67 "PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up-Down Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=1"* . In this case, if **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[x]** =1, the operation registers **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** can also be updated when **TOM[i]_CH[x]_CN0.CN0** = 0 (i.e. counting direction changes from down to up) or **TOM[i]_CH[x]_CN0.CN0** reaches its maximal value (i.e. counting direction changes from up to down) depending on various configurations of **TOM[i]_CH[x]_CTRL.UDMODE** (see the register description for more details). Especially, it is not recommended to change the value of **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** when **TOM[i]_CH[x]_CN0.CN0** its maximal value (i.e. counting direction changes from up to down). Otherwise, the behavior may be unexpected.

If **TOM[i]_CH[x]_IRQ_EN** =0b11, interrupt signals *TOM_CCU0TC[x]_IRQ* and *TOM_CCU1TC[x]_IRQ* of channel x are generated together with PWM signal output. When **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM1.CM1** at the first time and the first edge of *TOM_OUT* [x:x] is generated, an interrupt on *TOM_CCU1TC[x]_IRQ* is generated. When **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** at the first time and the first edge of *TOM_OUT_T* [x:x] is generated, an interrupt on *TOM_CCU0TC[x]_IRQ* is generated.

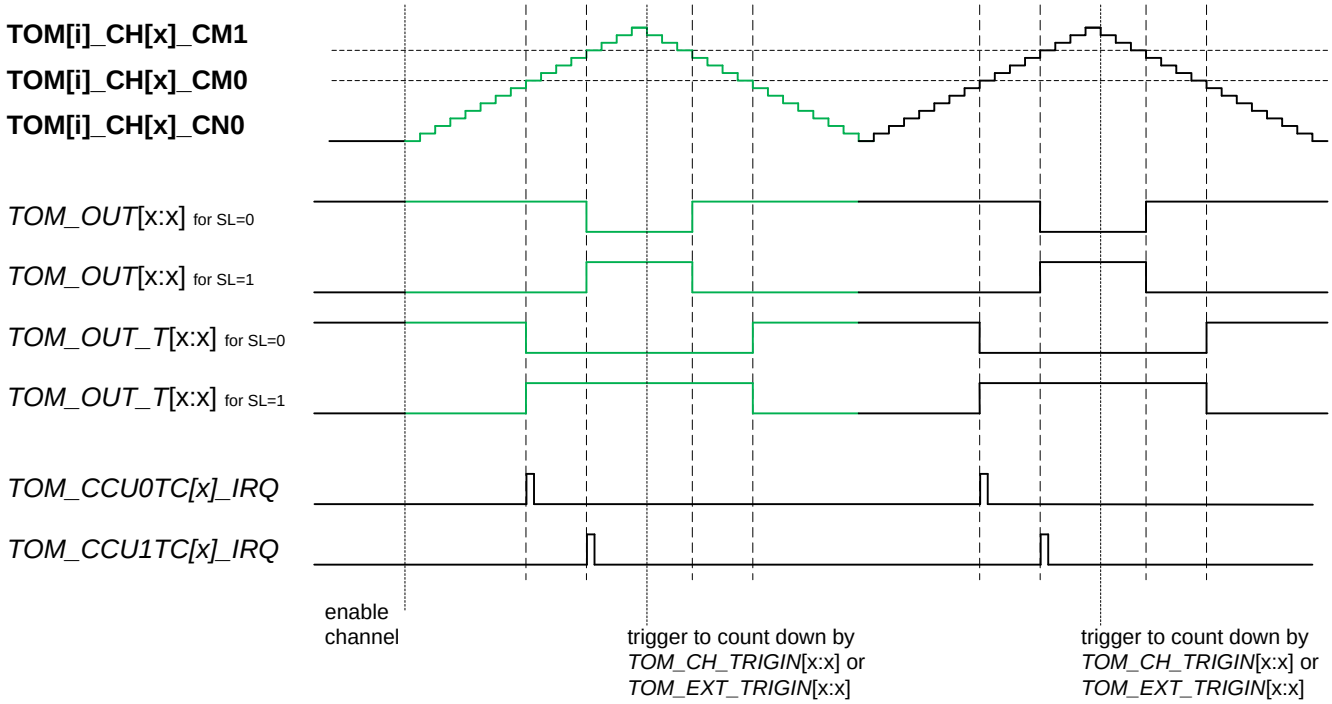
In case of continuous counting up-down mode and **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, it is recommended to configure the TOM channel according to the following statements:

- The triggering channel and the triggered channel are both running in up-down mode.
- The time between two trigger signals is equal to the time needed for **TOM[i]_CH[x]_CN0.CN0** of triggered channel to count back to 0 and again up to the same upper value.

The second recommendation can be fulfilled by synchronizing the start of the triggering channel and the triggered channel, i.e. let both channels start with **TOM[i]_CH[x]_CN0.CN0** = 0 and same clock source.

If there is a pipeline register in the trigger chain, the additional delay of one clock period has to be taken into account by starting the triggering channel with **TOM[i]_CH[x]_CN0.CN0** = 1 (i.e. 1 greater than **TOM[i]_CH[x]_CN0.CN0** of the triggered channel).

Figure 67 PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up-Down Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=1



14.6 One-Shot Counting Up Mode

The TOM channel can operate in one-shot mode when the **TOM[i]_CH[x]_CTRL.OSM** bit is set. One-shot mode means that a single pulse with the pulse level defined in bit **TOM[i]_CH[x]_CTRL.SL** is generated on the output signal.

Firstly, the channel has to be enabled by setting the corresponding **TOM[i]_TGC[g]_ENDIS_STAT.ENDIS_STAT[c]** .

Unlike in the continuous counting modes, the counter **TOM[i]_CH[x]_CN0.CN0** will not be incremented once the channel is enabled. Instead, according to the configuration bit **TOM[i]_CH[x]_CTRL.OSM_TRIG** , the start of counting and pulse generation is triggered by a write access of **TOM[i]_CH[x]_CN0.CN0** or the internal trigger signal *TOM_CH_TRIGIN* [x:x] or the external trigger signal *TOM_EXT_TRIGIN* [x:x].

If SPE mode of TOM[i] channel x (x∈{2, 6, 7, 8, 9}) is enabled (set bit **TOM[i]_CH[x]_CTRL.SPEM** (x∈{2, 6, 7, 8, 9})), also the trigger signal *SPE[i]_NIPD* can trigger the reset of register **TOM[i]_CH[x]_CN0.CN0** to zero and a start of the pulse generation.

When **TOM[i]_CH[x]_CTRL.OSM_TRIG** =0, a write access to the register **TOM[i]_CH[x]_CN0.CN0** triggers the start of incrementing the counter register **TOM[i]_CH[x]_CN0.CN0** and pulse generation.

If the time between the write access and the first edge generation of one-shot PWM signal is defined as initial delay, the initial delay is determined by the value of (**TOM[i]_CH[x]_CM0.CM0** - **TOM[i]_CH[x]_CN0.CN0**) multiplied with the selected CMU clock period.

Especially, if writing a value of greater than or equal to **TOM[i]_CH[x]_CM0.CM0** -1 to **TOM[i]_CH[x]_CN0.CN0** and configuring the initial clock source as cluster clock (the clock source at reset state), the initial delay is minimum, i.e. 1 cluster clock period *CLS[i]_CLK* . That means after the write access of **TOM[i]_CH[x]_CN0.CN0** , the first edge of the output with the interrupt will be set in 1 *CLS[i]_CLK* period.

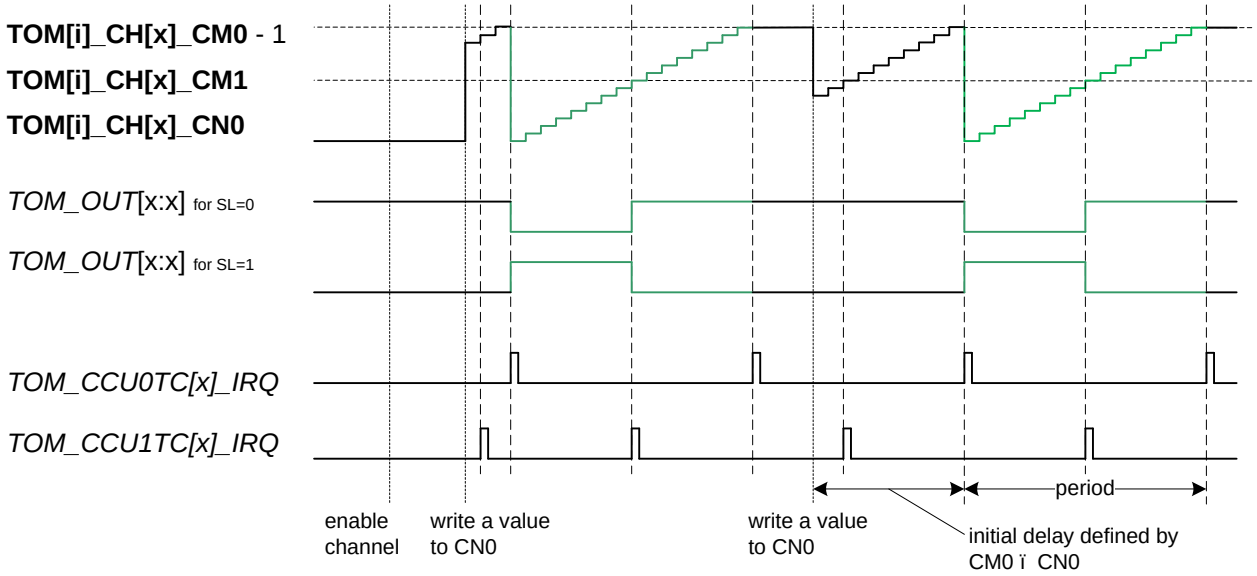
If the counter **TOM[i]_CH[x]_CN0.CN0** is reset from **TOM[i]_CH[x]_CM0.CM0** -1 back to zero, the first edge at *TOM_OUT* [x:x] is generated. And the operation phase of the pulse generation starts incrementing **TOM[i]_CH[x]_CN0.CN0** from 0 to **TOM[i]_CH[x]_CM0.CM0** -1 again.

The second edge in the operation phase is generated if **TOM[i]_CH[x]_CN0.CN0** increments until it reaches **TOM[i]_CH[x]_CM1.CM1** .

If the counter **TOM[i]_CH[x]_CN0.CN0** reaches the value of **TOM[i]_CH[x]_CM0.CM0** -1 for the second time, the counter will stop.

Figure *68 "PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Up Mode if TOM[i]_CH[x]_CRTL.OSM_TRIG=0"* depicts the pulse generation process mentioned above. The generation of the interrupt signals during the generation of the PWM signal is similar as described in continuous counting up mode. Furthermore, if the value written to **TOM[i]_CH[x]_CN0.CN0** for triggering or retriggering the counter is greater than or equal to **TOM[i]_CH[x]_CM1.CM1** -1, an interrupt on *TOM_CCU1TC[x]_IRQ* is generated after 1 selected CMU clock period. Similarly but not shown in the figure, if the value written to **TOM[i]_CH[x]_CN0.CN0** for triggering or retriggering the counter is greater than or equal to **TOM[i]_CH[x]_CM0.CM0** - 1, an interrupt on *TOM_CCU0TC[x]_IRQ* is generated after 1 selected CMU clock period. Afterwards when **TOM[i]_CH[x]_CN0.CN0** is reset from **TOM[i]_CH[x]_CM0.CM0** -1 to 0, *TOM_CCU0TC[x]_IRQ* of channel x is generated. When **TOM[i]_CH[x]_CN0.CN0** is incrementing to reach **TOM[i]_CH[x]_CM1.CM1** , *TOM_CCU1TC[x]_IRQ* of channel x is generated.

Figure 68 PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Up Mode if TOM[i]_CH[x]_CRTL.OSM_TRIG=0



Retriggering an one-shot cycle (while **TOM[i]_CH[x]_CN0.CN0** is already incrementing) by writing a value to **TOM[i]_CH[x]_CN0.CN0** is possible but depends on the phase of **TOM[i]_CH[x]_CN0.CN0** :

- Phase 1: update of **TOM[i]_CH[x]_CN0.CN0** before **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** for the first time (initial phase)
- Phase 2: update of **TOM[i]_CH[x]_CN0.CN0** after **TOM[i]_CH[x]_CN0.CN0** has reached **TOM[i]_CH[x]_CM0.CM0** for the first time until the current single PWM pulse is finished (operation phase)

In phase 1 (initial phase): writing a value CN0(new) to counter **TOM[i]_CH[x]_CN0.CN0** (CN0(new)< **TOM[i]_CH[x]_CM0.CM0**) leads to a shift of the first edge by the time **TOM[i]_CH[x]_CM0.CM0** -CN0(new), i.e. the first edge is generated when **TOM[i]_CH[x]_CN0.CN0** is reset from **TOM[i]_CH[x]_CM0.CM0** -1 back to 0 for the first time.

In phase 2 (operation phase): writing to counter **TOM[i]_CH[x]_CN0.CN0** is not possible and protected by the hardware.

A failed write access to **TOM[i]_CH[x]_CN0.CN0** in phase 2 (operation phase) is indicated by the status register flag **TOM[i]_CH[x]_STAT.OSM_RTF** .

After one PWM pulse is ended, other single pulses can be further triggered by a write access to register **TOM[i]_CH[x]_CN0.CN0** .

It is recommended to enable the update mechanism by setting of **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** = 0b10 for retriggering the second one-shot PWM signal generation with different initial delay and different PWM parameters. Similar to the continuous counting up mode with **TOM[i]_CH[x]_CTRL.RST_CCU0** =0, the operation registers **TOM[i]_CH[x]_CM0.CM0** / **TOM[i]_CH[x]_CM1.CM1** / **TOM[i]_CH[x]_CTRL.SL** / **TOM[i]_CH[x]_CTRL.CLK_SRC** are always synchronously updated when **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** -1. Firstly, after the first one-shot PWM signal is generated, the new values for initial delay (phase 1) can be written into the shadow registers **TOM[i]_CH[x]_SR0.SR0** / **TOM[i]_CH[x]_SR1.SR1** / **TOM[i]_CH[x]_CTRL_SR.SL_SR** / **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** before the retriggering is executed by writing of register **TOM[i]_CH[x]_CN0.CN0** . The shadow registers should be immediately loaded to the operation registers **TOM[i]_CH[x]_CM0.CM0** / **TOM[i]_CH[x]_CM1.CM1** / **TOM[i]_CH[x]_CTRL.SL** / **TOM[i]_CH[x]_CTRL.CLK_SRC** , because **TOM[i]_CH[x]_CN0.CN0** keeps as **TOM[i]_CH[x]_CM0.CM0** -1. When **TOM[i]_CH[x]_CN0.CN0** is written to be retriggered afterwards, the initial delay is defined according to the updated operation registers. Secondly, during the initial delay phase (phase 1), the new parameters for PWM signal can be written into the shadow registers again. The operational registers will be updated with the shadow registers when **TOM[i]_CH[x]_CN0.CN0** is reset from **TOM[i]_CH[x]_CM0.CM0** -1 at the end of phase 1 and phase 2 is started to generate the second one-shot PWM signal with the new updated operation registers.

If a channel is configured to one-shot mode and configuration bit **TOM[i]_CH[x]_CTRL.OSM_TRIG** is set to 1, the trigger signal only triggers start of one pulse generation.

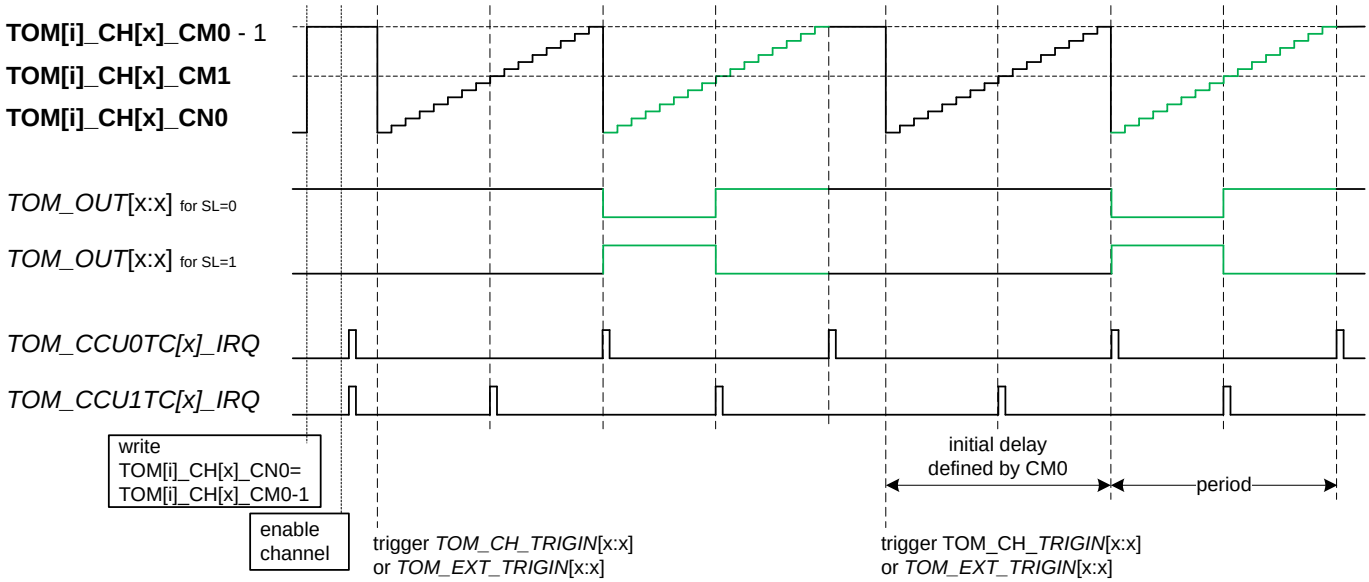
If a channel is configured to one-shot mode and configuration bit **TOM[i]_CH[x]_CTRL.OSM_TRIG** is set to 1, the trigger signal *TOM_CH_TRIGIN* [x:x] or *TOM_EXT_TRIGIN* [x:x] triggers start of pulse generation as shown in figure *69 "PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Mode if TOM[i]_CH[x]_CTRL.OSM_TRIG=1"* . In this case, it is not allowed to write **TOM[i]_CH[x]_CN0.CN0** and so the register **TOM[i]_CH[x]_STAT.OSM_RTF** is not relevant. Otherwise, a write access to **TOM[i]_CH[x]_CN0.CN0** can make an unpredictable output behavior. Furthermore, as shown in figure *69 "PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Mode if TOM[i]_CH[x]_CTRL.OSM_TRIG=1"* , **TOM[i]_CH[x]_CN0.CN0** should be initially configured as **TOM[i]_CH[x]_CM0.CM0** -1 or greater value before enabling the channel. Otherwise, it is hard to define the counting behavior and the initial delay behavior. As shown in figure *69 "PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Mode if TOM[i]_CH[x]_CTRL.OSM_TRIG=1"* , when the channel is enabled, after the falling edge of the trigger pulse *TOM_CH_TRIGIN* [x:x] or *TOM_EXT_TRIGIN_S* (the synchronized signal *TOM_EXT_TRIGIN* [x:x], see chapter *14.3.1 "External Trigger Interface"*), **TOM[i]_CH[x]_CN0.CN0** will be triggered to be reset that is synchronous with the next enable high pulse of the selected CMU clock source (i.e. synchronous to the next falling edge of the selected CMU enable signal). Then it counts from 0 to **TOM[i]_CH[x]_CM0.CM0** -1. This process is defined as initial phase. If the internal trigger signal *TOM_TRIGIN* [x:x] is in use and it is synchronous to the selected CMU clock source or the external trigger signal *TOM_EXT_TRIGIN* [x:x] is in use and it must be synchronized to the selected CMU clock source before triggering, the initial delay can

be defined as the time between the falling edge of the synchronized trigger signal to the point of time of starting the generation of the PWM signal (green phase in the figure), i.e. **TOM[i]_CH[x]_CM0.CM0** +1 selected CMU clock periods. After that, **TOM[i]_CH[x]_CN0.CN0** will be reset and counts from 0 to **TOM[i]_CH[x]_CM0.CM0** −1 at the second time to generate one single PWM signal and then stops. This second counting is defined as operation phase.

After the first OSM signal is generated, the second trigger signal is allowed to be applied to start the counter and generate the next OSM signal after a certain initial delay. Figure *69 "PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Mode if TOM[i]_CH[x]_CTRL.OSM_TRIG=1"* shows such an application of generating 2 OSM signals without changing PWM parameters. In the case, where the second trigger signal is generated, the counter exhibits the same behavior as described above for the first OSM signal generation. That means the initial delay is also **TOM[i]_CH[x]_CM0.CM0** +1 selected CMU clock periods and the OSM signal is the same as the first one.

However, in some real applications of generating various single PWM signals, it is necessary to synchronously update the parameters in **TOM[i]_CH[x]_CM0.CM0** / **TOM[i]_CH[x]_CM1.CM1** / **TOM[i]_CH[x]_CTRL.SL** / **TOM[i]_CH[x]_CTRL.CLK_SRC** by setting **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[x]** = 0b10. In this case, these registers are updated from their shadow registers when **TOM[i]_CH[x]_CN0.CN0** is matching **TOM[i]_CH[x]_CM0.CM0** −1. After the first OSM PWM signal, if **TOM[i]_CH[x]_CM0.CM0** is updated with a new value CM0(new) before the second trigger signal comes, the initial delay of the second OSM signal depends on CM0(new) and the value CN0(old) of **TOM[i]_CH[x]_CN0.CN0** when the counter stopped at the end of the first OSM signal. The value CN0(old) is actually the value of **TOM[i]_CH[x]_CM0.CM0** −1 of the first OSM signal before the synchronous update, i.e. CN0(old)=CM0(old)−1. Like the first OSM signal generation, with the falling edge of the second trigger pulse signal, the counter will be retriggered synchronous to the next enable high pulse of the selected CMU clock source (i.e. synchronous to the next falling edge of the selected CMU enable signal). Assuming the internal trigger signal *TOM_TRIGIN* [x:x] is in use and it is synchronous to the selected CMU clock source or the external trigger signal *TOM_EXT_TRIGIN* [x:x] is in use and it will be synchronized to the selected CMU clock source before triggering, the initial delay can be defined as the time between the falling edge of the second synchronized trigger signal to the point of time of starting the generation of the PWM signal (green phase in the figure). On one hand, if CN0(old) is greater than or equal to updated CM0(new)−1, the counter will be reset and counts up to CM0(new)−1 and the initial delay is CM0(new)+1 selected CMU clock periods, i.e. **TOM[i]_CH[x]_CM0.CM0** +1 selected CMU clock periods. The behavior of the counter is the same as described above for the initial delay of the first OSM signal. On the other hand, if CN0(old) is smaller than updated CM0(new)−1, the initial phase or the initial delay is longer. In the initial phase, the counter will count up from CN0(old) to CM0(new)−1 and reset, and then increment again to CM0(new)−1. That means the initial delay is no longer CM0(new)+1, but the larger value of 2*CM0(new)−CN0(old) CMU clock periods, i.e. 2*CM0(new)−CM0(old)+1 selected CMU clock periods.

Figure 69 PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Mode if TOM[i]_CH[x]_CTRL.OSM_TRIG=1



As shown in figure *69 "PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Mode if TOM[i]_CH[x]_CTRL.OSM_TRIG=1"* , the generation of the interrupt signals during the generation of the PWM signal is similar as described in continuous counting up mode. As before enabling the channel the initial value written to **TOM[i]_CH[x]_CN0.CN0** is greater than or equal to **TOM[i]_CH[x]_CM0.CM0** − 1 and **TOM[i]_CH[x]_CM1.CM1** − 1, both interrupt signals *TOM_CCU0TC[x]_IRQ* and *TOM_CCU1TC[x]_IRQ* are generated after 1 selected CMU clock period. Afterwards when **TOM[i]_CH[x]_CN0.CN0** is reset from **TOM[i]_CH[x]_CM0.CM0** −1 to 0, *TOM_CCU0TC[x]_IRQ* of channel x is generated. When **TOM[i]_CH[x]_CN0.CN0** is incrementing to reach **TOM[i]_CH[x]_CM1.CM1** , *TOM_CCU1TC[x]_IRQ* of channel x is generated.

In one-shot mode, it is not recommended to configure **TOM[i]_CH[x]_CM0.CM0** <2. Otherwise, the behavior may be unexpected.

14.7 One-Shot Counting Up-Down Mode

The TOM channel can operate in one-shot counting up-down mode when the bit **TOM[i]_CH[x]_CTRL.OSM** = 1 and the **TOM[i]_CH[x]_CTRL.UDMODE** != 0b00. One-shot mode means that a single pulse with the pulse level defined in bit **TOM[i]_CH[x]_CTRL.SL** is generated on the output signal.

Firstly, the channel has to be enabled by setting the corresponding **TOM[i]_TGC[g]_ENDIS_STAT.ENDIS_STAT[c]** .

In one-shot mode the counter **TOM[i]_CH[x]_CN0.CN0** will not be incremented once the channel is enabled.

When **TOM[i]_CH[x]_CTRL.OSM_TRIG** =0, a write access to the register **TOM[i]_CH[x]_CN0.CN0** triggers the start of incrementing the counter register **TOM[i]_CH[x]_CN0.CN0** and pulse generation.

If the counter **TOM[i]_CH[x]_CN0.CN0** is greater than or equal to **TOM[i]_CH[x]_CM1.CM1** , the output *TOM_OUT* [x:x] is set to **TOM[i]_CH[x]_CTRL.SL** value.

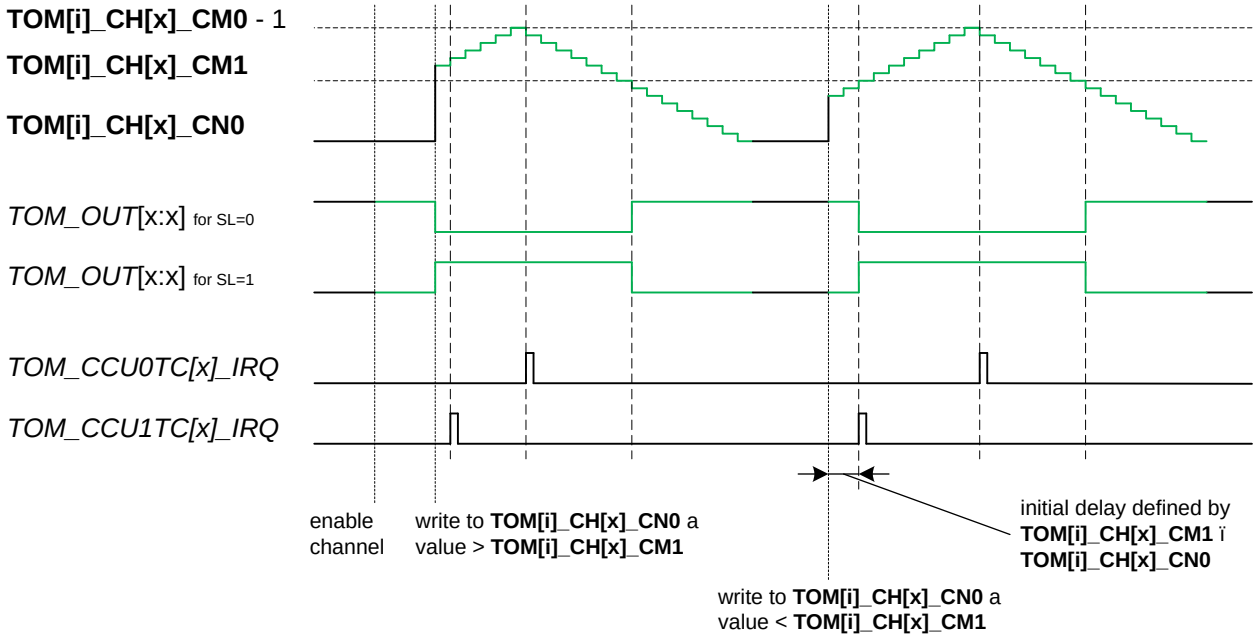
If the counter **TOM[i]_CH[x]_CN0.CN0** is less than **TOM[i]_CH[x]_CM1.CM1** , the output *TOM_OUT* [x:x] is set to ! **TOM[i]_CH[x]_CTRL.-SL** value.

If the counter **TOM[i]_CH[x]_CN0.CN0** has reached the value 0 (by counting down), it will stop.

The written value of **TOM[i]_CH[x]_CN0.CN0** determines the start delay of the first edge. The delay time of the first edge is given by (**TOM[i]_CH[x]_CM1.CM1** – **TOM[i]_CH[x]_CN0.CN0**) multiplied with the period defined by the current value of **TOM[i]_CH[x]_CTRL.CLK_-SRC** .

Figure *70 "PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Up-Down Mode if TOM[i]_CH[x]_CRTL.-OSM_TRIG= 0"* depicts the pulse generation in one-shot counting up-down mode. The interrupt signals are generated similarly as described in one-shot counting up mode and continuous counting up-down mode. The write access of **TOM[i]_CH[x]_CN0.CN0** can cause interrupts if the write value is greater than or equal to **TOM[i]_CH[x]_CM1.CM1** – 1 or **TOM[i]_CH[x]_CM0.CM0** –1. If the value written to **TOM[i]_C-H[x]_CN0.CN0** for triggering or retriggering the counter is greater than or equal to **TOM[i]_CH[x]_CM1.CM1** –1, the output *TOM_OUT* [x:x] will be set to **TOM[i]_CH[x]_CTRL.SL** after 1 selected CMU clock period together with an interrupt on *TOM_CCU1TC[x]_IRQ* . Similarly but not shown in the figure, if the value written to **TOM[i]_CH[x]_CN0.CN0** for triggering or retriggering the counter is greater than or equal to **TOM[i]_CH[x]_CM0.CM0** – 1, an interrupt on *TOM_CCU0TC[x]_IRQ* will be generated after 1 selected CMU clock period. Afterwards when **TOM[i]_CH[x]_CN0.CN0** is incrementing to reach **TOM[i]_CH[x]_CM1.CM1** , an interrupt on *TOM_CCU1TC[x]_IRQ* is generated. When **TOM[i]_CH[x]_CN0.CN0** changes its counting direction from up to down, an interrupt on *TOM_CCU0TC[x]_IRQ* is generated.

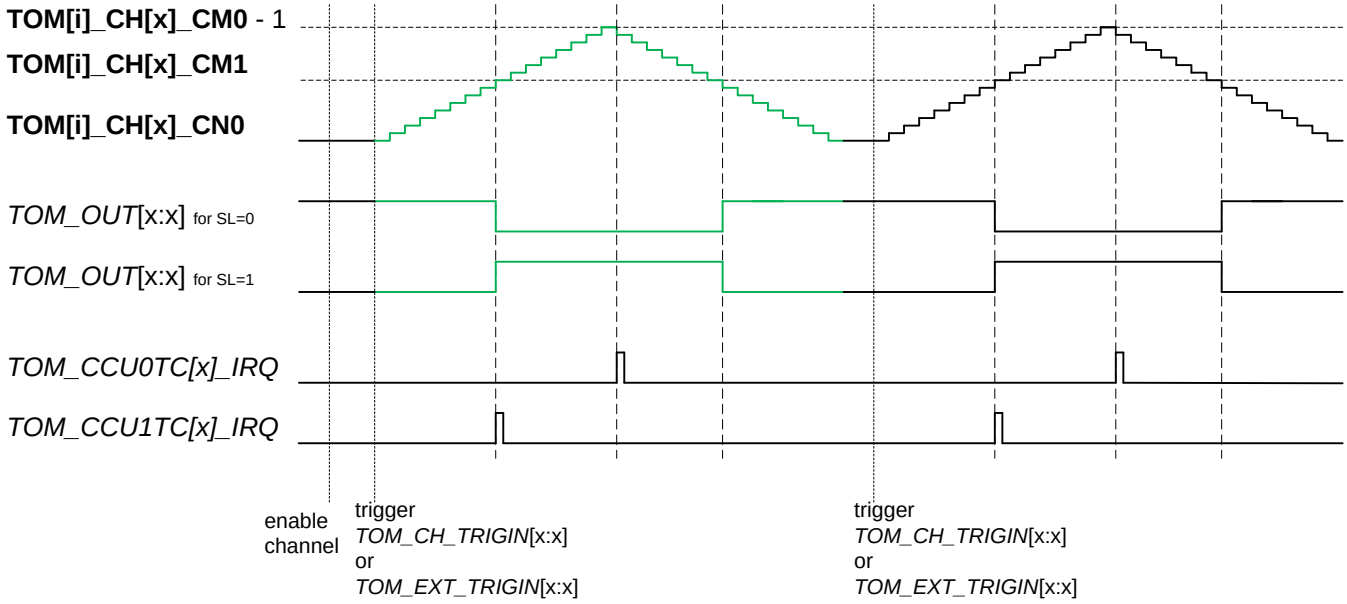
Figure 70 PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Up-Down Mode if T-OM[i]_CH[x]_CRTL.OSM_TRIG= 0



Further output of single pulses can be started by writing to register **TOM[i]_CH[x]_CN0.CN0** .

If a channel is configured to one-shot counting up-down mode and configuration bit **TOM[i]_CH[x]_CTRL.OSM_TRIG** is set to 1, the falling edge of the trigger signal *TOM_CH_TRIGIN* [x:x] or *TOM_EXT_TRIGIN_S* (i.e. in fact the synchronized signal of *TOM_EXT_TRIGI-N* [x:x]) triggers start of incrementing **TOM[i]_CH[x]_CN0.CN0** and pulse generation. The output generation with interrupt signals is similar as mentioned above. If the counter **TOM[i]_CH[x]_CN0.CN0** is greater than or equal to **TOM[i]_CH[x]_CM1.CM1** , the output *TOM_OUT* is set to **TOM[i]_CH[x]_CTRL.SL** value. Otherwise, the output *TOM_OUT* is set to ! **TOM[i]_CH[x]_CTRL.SL** value. When **TOM[i]_CH[x]_CN0.CN0** is incrementing to reach **TOM[i]_CH[x]_CM1.CM1** , an interrupt is generated on *TOM_CCU1TC[x]_IRQ* . When **TOM[i]_CH[x]_CN0.CN0** changes its counting direction from up to down, an interrupt is generated on *TOM_CCU0TC[x]_IRQ* .

Figure 71 PWM Output with Respect to Configuration Bit TOM[i]_CH[x]_CTRL.SL in One-Shot Counting Up-Down Mode if TOM[i]_CH[x]_CRTL.OSM_TRIG = 1



14.8 Pulse Count Modulation Mode

At the output *TOM_OUT* [15:15] a pulse count modulated signal can be generated instead of the simple PWM output signal. This specific behavior is called pulse count modulation (PCM) mode. PCM mode is only available when TOM[i]_CH[x]_CTRL.UDMODE=0.

Figure *60 "TOM Channel Architecture"* outlines the circuit for Pulse Count Modulation.

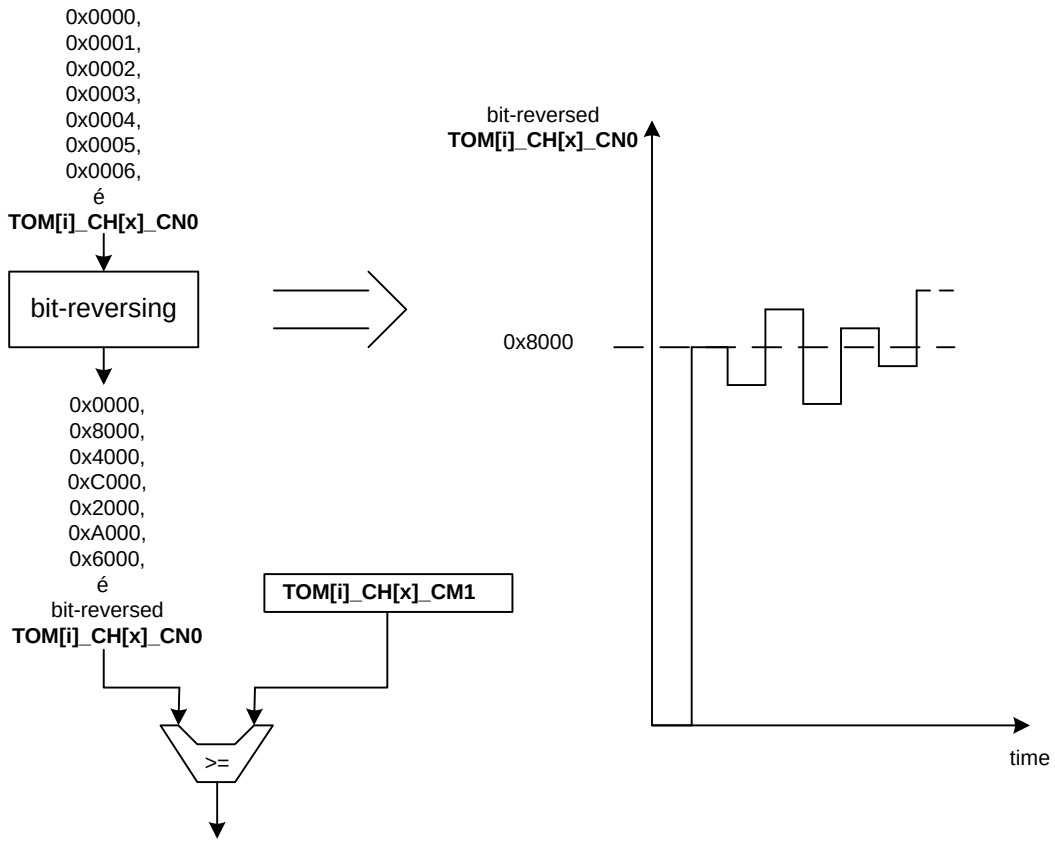
The PCM mode is enabled by setting bit **TOM[i]_CH[x]_CTRL.BITREV** to 1.

The PCM mode is only available for TOM channel 15.

With the configuration bit **TOM[i]_CH[x]_CTRL.BITREV** =1 a bit reversing of the counter output **TOM[i]_CH[x]_CN0.CN0** is configured. In this case the bits are swapped, for example, LSB and MSB, LSB+1 and MSB-1, LSB+2 and MSB-2 and so on.

The effect of bit-reversing of the **TOM[i]_CH[x]_CN0.CN0** register value is shown in the following figure *72 "Bit Reversing of Counter TOM[i]_CH[x]_CN0.CN0 Output"* .

Figure 72 Bit Reversing of Counter TOM[i]_CH[x]_CN0.CN0 Output



In the PCM mode the counter **TOM[i]_CH[x]_CN0.CN0** is incremented by every clock tick depending on configured CMU clock (*CCM[i]_FXCLK_RES*).

The output of counter **TOM[i]_CH[x]_CN0.CN0** is first bit-reversed and then compared with the configured register value **TOM[i]_CH[x]_CM1.CM1** .

When the bit-reversed value of register **TOM[i]_CH[x]_CN0.CN0** is greater than or equal to **TOM[i]_CH[x]_CM1.CM1** , the output *TOM_OUT* [15:15] is set to ! **TOM[i]_CH[x]_CTRL.SL** and otherwise set to **TOM[i]_CH[x]_CTRL.SL** .

In PCM mode the **TOM[i]_CH[x]_CM0.CM0** register, in which the period is defined, normally has to be set to its maximum value 0xFFFF.

To reduce time period of updating pulse width value in **TOM[i]_CH[x]_CM1.CM1** register, it is additionally possible to setup period value in **TOM[i]_CH[x]_CM0.CM0** register to smaller values than the maximum value as described before.
Possible values for **TOM[i]_CH[x]_CM0.CM0** register are each even numbered values to the power of 2 e.g. 0x8000, 0x4000, 0x2000 ... .
In this case the pulse width has to be configured in the following manner.
Depending on how much the period in **TOM[i]_CH[x]_CM0.CM0** register is decreased, means shifted right starting from 0x10000 – the pulse width in **TOM[i]_CH[x]_CM1.CM1** register has to be shifted left (= rotated: shift MSB back into LSB) with same value, e.g. :
Period **TOM[i]_CH[x]_CM0.CM0** = 0x0100 –> shifted 8 bits right from 0x10000
–––> so pulse width has to be shifted 8 bits left:
e.g. 50% pulse width = 0x00080 –> shift 8 bits left –> **TOM[i]_CH[x]_CM1.CM1** = 0x8000

Table 26 PCM Examples

CM0 (period)	CM1 (pulse width)	Shift	CM1 (configured value)
0xFFFF	0x8000	No shift	0x8000
0x8000	0x4000	Shift 1 bit left	0x8000
0x4000	0x1000	Shift 2 bits left	0x4000
0x2000	0x0FFF	Shift 3 bits left	0x7FF8
0x1000	0x0333	Shift 4 bits left	0x3330
0x0800	0x0055	Shift 5 bits left	0x0AA0
0x0020	0x0008	Shift 11 bits left	0x4000
0x0010	0x0005	Shift 12 bits left	0x5000

In this mode the **TOM[i]_CH[x]_IRQ_NOTIFY.CCU1TC** (see register **TOM[i]_CH[x]_IRQ_NOTIFY**) is set every time if bit reverse value of **TOM[i]_CH[x]_CN0.CN0** is greater than or equal to **TOM[i]_CH[x]_CM1.CM1** which may be multiple times during one period. Therefore, from application point of view it is not useful to enable this interrupt.

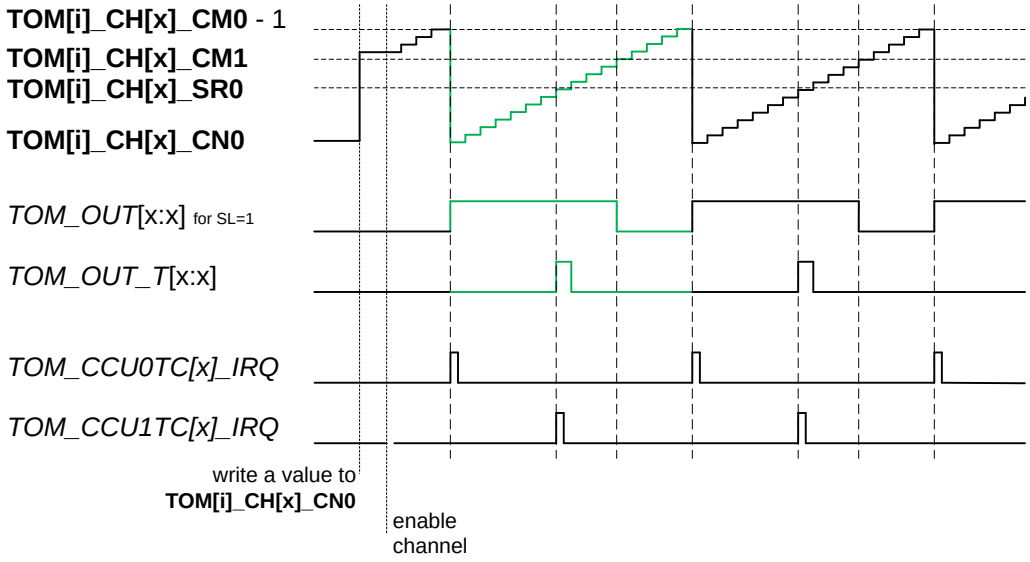
14.9 Trigger Generation

For applications with constant PWM period defined by **TOM[i]_CH[x]_CM0.CM0** , it is not necessary to update the **TOM[i]_CH[x]_CM0.CM0** register regularly with **TOM[i]_CH[x]_SR0.SR0** register. For these applications, the **TOM[i]_CH[x]_SR0.SR0** register can be used to define an additional output signal and interrupt trigger event.
If bit **TOM[i]_CH[x]_CTRL.SR0_TRIG** is set to 1, the register **TOM[i]_CH[x]_SR0.SR0** is no longer used as a shadow register for register **TOM[i]_CH[x]_CM0.CM0** . Instead, **TOM[i]_CH[x]_SR0.SR0** is compared with **TOM[i]_CH[x]_CN0.CN0** . If both are equal, a pulse of signal level '1' is generated at the output *TOM_OUT_T* [x:x] of instance i channel x. In this case, **TOM[i]_CH[x]_CM1.CM1** will still be synchronously updated from its shadow register **TOM[i]_CH[x]_SR1.SR1** if the update is enabled by setting **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** =0b10.
The bit **TOM[i]_CH[x]_CTRL.SR0_TRIG** should only be set if bit **TOM[i]_CH[x]_CTRL.RST_CCU0** of this channel is 0.

If bit **TOM[i]_CH[x]_CTRL.SR0_TRIG** =1 is set, the interrupt on *TOM_CCU1TC[x]_IRQ* is set at the same time with pulse generation on *TOM_OUT_T* in case of a compare equal match of **TOM[i]_CH[x]_SR0.SR0** = **TOM[i]_CH[x]_CN0.CN0** . If bit **TOM[i]_CH[x]_CTRL.SR0_TRIG** is set the interrupt notify flag **TOM[i]_CH[x]_IRQ_NOTIFY.CCU1TC** is no longer set on a compare match of **TOM[i]_CH[x]_CM1.CM1** and **TOM[i]_CH[x]_CN0.CN0** . Instead, the **TOM[i]_CH[x]_IRQ_NOTIFY.CCU1TC** interrupt notify flag is set in case of a compare equal match of **TOM[i]_CH[x]_SR0.SR0** and **TOM[i]_CH[x]_CN0.CN0** .

With configuration bit **TOM[i]_CH[x]_CTRL.TRIG_PULSE** one can select if the output *TOM_OUT_T* [x:x] is high as long as **TOM[i]_CH[x]_CN0.CN0** = **TOM[i]_CH[x]_SR0.SR0** (**TOM[i]_CH[x]_CTRL.TRIG_PULSE** =0) or if there will be only one pulse of length one cluster clock period when **TOM[i]_CH[x]_CN0.CN0** becomes SR0 (**TOM[i]_CH[x]_CTRL.TRIG_PULSE** =1). Figure *73 "Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.TRIG_PULSE=1 in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.SR0_TRIG=1"* shows the case of **TOM[i]_CH[x]_CTRL.TRIG_PULSE** =1.

Figure 73 Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.TRIG_PULSE=1 in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.SR0_TRIG=1



TOM/SPE interface must not be used when **TOM[i]_CH[x]_CTRL.SR0_TRIG** is set to 1.

The TOM output signal routing to DTM or GTM-IP top level is described in chapter *16.9 "DTM Connections on GTM-IP Top Level"*

14.10 TOM BLDC Support

The TOM sub-module offers a BLDC support in combination with the SPE sub-module. TOM channels 0 to 7 can be used to drive a BLDC engine.

The BLDC support can be configured by setting the bit **TOM[i]_CH[x]_CTRL.SPEM** = 1. If this bit is set, the TOM channel output will be controlled through the *TOM[i]_SPE_OUT* [x:x] signal coming from *SPE[i]_OUT* [x:x] of the SPE sub-module (see figure *60 "TOM Channel Architecture"*). Please refer to chapter *23 "Sensor Pattern Evaluation (SPE)"* for a detailed description of the SPE sub-module.

The TOM[i] channel [x] (x∈{2, 6, 7, 8, 9}) can be used together with the SPE module to trigger a delayed update of **SPE[i]_OUT_CTRL** register (i.e. commutation delay) after new input pattern detected by SPE (signaled by *SPE[i]_NIPD*). This feature is configured on TOM[i] channel[x] (x∈{2, 6, 7, 8, 9}) by setting **TOM[i]_CH[x]_CTRL.SPE_TRIG** =1, **TOM[i]_CH[x]_CTRL.SPEM** =0 or **TOM[i]_CH[x]_CTRL.SPE_TRIG** =0, **TOM[i]_CH[x]_CTRL.SPEM** =1 with **TOM[i]_CH[x]_CTRL.OSM** =1. With this configuration the TOM[i] channel [x] (x∈{2, 6, 7, 8, 9}) generates one single PWM pulse when receiving trigger signal *TOM[i]_SPE_NIPD* . Then the update of **SPE[i]_OUT_CTRL** register can be triggered by *TOM[i]_CH[x]_TRIG_CCU1* that are generated by compare event in CCU1 unit. That means the update of **SPE[i]_OUT_CTRL** can be synchronized to the second edge of the one shot PWM signal.
For details please refer to chapter of SPE sub-module description.

Normally the trigger signals *TOM[i]_CH[x]_TRIG_CCU0* / *TOM[i]_CH[x]_TRIG_CCU1* to SPE sub-module are single cycle pulses with the length of one cluster clock cycle. Depending on the configuration, it may happen that the trigger signal *TOM[i]_CH[x]_TRIG_CCU0* / *TOM[i]_CH[x]_TRIG_CCU1* to SPE sub-module is high for two system clock cycles once at the starting condition.

14.11 TOM Gated Counter Mode

Each TOM SPE module combination provides also the feature of a gated counter mode. This is reached by using the *FSOI[i]* input of the SPE module i (see figure *157 "SPE Submodule Integration Concept into GTM-IP"*) to gate the clock of a CCU0 sub-module.
To configure this mode, register of module SPE should be set as following:

- The SPE should be enabled (bit **SPE[i]_CTRL_STAT.EN** = 1)
- All three TIM inputs should be disabled (**SPE[i]_CTRL_STAT.SIE[0]** = **SPE[i]_CTRL_STAT.SIE[1]** = **SPE[i]_CTRL_STAT.SIE[2]** = 0) according to *162 "SPE[i]_CTRL_STAT Register Representation"* ,
- **SPE[i]_OUT_CTRL.SPE_OUT_CTRL[x]** should be set to 0b10 (set *TOM[i]_SPE_OUT* [x:x] to '0'),
- Mode FSOM should be enabled (**SPE[i]_CTRL_STAT.FSOM** =1), **SPE[i]_CTRL_STAT.FSOL** [x:x] if channel x of module TOM i is chosen for gated counter mode

Additionally in module TOM:

- The SPE mode should be disabled (**TOM[i]_CH[x]_CTRL.SPEM** =0)
- The gated counter mode should be enabled (**TOM[i]_CH[x]_CTRL.GCM** =1)

As a result of this configuration, the counter **TOM[i]_CH[x]_CN0.CN0** in sub-module CCU0 of TOM channel x counts as long as input *FSOI[i]* of the connected SPE module is '0'.

14.12 High Resolution PWM Support (HRES Mode)

In HRES mode the TOM sub-module offers the output signal *HRES_OUT* [x:x] in addition to the PWM output signal *TOM_OUT* [x:x] to control a circuit, e.g. a delay chain, which is needed to realize the high-resolution PWM and which has to be implemented outside the GTM-IP. Please refer to chapter *3.8 "High Resolution PWM Support"* for additional information on the high resolution PWM support of the GTM-IP.

High-resolution mode in TOM sub-module is only available in SOMP continuous counting up mode (**TOM[i]_CH[x]_CTRL.UDMODE** =0, **TOM[i]_CH[x]_CTRL.SR0_TRIG** =0). HRES mode is not supported in continuous counting up-down mode or when **TOM[i]_CH[x]_CTRL.SR0_TRIG** =1.

The high-resolution PWM support is not available in one-shot mode. If one-shot mode is enabled by setting of **TOM[i]_CH[x]_CTRL.OSM** to 1, the high-resolution PWM support has to be disabled by setting of **TOM[i]_CH[x]_CTRL2.HRES** to 0, because it is not supported.

The high-resolution PWM support is also not available in PCM mode (**TOM[i]_CH[x]_CTRL.BITREV** =1).

The output signal *HRES_OUT* [x:x] will be set at each edge of the PWM output signal *TOM_OUT* [x:x].

This output signal *TOM_OUT_HRES[x]* has a width of five bit while the decimal value of these five bits defines the number of micro-steps by which the output signal *TOM_OUT* [x:x] will be delayed in a suitable circuit (e.g. delay chain) outside the GTM-IP.

The high-resolution support will be enabled by setting bit **TOM[i]_CH[x]_CTRL2.HRES** = 1.

If HRES mode is switched off by setting the configuration bit **TOM[i]_CH[x]_CTRL2.HRES** to zero, then the output signals *TOM_OUT_HRES[x]* and *TOM_OUT_T_HRES[x]* and of instance i are clamped to zero.

The value for the PWM period, stored in the registers **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_SR0.SR0** , and pulse width, stored in the registers **TOM[i]_CH[x]_CM1.CM1** and **TOM[i]_CH[x]_SR1.SR1** , represents the length of period and pulse width in number of cluster clock steps and micro-steps.

Hereby the number of cluster clock steps for the PWM output signal *TOM_OUT* [x:x] are defined by the upper bits [15:5], whereas the lower 5 bits [4:0] represent the additional micro steps, by which the output signal has to be delayed outside the GTM-IP. This is shown in figure *74 "Period and Duty-cycle Register in HRES Mode"* .

Figure 74 Period and Duty-cycle Register in HRES Mode

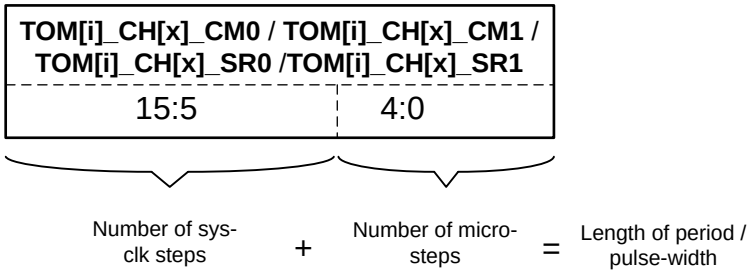
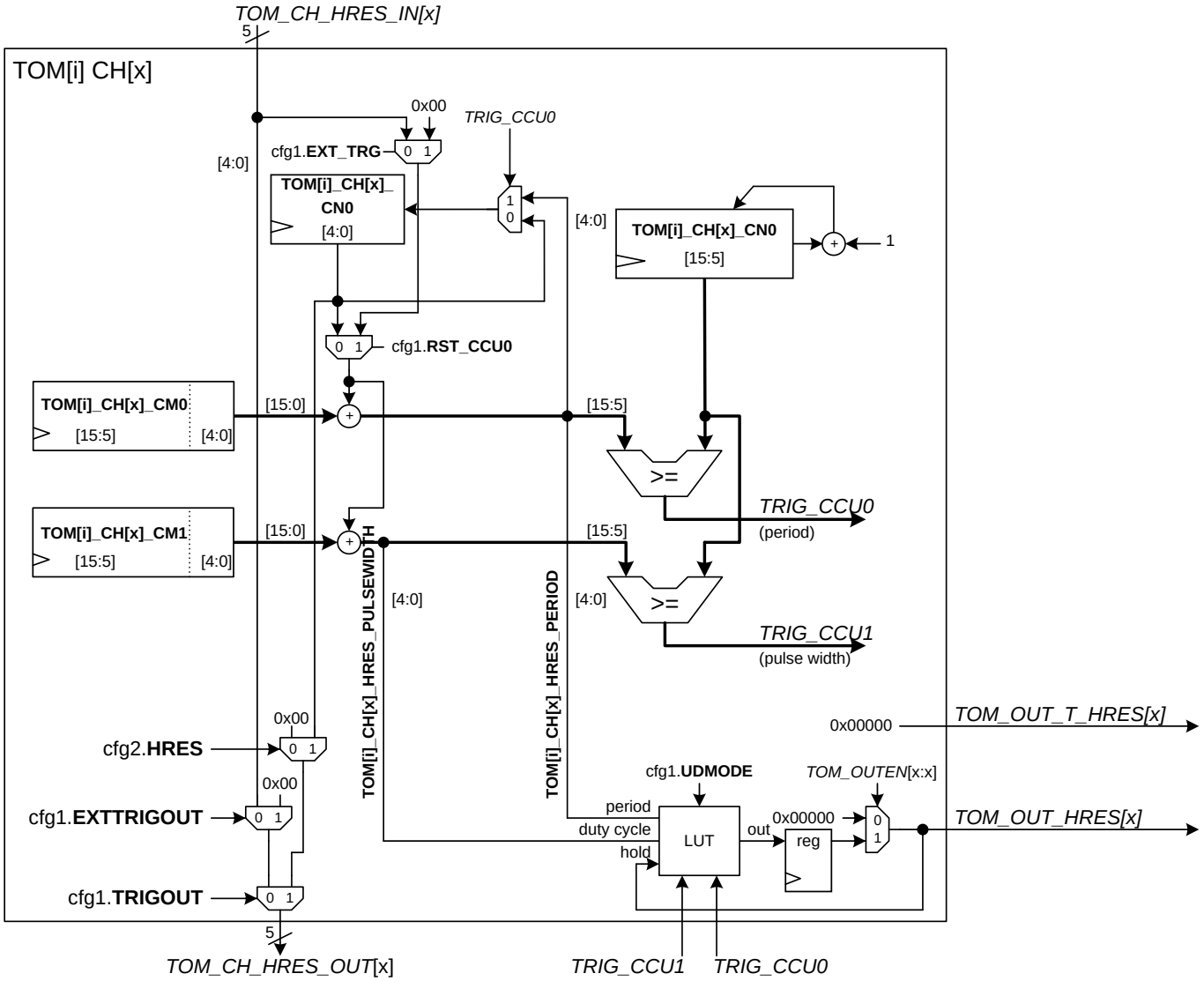


Figure *75 "High-level Schematic View of TOM_HRES_OUT[x:x] Calculation"* shows the high-level schematic of the generation of the output signal *TOM_OUT_HRES[x]* , which defines the number of micro-steps mentioned above. If the TOM channel x has HRES support with the configuration of x<TOM_HIGH_RES[i], a HRES trigger chain of *TOM_CH_HRES_IN[x]* / *TOM_CH_HRES_OUT[x]* is available together with the trigger chain of *TOM_CH_TRIGIN[x:x]* / *TOM_CH_TRIGOUT[x:x]* . As shown in figure *57 "TOM Block Diagram"* , the HRES trigger chain signals are routed through all TOM channels and instances that have HRES support. TOM0 channel 0 is driven by the output of TOM0 channel 15 with a register in the trigger chain, which means a delay of one cluster clock period. The pipeline registers in the trigger chain of *TOM_CH_TRIGIN[x:x]* / *TOM_CH_TRIGOUT[x:x]* as mentioned in chapter *14.3.2 "Internal Trigger Interface"* are also inserted in the trigger chain of *TOM_CH_HRES_IN[x]* / *TOM_CH_HRES_OUT[x]* .

The channels x without HRES support (x>=TOM_HRES_RES[i]) still have the HRES trigger chain of *TOM_CH_HRES_IN[x]* / *TOM_CH_HRES_OUT[x]* across these channels. The *TOM_CH_HRES_IN[x]* coming from *TOM_CH_HRES_OUT[x-1]* is directly connected to *TOM_CH_HRES_OUT[x]* that is connected to *TOM_CH_HRES_IN[x+1]* . There is no specific HRES functionalities in these channels without HRES support. Both HRES output signals *TOM_OUT_HRES[x]* and *TOM_OUT_T_HRES[x]* of these channels (x>=TOM_HIGH_RES[i]) are clamped to zero.

When **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** -1, *TRIG_CCU0* =1 will be set and then *TOM_OUT* [x:x] will be set to **TOM[i]_CH[x]_CTRL.SL** in the next CMU period. Similarly, when **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1** -1, *TRIG_CCU1* =1 will be set and then *TOM_OUT* [x:x] will be set to ! **TOM[i]_CH[x]_CTRL.SL** in the next CMU period. When **TOM[i]_CH[x]_CN0.CN0** < **TOM[i]_CH[x]_CM0.CM0** -1, *TRIG_CCU0* will be reset as 0. Similarly, when **TOM[i]_CH[x]_CN0.CN0** < **TOM[i]_CH[x]_CM1.CM1** -1, *TRIG_CCU1* will be reset to 0. According to the unit LUT in Figure *75 "High-level Schematic View of TOM_HRES_OUT[x:x] Calculation"* , the corresponding output signal *TOM_OUT_HRES[x]* is generated together with each edge of PWM output signal *TOM_OUT* [x:x].

Figure 75 High-level Schematic View of TOM_HRES_OUT[x:x] Calculation



cfg1: **TOM[i]_CH[x]_CTRL**
cfg2: **TOM[i]_CH[x]_CTRL2**

LUT

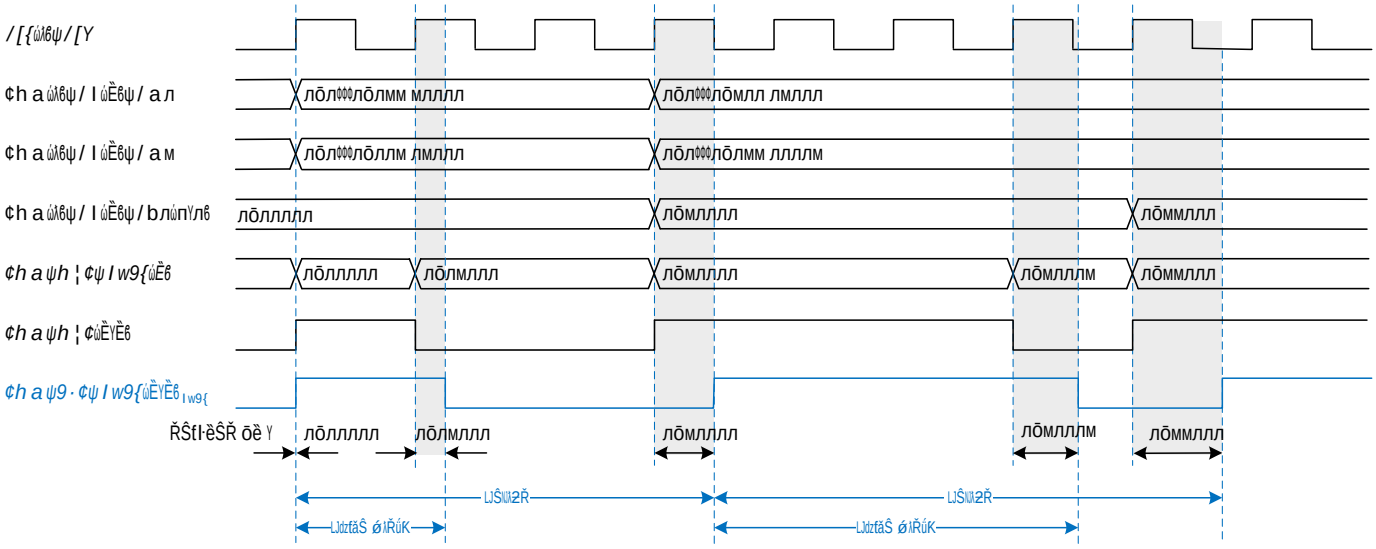
UDMODE	TRIG_CCU0	TRIG_CCU1	out
0	0	0	hold
0	0	1	**TOM[i]_CH[x]_HRES_PULSEWIDTH**
0	1	-	**TOM[i]_CH[x]_HRES_PERIOD**

As shown in figure *75 "High-level Schematic View of TOM_HRES_OUT[x:x] Calculation"* , when it is configured as continuous counting up mode (**TOM[i]_CH[x]_CTRL.UDMODE** =0) and **TOM[i]_CH[x]_CTRL.RST_CCU0** =0, the output signal *TOM_OUT_HRES[x]* has to be calculated based on the internal signal *TOM[i]_CH[x]_HRES_PERIOD* and *TOM[i]_CH[x]_HRES_PULSEWIDTH* in the following manner (here, t means the index of a PWM period):

- Period: *TOM[i]_CH[x]_HRES_PERIOD* = **TOM[i]_CH[x]_CN0.CN0** [4:0] + **TOM[i]_CH[x]_CM0.CM0** [4:0]
- Pulse width: TOM[i]_CH[x]_HRES_PULSEWIDTH = **TOM[i]_CH[x]_CN0.CN0** [4:0] + **TOM[i]_CH[x]_CM1.CM1** [4:0]
- With : (**TOM[i]_CH[x]_CN0.CN0** [4:0]) $_{t+1}$ = (**TOM[i]_CH[x]_CN0.CN0** [4:0]) $_{t}$ + (**TOM[i]_CH[x]_CM0.CM0** [4:0]) $_{t}$

The signal diagram in figure *76 "Signals in HRES Mode when TOM[i]_CH[x]_CTRL.RST_CCU0=0 and TOM[i]_CH[x]_CTRL.SL=1"* shows an example for the output generation in HRES mode when **TOM[i]_CH[x]_CTRL.RST_CCU0** =0 and **TOM[i]_CH[x]_CTRL.SL** =1. The PWM output signal *TOM_OUT* [x:x] with cluster clock steps is generated based on the upper bits [15:5] of **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM1.CM1** according to the same way of continuous counting up mode. Please refer to figure *64 "PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=0"* for more details. The output signal *TOM_OUT_HRES[x]* can be calculated based on the lower bits [4:0] of **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM1.CM1** according to figure *75 "High-level Schematic View of TOM_HRES_OUT[x:x] Calculation"* and the formulas above. As shown in figure *76 "Signals in HRES Mode when TOM[i]_CH[x]_CTRL.RST_CCU0=0 and TOM[i]_CH[x]_CTRL.SL=1"* , the value of *TOM_OUT_HRES[x]* represents the number of micro-steps that should be taken by the external circuit to delay each edge of *TOM_OUT* [x:x]. In this way, the delayed PWM signal *TOM_EXT_HRES* [x:x] achieves higher resolution.

Figure 76 Signals in HRES Mode when TOM[i]_CH[x]_CTRL.RST_CCU0=0 and TOM[i]_CH[x]_CTRL.SL=1



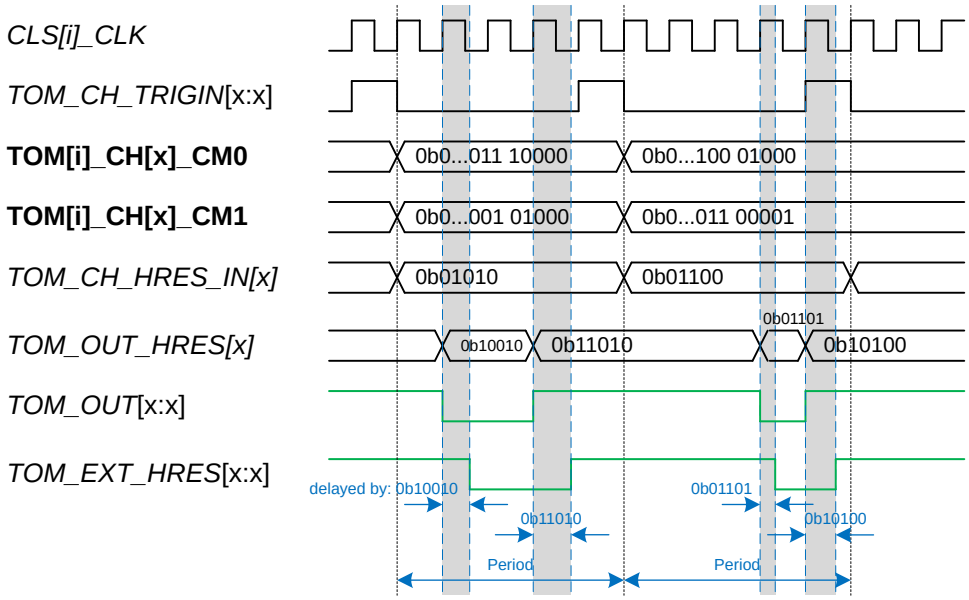
The HRES mode is also available in continuous counting up mode when **TOM[i]_CH[x]_CTRL.RST_CCU0** =1.

In this case (**TOM[i]_CH[x]_CTRL.RST_CCU0** = 1), if **TOM[i]_CH[x]_CTRL.EXT_TRIG** =0, the internal trigger signal *HRES_OUT* [x-1:x-1] from the previous channel is used for calculation instead of **TOM[i]_CH[x]_CN0.CN0** [4:0] :

- Period: *TOM[i]_CH[x]_HRES_PERIOD* = *HRES_IN* [x:x] + **TOM[i]_CH[x]_CM0.CM0** [4:0]
- Pulse width: *TOM[i]_CH[x]_HRES_PULSEWIDTH* = *HRES_IN* [x:x] + **TOM[i]_CH[x]_CM1.CM1** [4:0]

The signal diagram in figure *77 "Signals in HRES Mode when TOM[i]_CH[x]_CTRL.RST_CCU0=1, TOM[i]_CH[x]_CTRL.EXT_TRIG=0 and TOM[i]_CH[x]_CTRL.SL=1"* shows an example for the output generation in HRES mode when **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, **TOM[i]_CH[x]_CTRL.EXT_TRIG** =0 and **TOM[i]_CH[x]_CTRL.SL** =1. The PWM output signal *TOM_OUT* [x:x] with cluster clock steps is generated based on the upper bits [15:5] of **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM1.CM1** according to the same way of continuous counting up mode. Please refer to figure *65 "PWM Output Behavior with Respect to the TOM[i]_CH[x]_CTRL.SL Bit in Continuous Counting Up Mode if TOM[i]_CH[x]_CTRL.RST_CCU0=1"* for more details. The output signal *TOM_OUT_HRES[x]* representing the number of micro-steps can be calculated based on the lower bits [4:0] of **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** and *TOM_CH_HRES_IN[x]* according to figure *75 "High-level Schematic View of TOM_HRES_OUT[x:x] Calculation"* and the formulas above. Then the external circuit can take the value of *TOM_OUT_HRES[x]* to generate the delayed signal *TOM_EXT_HRES* [x:x].

Figure 77 Signals in HRES Mode when TOM[i]_CH[x]_CTRL.RST_CCU0=1, TOM[i]_CH[x]_CTRL.-EXT_TRIG=0 and TOM[i]_CH[x]_CTRL.SL=1

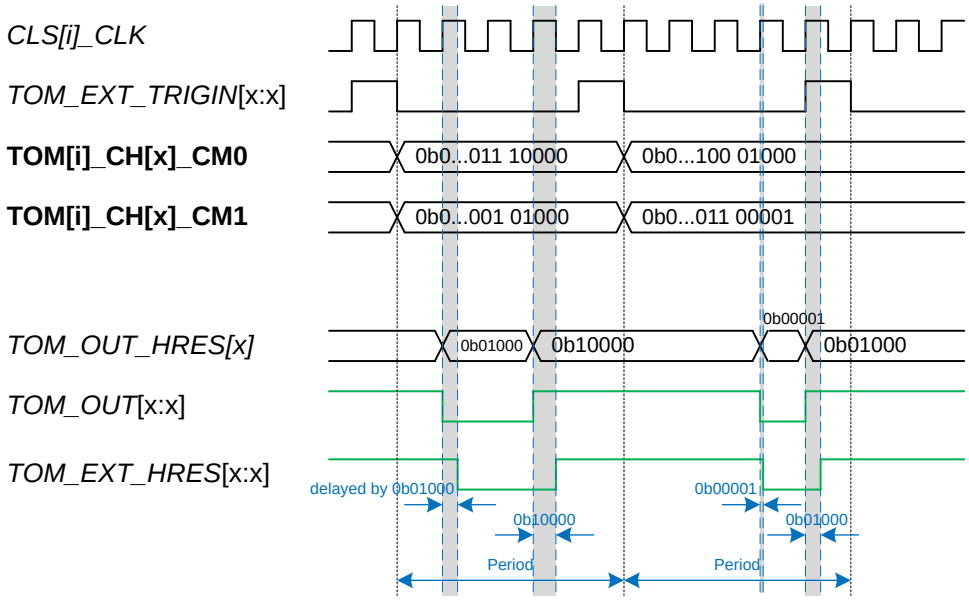


If **TOM[i]_CH[x]_CTRL.EXT_TRIG** =1, the HRES output signal is only depending on **TOM[i]_CH[x]_CM0.CM0** [4:0] and **TOM[i]_CH[x]_CM1.CM1** [4:0]:

- Period: *TOM[i]_CH[x]_HRES_PERIOD* = 0b00 + **TOM[i]_CH[x]_CM0.CM0** [4:0]
- Pulse width: *TOM[i]_CH[x]_HRES_PULSEWIDTH* = 0b00 + **TOM[i]_CH[x]_CM1.CM1** [4:0]

The signal diagram in figure *78 "Signals in HRES Mode when TOM[i]_CH[x]_CTRL.RST_CCU0=1, TOM[i]_CH[x]_CTRL.EXT_TRIG=1 and TOM[i]_CH[x]_CTRL.SL=1"* shows an example for the output generation in HRES mode when **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, **ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG** =1 and **TOM[i]_CH[x]_CTRL.SL** =1. Similarly, *TOM_OUT* [x:x] is normal continuous counting up mode PWM signal with cluster clock steps, while *TOM_OUT_HRES[x]* represents the number of micro-steps for generating *TOM_EXT_HRES* [x:x].

Figure 78 Signals in HRES Mode when TOM[i]_CH[x]_CTRL.RST_CCU0=1, TOM[i]_CH[x]_CTRL.EXT_TRIG=1 and TOM[i]_CH[x]_CTRL.SL=1



The following restrictions and characteristics must be considered, if the HRES mode is enabled:

- The selected clock resolution source **TOM[i]_CH[x]_CTRL.CLK_SRC** of *CCM[i]_FXCLK_RES* [x:x] has to be configured to the cluster clock *CLS[i]_CLK* .
- The maximum possible period is reduced to 2^11 cluster clock cycles.
- Only one edge on *TOM_OUT* [x:x] with one value of *TOM_OUT_HRES[x]* per cluster clock cycle is possible.

In order to fulfill the constraint above, there are more constraints when **TOM[i]_CH[x]_CTRL.RST_CCU0** =0:

- For normal PWM signal, the constraint **TOM[i]_CH[x]_CM0.CM0** >= 64 and 32<= **TOM[i]_CH[x]_CM1.CM1** <= **TOM[i]_CH[x]_CM0.CM0** – 32 must be fulfilled.
- For 0% duty cycle PWM signal (keep ! **TOM[i]_CH[x]_CTRL.SL**), the constraint **TOM[i]_CH[x]_CM0.CM0** >= 64 and **TOM[i]_CH[x]_CM1.CM1** = 0 must be fulfilled.
- For 100% duty cycle PWM signal (keep **TOM[i]_CH[x]_CTRL.SL**), the constraint **TOM[i]_CH[x]_CM0.CM0** = **TOM[i]_CH[x]_CM1.CM1** >= 64 must be fulfilled.

However, when **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, there are other constraints and some of them are different from the case when HRES option is disabled:

- When configuring **TOM[i]_CH[x]_CM0.CM0** [15:5]=0 or **TOM[i]_CH[x]_CM1.CM1** [15:5]=0, the lower HRES bits **TOM[i]_CH[x]_CM0.CM0** [4:0]=0 or **TOM[i]_CH[x]_CM1.CM1** [4:0] must also be 0.
- It is allowed to configure the registers as **TOM[i]_CH[x]_CM1.CM1** <= **TOM[i]_CH[x]_CM0.CM0** – 32 or **TOM[i]_CH[x]_CM1.CM1** = **TOM[i]_CH[x]_CM0.CM0** .
- The configuration **TOM[i]_CH[x]_CM1.CM1** > **TOM[i]_CH[x]_CM0.CM0** is not allowed.

Furthermore, if **TOM[i]_CH[x]_CTRL.RST_CCU0** =1 and **TOM[i]_CH[x]_CTRL.EXT_TRIG** =0, the channel is triggered by the preceding channel with HRES output trigger chain. Besides the constraints above, there are more constraints between the triggered channel and the triggering channel. Here, the registers of the triggered channel are noted with a subscript "triggered" while the triggering of the preceding channel is marked with "master".

- The configuration must fulfill (**TOM[i]_CH[x]_CM0.CM0**)$_{triggered}$ <= (**TOM[i]_CH[x]_CM0.CM0**)$_{master}$ or (**TOM[i]_CH[x]_CM0.CM0** [15:5])$_{triggered}$ > (**TOM[i]_CH[x]_CM0.CM0** [15:5])$_{master}$.
- If (**TOM[i]_CH[x]_CM0.CM0** [15:5])$_{triggered}$ = (**TOM[i]_CH[x]_CM0.CM0** [15:5])$_{master}$, (**TOM[i]_CH[x]_CM1.CM1**)$_{triggered}$ must be greater or equal to 32.

When HRES mode is enabled, *TOM_CH_HRES_IN[x]* / *TOM_CH_HRES_OUT[x]* are always generated synchronously together with *TOM_CH_TRIGIN* [x:x]/ *TOM_CH_TRIGOUT* [x:x] and may be delayed by same pipeline registers according to device configuration parameter configuration as defined in *14.3.2 "Internal Trigger Interface"* . If one channel triggers several following several channels, the 4 trigger signals *TOM_CH_TRIGIN* [x:x]/ *TOM_CH_TRIGOUT* [x:x] with *TOM_CH_HRES_IN[x]* / *TOM_CH_HRES_OUT[x]* build up a trigger chain from the triggering channel to the triggered channels. In this case, if there is no pipeline register in the trigger chain, these triggered channels can be configured with same operation registers and synchronously triggered to generate consistent PWM signals. However, if there are pipeline registers that may introduce delay of number of cluster clock periods, it is hard to simultaneously trigger several channels to generate consistent PWM signals. The reason for that is in HRES mode the operation clock frequency is the cluster clock and each pipeline register may cause a delay or shift of one cluster clock on the trigger signal pulse and this delay must be taken into account.

14.13 TOM FREEZE Mode

It is possible, to disable the TOM channel (*TOM_ENDIS* =0) without changing internal and output register states by setting of **TOM[i]_CH[x]_CTRL.FREEZE** to 1. After re-enabling again (*TOM_ENDIS* =1), the channel is started from the state from which it was deactivated (*TOM_ENDIS* =0).

The following sequence has to be realized to use the FREEZE mode in a reliable manner:

1. TOM channel setup and enable by setting *TOM_ENDIS* =1
2. Setting of **TOM[i]_CH[x]_CTRL.FREEZE** to 1
3. TOM channel disable by setting *TOM_ENDIS* =0
4. TOM channel re-enable by setting *TOM_ENDIS* =1
5. Setting of **TOM[i]_CH[x]_CTRL.FREEZE** to 0

Note:
TOM[i]_CH[x]_CTRL.FREEZE must always be set, while the channel is enabled (*TOM_ENDIS* =1).

14.14 TOM Software Reset of Channels

Each channel x (x=g*2+c) is assigned a unique bit c in the configuration register **TOM[i]_TGC[g]_GLB_CTRL** for software reset of the TOM[i] channel x.

Writing the value 0b1 to the bit field **TOM[i]_TGC[g]_GLB_CTRL.RST_CH[c]** via the configuration interface will immediately reset the content of the following channel x registers to the initial hardware reset state. x = g*2+c

- **TOM[i]_CH[x]_CTRL**
- **TOM[i]_CH[x]_CTRL_SR**
- **TOM[i]_CH[x]_CTRL2**
- **TOM[i]_CH[x]_CN0**
- **TOM[i]_CH[x]_CM0**
- **TOM[i]_CH[x]_CM1**
- **TOM[i]_CH[x]_SR0**
- **TOM[i]_CH[x]_SR1**
- **TOM[i]_CH[x]_STAT**
- **TOM[i]_CH[x]_IRQ_NOTIFY**
- **TOM[i]_CH[x]_IRQ_EN**
- **TOM[i]_CH[x]_IRQ_MODE**
- Internal signals inside the TOM[i] channel [x]:

Atomic reset of multiple channels is possible by writing a 1 on each associated bit. E.g: Writing 0x0F to register **TOM[0]_TGC[0]_GLB_CTRL** will reset the TOM0 channel .

14.15 TOM Configuration Registers Description

14.15.1 TOM[i]_TGC[g]_GLB_CTRL

Description	TOM[i] TGC [g] global control register
Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}$
Condition	

Storage Type	REGISTER
Clock Active Cond	$\mathrm{TOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_TGC[g]_GLB_CTRL
Address	$0x20000 * i + 0x80 * g + 0x1430$
C-Name	GTM.CLS[i].TOM.TGC[g].GLB_CTRL

Interface: MCS[i]

Name	TOM[i]_TGC[g]_GLB_CTRL
Address	$0x80 * g + 0x1430$
C-Name	

HOST_TRIG	
Description	Trigger request signal to update the register TOM[i]_TGC[g]_ENDIS_STAT, TOM[i]_TGC[g]_OUTEN_STAT
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0 : No trigger request 1 : Set trigger request
	Note: This flag is reset automatically after triggering the update.

RST_CH[c]	
Description	Software reset of channel [x]; x = c + g*8
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c+8 : c+8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-

RST_CH[c]	
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0 : No action 1 : Reset channel
	Note: This bit is cleared automatically after write over the configuration interface. The channel registers are set to their reset values and channel operation is stopped immediately. The internal SR register SOUR is set to '1'. **Note:** The write access to **TOM[i]_TGC[g]_GLB_CTRL.RST_CH[c]** bit will take two internal bus clock cycles.

UPEN_CTRL[c]	
Description	TOM[i] channel [x] (x=c + g*8) enable update of register TOM[i]_CH[x]_CM0, TOM[i]_CH[x]_CM1, TOM[i]_CH[x]_CTRL.SL and TOM[i]_CH[x]_CTRL.CLK_SRC from TOM[i]_CH[x]_SR0, TOM[i]_CH[x]_SR1, TOM[i]_CH[x]_CTRL_SL.SL_SR and TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR.
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * c + 17 : 2 * c + 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Don't care, bits **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** will not be changed 0b01 : Disable update 0b10 : Enable update 0b11 : Don't care, bits **TOM[i]_TGC[g]_GLB_CTRL.UPEN_CTRL[c]** will not be changed
R-Coding	0b00 : Update disabled 0b01 : Unused 0b10 : Unused 0b11 : Update enabled

14.15.2 TOM[i]_TGC[g]_ENDIS_CTRL

Description	TOM[i] TGC [g] enable/disable control register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_TGC[g]_ENDIS_CTRL
Address	$0x20000 * i + 0x80 * g + 0x1470$

C–Name	GTM.CLS[i].TOM.TGC[g].ENDIS_CTRL

Interface: MCS[i]

Name	TOM[i]_TGC[g]_ENDIS_CTRL
Address	$0x80 * g + 0x1470$
C–Name	

ENDIS_CTRL[c]	
Description	TOM[i] channel [x] (x=c + g*8) enable/disable control register.
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * c + 1 : 2 * c]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W–Coding	0b00 : Bit field **TOM[i]_TGC[g]_ENDIS_STAT.ENDIS_STAT[c]** will not be changed on an update trigger 0b01 : Disable channel on an update trigger 0b10 : Enable channel on an update trigger 0b11 : Don't change bits of this register
R–Coding	0b00 : Bit field **TOM[i]_TGC[g]_ENDIS_STAT.ENDIS_STAT[c]** will not be changed on an update trigger 0b01 : Disable channel on an update trigger 0b10 : Enable channel on an update trigger

14.15.3 TOM[i]_TGC[g]_ENDIS_STAT

Description	TOM[i] TGC [g] enable/disable status register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_TGC[g]_ENDIS_STAT
Address	$0x20000 * i + 0x80 * g + 0x1474$
C–Name	GTM.CLS[i].TOM.TGC[g].ENDIS_STAT

Interface: MCS[i]

Name	TOM[i]_TGC[g]_ENDIS_STAT
Address	$0x80 * g + 0x1474$

C–Name	

ENDIS_STAT[c]	
Description	TOM[i] channel [x] (x=c + g*8) enable/disable status register
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * c + 1 : 2 * c]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W–Coding	0b00 : Don't care, bits **TOM[i]_TGC[g]_ENDIS_STAT.ENDIS_STAT[c]** will not be changed 0b01 : Disable channel 0b10 : Enable channel 0b11 : Don't care, bits **TOM[i]_TGC[g]_ENDIS_STAT.ENDIS_STAT[c]** will not be changed
R–Coding	0b00 : Channel disabled 0b01 : Unused 0b10 : Unused 0b11 : Channel enabled

14.15.4 TOM[i]_TGC[g]_ACT_TB

Description	TOM[i] TGC [g] action time base register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_TGC[g]_ACT_TB
Address	$0x20000 * i + 0x80 * g + 0x1434$
C–Name	GTM.CLS[i].TOM.TGC[g].ACT_TB

Interface: MCS[i]

Name	TOM[i]_TGC[g]_ACT_TB
Address	$0x80 * g + 0x1434$
C–Name	

ACT_TB	
Description	Action time base. When the selected TBU time base CCM[i]_TBU_TS0/CCM[i]_TBU_TS1/CCM[i]_TBU_TS2 is "cyclically greater than or equal to" the action time base TOM[i]_TGC[g]_ACT_TB.ACT_TB, the compare event specified in TOM[i]_TGC[g]_ACT_TB.ACT_TB is considered to have occurred in the past and so the trigger CTRL_TRIG is immediately generated in TGC[g] unit if TOM[i]_TGC[g]_ACT_TB.TB_TRIG is 1. Please refer to chapter GTM Architecture for more information about cyclic event compare strategy.
Loop	–
Bit Range	[23 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

TB_TRIG	
Description	Set trigger request
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : No trigger request 1 : Set trigger request
	Note: This flag is reset automatically if the selected time base unit (*CCM[i]_TBU_TS0* or *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* if present) has reached the value **TOM[i]_TGC[g]_ACT_TB.ACT_TB** and the update of the register was triggered.

TBU_SEL	
Description	Selection of time base used for comparison
Loop	–
Bit Range	[26 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

TBU_SEL	
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : *CCM[i]_TBU_TS0* selected 0b01 : *CCM[i]_TBU_TS1* selected 0b10 : *CCM[i]_TBU_TS2* selected 0b11 : Same as 0b00

14.15.5 TOM[i]_TGC[g]_OUTEN_CTRL

Description	TOM[i] TGC [g] output enable control register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_TGC[g]_OUTEN_CTRL
Address	$0x20000 * i + 0x80 * g + 0x1478$
C-Name	GTM.CLS[i].TOM.TGC[g].OUTEN_CTRL

Interface: MCS[i]

Name	TOM[i]_TGC[g]_OUTEN_CTRL
Address	$0x80 * g + 0x1478$
C-Name	

OUTEN_CTRL[c]	
Description	Output enable control of TOM [i] channel [x] output TOM_OUT[x:x], x=c+g*8
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * c + 1 : 2 * c]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Bit field **TOM[i]_TGC[g]_OUTEN_STAT.OUTEN_STAT[c]** will not be changed on an update trigger 0b01 : Disable channel output on an update trigger 0b10 : Enable channel output on an update trigger 0b11 : Don't change bits of this register

OUTEN_CTRL[c]	
R-Coding	0b00 : Bit field **TOM[i]_TGC[g]_OUTEN_STAT.OUTEN_STAT[c]** will not be changed on an update trigger 0b01 : Disable channel output on an update trigger 0b10 : Enable channel output on an update trigger

14.15.6 TOM[i]_TGC[g]_OUTEN_STAT

Description	TOM[i] TGC [g] output enable status register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_TGC[g]_OUTEN_STAT
Address	$0x20000 * i + 0x80 * g + 0x147C$
C-Name	GTM.CLS[i].TOM.TGC[g].OUTEN_STAT

Interface: MCS[i]

Name	TOM[i]_TGC[g]_OUTEN_STAT
Address	$0x80 * g + 0x147C$
C-Name	

OUTEN_STAT[c]	
Description	Output enable status of TOM [i] channel [x] output TOM_OUT[x:x], x=c+g*8
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * c + 1 : 2 * c]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Don't care, bits **TOM[i]_TGC[g]_OUTEN_STAT.OUTEN_STAT[c]** will not be changed 0b01 : Disable output 0b10 : Enable output 0b11 : Don't care, bits **TOM[i]_TGC[g]_OUTEN_STAT.OUTEN_STAT[c]** will not be changed
R-Coding	0b00 : Output disabled 0b01 : Unused 0b10 : Unused 0b11 : Output enabled

OUTEN_STAT[c]	
	Note: If the output is disabled (*TOM_OUTEN* [x:x]=0), the TOM[i] channel x output *TOM_OUT* [x:x] and *TOM_OUT_T* [x:x] is the inverse value of bit **TOM[i]_CH[x]_CTRL.SL** .

14.15.7 TOM[i]_TGC[g]_FUPD_CTRL

Description	TOM[i] TGC [g] force update control register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_TGC[g]_FUPD_CTRL
Address	$0x20000 * i + 0x80 * g + 0x1438$
C-Name	GTM.CLS[i].TOM.TGC[g].FUPD_CTRL

Interface: MCS[i]

Name	TOM[i]_TGC[g]_FUPD_CTRL
Address	$0x80 * g + 0x1438$
C-Name	

FUPD_CTRL[c]	
Description	Force update control of operation registers of TOM[i] channel [x] (x = c + g*8)
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * c + 1 : 2 * c]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Don't care, bits **TOM[i]_TGC[g]_FUPD_CTRL.FUPD_CTRL[c]** will not be changed 0b01 : Disable force update 0b10 : Enable force update 0b11 : Don't care, bits **TOM[i]_TGC[g]_FUPD_CTRL.FUPD_CTRL[c]** will not be changed
R-Coding	0b00 : Force update disabled 0b01 : Unused 0b10 : Unused 0b11 : Force update enabled

FUPD_CTRL[c]	
	If enabled, force update of register **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** triggered by **TOM[i]_TGC[g]_GLB_CTRL.HOST_TRIG** , **TOM[i]_TGC[g]_ACT_TB.ACT_TB** compare match or internal trigger. **Note:** The force update request is stored and executed synchronized to the selected clock source resolution defined in **TOM[i]_CH[x]_CTRL.CLK_SRC** .

RSTCN0_CH[c]	
Description	Reset TOM[i]_CH[x]_CN0 of channel [x] (x=c + g*8) with the force update event
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * c + 17 : 2 * c + 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W–Coding	0b00 : Don't care, bits **TOM[i]_TGC[g]_FUPD_CTRL.RSTCN0_CH[c]** will not be changed 0b01 : Do not reset **TOM[i]_CH[x]_CN0.CN0** on forced update 0b10 : Reset **TOM[i]_CH[x]_CN0.CN0** on forced update 0b11 : Don't care, bits **TOM[i]_TGC[g]_FUPD_CTRL.RSTCN0_CH[c]** will not be changed
R–Coding	0b00 : **TOM[i]_CH[x]_CN0.CN0** is not reset on forced update 0b01 : Unused 0b10 : Unused 0b11 : **TOM[i]_CH[x]_CN0.CN0** is reset on forced update
	If enabled, **TOM[i]_CH[x]_CN0.CN0** is reset together with a force update trigger event.

14.15.8 TOM[i]_TGC[g]_INT_TRIG

Description	TOM[i] TGC [g] internal trigger control register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_TGC[g]_INT_TRIG
Address	$0x20000 * i + 0x80 * g + 0x143C$
C–Name	GTM.CLS[i].TOM.TGC[g].INT_TRIG

Interface: MCS[i]

Name	TOM[i]_TGC[g]_INT_TRIG
Address	$0x80 * g + 0x143C$
C-Name	

INT_TRIG[c]	
Description	Select input signal TOM_CH_TRIGOUT[x:x] as a trigger source for TGC[g] (x=c + g*8)
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * c + 1 : 2 * c]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : Don't care, bits **TOM[i]_TGC[g]_INT_TRIG.INT_TRIG[c]** will not be changed 0b01 : Do not use internal trigger from channel x (*TOM_CH_TRIGOUT* [x:x]) 0b10 : Use internal trigger from channel x (*TOM_CH_TRIGOUT* [x:x]) 0b11 : Don't care, bits **TOM[i]_TGC[g]_INT_TRIG.INT_TRIG[c]** will not be changed
R-Coding	0b00 : Internal trigger from channel x (*TOM_CH_TRIGOUT* [x:x]) not used 0b01 : Unused 0b10 : Unused 0b11 : Internal trigger from channel x (*TOM_CH_TRIGOUT* [x:x]) used

14.15.9 TOM[i]_CH[x]_CTRL

Description	TOM[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_CTRL
Address	$0x20000 * i + 0x40 * x + 0x1000$
C-Name	GTM.CLS[i].TOM.CH[x].CTRL

Interface: MCS[i]

Name	TOM[i]_CH[x]_CTRL
Address	$0x40 * x + 0x1000$
C-Name	

SR0_TRIG	
Description	TOM[i]_CH[x]_SR0 is used to generate a trigger on output TOM_OUT_T[x:x] if equal to TOM[i]_CH[x]_CN0.
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	TOM[i]_CH[x]_CTRL.RST_CCU0 == 0 0 : **TOM[i]_CH[x]_SR0** is used as a shadow register for register **TOM[i]_CH[x]_CM0.CM0** . 1 : **TOM[i]_CH[x]_SR0** is not used as a shadow register for register **TOM[i]_CH[x]_CM0.CM0** . **TOM[i]_CH[x]_SR0** is compared with **TOM[i]_CH[x]_CN0** and if both are equal, a trigger pulse is generated at output *TOM_OUT_T* [x:x] .
RW-Coding	TOM[i]_CH[x]_CTRL.RST_CCU0 == 1 0 : Not applicable, initial value 1 : Prohibited
	Attention: This bit is only supported if **TOM[i]_CH[x]_CTRL.RST_CCU0** of this channel is 0.

SL	
Description	Signal level for pulse width
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	!TOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Low signal level 1 : High signal level

SL	
	Note: Reset value depends on the hardware configuration chosen by silicon vendor. **Note:** If the output is disabled, the output *TOM_OUT* [x:x] is set to inverse value of **TOM[i]_CH[x]_CTRL.SL** . **Note:** If **TOM[i]_CH[x]_CTRL.FREEZE** =0, when the channel is disabled (*TOM_ENDIS* [x:x]=0) and the output is enabled(*TOM_OUTEN* [x:x]=1), the output *TOM_OUT* [x:x] will be set to ! **TOM[i]_CH[x]_CTRL.SL** and in continuous counting up-down mode with **TOM[i]_CH[x]_CTRL.RST_CCU0** =1 the other output *TOM_OUT_T* [x:x] will be also set to ! **TOM[i]_CH[x]_CTRL.SL** . Thus, when the channel is re-enabled (*TOM_ENDIS* [x:x]=1 && *TOM_OUTEN* [x:x]=1) afterwards, the output *TOM_OUT* [x:x] is initially ! **TOM[i]_CH[x]_CTRL.SL** and in continuous counting up-down mode with **TOM[i]_CH[x]_CTRL.RST_CCU0** =1 the other output *TOM_OUT_T* [x:x] is also initially ! **TOM[i]_CH[x]_CTRL.SL** . In other PWM modes, *TOM_OUT_T* [x:x] is not supported. **Note:** If **TOM[i]_CH[x]_CTRL.FREEZE** =1, when the channel is disabled (*TOM_ENDIS* [x:x]=0) and the output is enabled(*TOM_OUTEN* [x:x]=1), the output *TOM_OUT* [x:x] will be not changed and in continuous counting up--down mode with **TOM[i]_CH[x]_CTRL.RST_CCU0** =1 the other output *TOM_OUT_T* [x:x] will also be not changed. Thus, when the channel is re-enabled (*TOM_ENDIS* [x:x]=1 && *TOM_OUTEN* [x:x]=1) afterwards, the output *TOM_OUT* [x:x] is initially the same state as it is before the channel is disabled and in continuous counting up-down mode with **TOM[i]_CH[x]_CTRL.RST_CCU0** =1 the other output *TOM_OUT_T* [x:x] also initially remains the same. In other PWM modes, *TOM_OUT_T* [x:x] is not supported.

CLK_SRC	
Description	Clock source select for channel
Loop	-
Bit Range	[15 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0b0000 : *CCM[i]_FXCLK_RES* [0:0] selected 0b0001 : *CCM[i]_FXCLK_RES* [1:1] selected 0b0010 : *CCM[i]_FXCLK_RES* [2:2] selected 0b0011 : *CCM[i]_FXCLK_RES* [3:3] selected 0b0100 : *CCM[i]_FXCLK_RES* [4:4] selected 0b0101 : Prohibited 0b0110 : Prohibited 0b0111 : Prohibited 0b1000 : Prohibited 0b1001 : Prohibited 0b1010 : Prohibited 0b1011 : Prohibited 0b1100 : Functional operation stopped, clock resolution tied to zero 0b1101 : *TOM_CH_TRIGOUT* [x-1:x-1] selected 0b1110 : *TOM_EXT_TRIGIN* [x:x] selected 0b1111 : Prohibited

CLK_SRC	
	The register **TOM[i]_CH[x]_CTRL.CLK_SRC** is updated with the value of **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** together with the update of register **TOM[i]_CH[x]_CM0.CM0** and **TOM[i]_CH[x]_CM1.CM1** . **Note:** The clock of the channel is stopped, if "0b1100" value is configured. Restarting a channel by writing a value for a valid clock selection again is possible.

TRIG_PULSE	
Description	Trigger output pulse length of one cluster clock period
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	TOM[i]_CH[x]_CTRL.SR0_TRIG == 1 0 : Output on *TOM_OUT_T* of instance i channel x is 1 as long as **TOM[i]_CH[x]_CN0.CN0** = **TOM[i]_CH[x]_SR0.SR0** 1 : Output on *TOM_OUT_T* of instance i channel x is 1 for only one cluster clock period for **TOM[i]_CH[x]_CN0.CN0** = **TOM[i]_CH[x]_SR0.SR0**
RW-Coding	TOM[i]_CH[x]_CTRL.SR0_TRIG == 0 0 : Not applicable, initial value 1 : Prohibited
	Attention: This bit may only be changed if bit **TOM[i]_CH[x]_CTRL.SR0_TRIG** of this channel is 1.

UDMODE	
Description	Up-down counter mode
Loop	–
Bit Range	[19 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync

UDMODE	
RW-Coding	0b00 : Up-down counter mode disabled: **TOM[i]_CH[x]_CN0.CN0** counts always up. 0b01 : Up/down counter mode enabled: **TOM[i]_CH[x]_CN0.CN0** counts up and down, **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** are updated if **TOM[i]_CH[x]_CN0.CN0** is 0 (i.e. the counting direction changes from down to up). 0b10 : Up/down counter mode enabled: **TOM[i]_CH[x]_CN0.CN0** counts up and down, **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** are updated if **TOM[i]_CH[x]_CN0.CN0** reaches **TOM[i]_CH[x]_CM0.CM0** -1 or the channel receives the trigger signal and the counting direction changes from up to down. 0b11 : Up/down counter mode enabled: **TOM[i]_CH[x]_CN0.CN0** counts up and down, **TOM[i]_CH[x]_CM0.CM0** , **TOM[i]_CH[x]_CM1.CM1** , **TOM[i]_CH[x]_CTRL.SL** and **TOM[i]_CH[x]_CTRL.CLK_SRC** are updated if **TOM[i]_CH[x]_CN0.CN0** is 0 or reaches **TOM[i]_CH[x]_CM0.CM0** -1 or the channel receives the trigger signal and the counting direction changes from up to down.

RST_CCU0	
Description	Reset source of CCU0
Loop	-
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Reset counter **TOM[i]_CH[x]_CN0.CN0** to 0 on matching comparison **TOM[i]_CH[x]_CM0.CM0** 1 : Reset counter **TOM[i]_CH[x]_CN0.CN0** to 0 on trigger *TRIG_IN* [x:x] or *TOM_EXT_TRIGIN* [x:x].
	Attention: This bit may only be set if bit **TOM[i]_CH[x]_CTRL.OSM** =0 (i.e. in continuous mode)

OSM_TRIG	
Description	One-shot pulse generation enabled by the selected trigger signal
Loop	-
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync

OSM_TRIG	
RW-Coding	0 : Signal *TOM_CH_TRIGIN* [x:x] or *TOM_EXT_TRIGIN* [x:x] cannot trigger start of single pulse generation 1 : Signal *TOM_CH_TRIGIN* [x:x] or *TOM_EXT_TRIGIN* [x:x] can trigger start of single pulse generation (only if bit **TOM[i]_CH[x]_CTRL.OSM** = 1)
	Attention: This bit may only be set if bit **TOM[i]_CH[x]_CTRL.OSM** =1 and bit **TOM[i]_CH[x]_CTRL.RST_CCU0** =0.

EXT_TRIG	
Description	Select TOM_EXT_TRIGIN[x:x] as trigger signal
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : If **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, signal *TOM_CH_TRIGIN* [x:x] is selected as trigger to reset **TOM[i]_CH[x]_CN0.CN0** or to start single pulse generation. 1 : If **TOM[i]_CH[x]_CTRL.RST_CCU0** =1, signal *TOM_EXT_TRIGIN* [x:x] is selected as trigger to reset **TOM[i]_CH[x]_CN0.CN0** or to start single pulse generation.

EXTTRIGOUT	
Description	Select TOM_EXT_TRIGIN[x:x] as potential output signal TOM_CH_TRIGOUT[x:x]
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Signal *TOM_CH_TRIGIN* [x:x] is selected as output on *TOM_CH_TRIGOUT* [x:x] (if **TOM[i]_CH[x]_CTRL.TRIGOUT** =0) 1 : Signal *TOM_EXT_TRIGIN* [x:x] after synchronization to the selected CMU clock resolution is selected as output on *TOM_CH_TRIGOUT* [x:x] (if **TOM[i]_CH[x]_CTRL.TRIGOUT** = 0)

TRIGOUT	
Description	Trigger output selection (output signal TOM_CH_TRIGOUT[x:x]) of module TOM channel [x]
Loop	–

TRIGOUT	
Bit Range	$[24 : 24]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : *TOM_CH_TRIGOUT* [x:x] is *TOM_CH_TRIGIN* [x:x] or *TOM_EXT_TRIGIN* [x:x]. 1 : *TOM_CH_TRIGOUT* [x:x] is *TRIG_CCU0*

SPE_TRIG	
Description	SPE trigger to reset TOM[i]_CH[x]_CN0
Loop	–
Bit Range	$[25 : 25]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$x < 10 \&\& i < \mathrm{NSPE}$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	(TOM[i]_CH[x]_CTRL.SPEM == 0) && ((x == 2) || (x == 6) || (x == 7) || (x == 8) || (x == 9)) 0 : **TOM[i]_CH[x]_CN0.CN0** reset is defined by configuration of bit **TOM[i]_CH[x]_CTRL.RST_CCU0** 1 : **TOM[i]_CH[x]_CN0.CN0** is reset by signal at the port *TOM[i]_SPE_NIPD* and depends on configuration bit **TOM[i]_CH[x]_CTRL.RST_CCU0**
RW-Coding	(TOM[i]_CH[x]_CTRL.SPEM == 1) && ((x == 2) || (x == 6) || (x == 7) || (x == 8) || (x == 9)) 0 : **TOM[i]_CH[x]_CN0.CN0** is reset by signal at the port *TOM[i]_SPE_NIPD* and depends on configuration bit **TOM[i]_CH[x]_CTRL.RST_CCU0** 1 : **TOM[i]_CH[x]_CN0.CN0** reset is defined by configuration of bit **TOM[i]_CH[x]_CTRL.RST_CCU0**
RW-Coding	(x == 0) || (x == 1) || (x == 3) || (x == 4) || (x == 5) 0 : Not applicable, initial value 1 : Prohibited

SPE_TRIG	
	For TOM[i] channel x (x∈{2, 6, 7, 8, 9}) this bit defines in combination with bit **TOM[i]_CH[x]_CTRL.SPEM** the source of output pin *TOM_OUT* [x:x] and if **TOM[i]_CH[x]_CN0.CN0** can be reset by TOM[i] input signal *TOM[i]_SPE_NIPD* . **Note:** If a configuration of **TOM[i]_CH[x]_CTRL.SPEM** = 0 and **TOM[i]_CH[x]_CTRL.SPE_TRIG** = 1 or **TOM[i]_CH[x]_CTRL.SPEM** = 1 and **TOM[i]_CH[x]_CTRL.SPE_TRIG** = 0 is chosen and **TOM[i]_CH[x]_CTRL.RST_CCU0** = 1, the **TOM[i]_CH[x]_CN0.CN0** is reset by a *TOM[i]_SPE_NIPD* event or a *TOM_CH_TRIGIN* [x:x] event if **TOM[i]_CH[x]_CTRL.EXT_TRIG** = 0 or a *TOM_EXT_TRIGIN* [x:x] event if **TOM[i]_CH[x]_CTRL.EXT_TRIG** = 1. **Note:** For TOM[i] channel 8 and 9 this bit defines only if **TOM[i]_CH[x]_CN0.CN0** reset is defined by input signal *TOM[i]_SPE_NIPD* or by configuration of **TOM[i]_CH[x]_CTRL.RST_CCU0** . The output *TOM_OUT* [x:x] is not affected. The configuration bit **TOM[i]_CH[x]_CTRL.SPEM** is not supported for these channels and thus assumed to be 0. **Note:** If a configuration of **TOM[i]_CH[x]_CTRL.SPEM** = 0 and **TOM[i]_CH[x]_CTRL.SPE_TRIG** = 1 or **TOM[i]_CH[x]_CTRL.SPEM** = 1 and **TOM[i]_CH[x]_CTRL.SPE_TRIG** = 0 is chosen (i.e. **TOM[i]_CH[x]_CN0.CN0** is reset by signal *TOM[i]_SPE_NIPD*), the one-shot mode in corresponding TOM[i] channel should also be enabled by setting bit **TOM[i]_CH[x]_CTRL.OSM** = 1 to generate one PWM pulse in case of trigger *TOM[i]_SPE_NIPD* . **Note:** In SPE module one of the trigger signals *TOM_CH[2]_TRIG_CCU1* , *TOM_CH[6]_TRIG_CCU1* , *TOM_CH[7]_TRIG_CCU1* , *TOM_CH[8]_TRIG_CCU1* , or *TOM_CH[9]_TRIG_CCU1* of TOM instance i can be used to trigger the update of register **SPE[i]_OUT_CTRL** .

OSM	
Description	One-shot mode. In this mode the counter TOM[i]_CH[x]_CN0 counts for only one period. The length of period is defined by TOM[i]_CH[x]_CM0. A write access to the register TOM[i]_CH[x]_CN0 triggers the start of counting.
Loop	-
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : One-shot mode disabled 1 : One-shot mode enabled

BITREV	
Description	Bit-reversing of output of counter TOM[i]_CH[x]_CN0.
Loop	-
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

BITREV	
Condition	$x == 15$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
	Note: This bit enables the PCM mode and is only available for TOM[i] channel 15.

SPEM	
Description	SPE output mode enable for channel.
Loop	–
Bit Range	$[28 : 28]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$x < 8 \&\& i < \mathrm{NSPE}$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW–Coding	0 : SPE output mode disabled: *TOM_OUT* [x:x] defined by TOM[i] channel [x] SOUR register 1 : SPE output mode enabled: *TOM_OUT* [x:x] is defined by *TOM[i]_SPE_OUT* [x:x]
	Note: The bit is only implemented for TOM[i] instances connected to a SPE module and only for TOM[i] channels (x∈{0, 1,..., 7}). In other channels, no matter which value is written to this bit, the reading value of the bit is always 0. **Note:** For TOM[i] channel x (x∈{2, 6, 7, 8, 9}) this bit defines in combination with bit **TOM[i]_CH[x]_CTRL.SPE_TRIG** the source of output pin *TOM_OUT* [x:x] and if **TOM[i]_CH[x]_CN0.CN0** can be reset by TOM[i] input signal *TOM[i]_SPE_NIPD* .

GCM	
Description	Gated Counter Mode enable
Loop	–
Bit Range	$[29 : 29]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$x < 8 \&\& i < \mathrm{NSPE}$

GCM	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Gated Counter Mode disabled 1 : Gated Counter Mode enabled
	Note: The Gated Counter Mode is only available for TOM[i] instances connected to a SPE module and only for channels x (x∈{0, 1,..., 7}).

FREEZE	
Description	TOM[i] Freeze Mode enable
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : A channel disable/enable may change internal register and output register 1 : A channel enable/disable does not change an internal or output register but stops counter **TOM[i]_CH[x]_CN0.CN0**

14.15.10 TOM[i]_CH[x]_CTRL_SR

Description	TOM[i] channel [x] control shadow register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_CTRL_SR
Address	$0x20000 * i + 0x40 * x + 0x1030$
C-Name	GTM.CLS[i].TOM.CH[x].CTRL_SR

Interface: MCS[i]

Name	TOM[i]_CH[x]_CTRL_SR

Address	$0x40 * x + 0x1030$
C-Name	

SL_SR	
Description	Shadow register for TOM[i]_CH[x]_CTRL.SL
Loop	–
Bit Range	$[11 : 11]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	!TOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Low signal level 1 : High signal level
	Note: Reset value depends on the hardware configuration chosen by silicon vendor.

CLK_SRC_SR	
Description	Shadow register for TOM[i]_CH[x]_CTRL.CLK_SRC
Loop	–
Bit Range	$[15 : 12]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
	Note: This register is a shadow register for **TOM[i]_CH[x]_CTRL.CLK_SRC** . Thus, if the selected clock resolution source for PWM generation should be changed during operation, the old selected clock resolution has to operate until the update of the TOM[i] channels register **TOM[i]_CH[x]_CTRL.CLK_SRC** by the **TOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** content is done either by an end of a period or a forced update.

14.15.11 TOM[i]_CH[x]_CTRL2

Description	TOM[i] channel [x] control register

Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	$x < \text{TOM\_HIGH\_RES}[i]$
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_CTRL2
Address	$0x20000 * i + 0x40 * x + 0x102C$
C-Name	GTM.CLS[i].TOM.CH[x].CTRL2

Interface: MCS[i]

Name	TOM[i]_CH[x]_CTRL2
Address	$0x40 * x + 0x102C$
C-Name	

HRES	
Description	TOM[i] high-resolution support
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	x < TOM_HIGH_RES[i] 0 : High resolution support disabled 1 : High resolution support enabled
RW-Coding	x >= TOM_HIGH_RES[i] 0 : Not applicable, initial value 1 : Prohibited

14.15.12 TOM[i]_CH[x]_CN0

Description	TOM[i] channel [x] CCU0 counter
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_CN0
Address	$0x20000 * i + 0x40 * x + 0x1014$
C-Name	GTM.CLS[i].TOM.CH[x].CN0

Interface: MCS[i]

Name	TOM[i]_CH[x]_CN0
Address	$0x40 * x + 0x1014$
C-Name	

CN0	
Description	TOM[i] CCU0 counter
Loop	-
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
	This counter is stopped if the TOM[i] channel is disabled and not reset on an enable event of TOM[i] channel.

14.15.13 TOM[i]_CH[x]_CM0

Description	TOM[i] channel [x] CCU0 compare register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_CM0
Address	$0x20000 * i + 0x40 * x + 0x100C$
C-Name	GTM.CLS[i].TOM.CH[x].CM0

Interface: MCS[i]

Name	TOM[i]_CH[x]_CM0
Address	$0x40 * x + 0x100C$

C–Name	

CM0	
Description	TOM[i] channel [x] CCU0 compare register
Loop	–
Bit Range	$[15:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync

14.15.14 TOM[i]_CH[x]_SR0

Description	TOM[i] channel [x] CCU0 compare shadow register
Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_SR0
Address	$0x20000 * i + 0x40 * x + 0x1004$
C–Name	GTM.CLS[i].TOM.CH[x].SR0

Interface: MCS[i]

Name	TOM[i]_CH[x]_SR0
Address	$0x40 * x + 0x1004$
C–Name	

SR0	
Description	TOM[i] channel [x] shadow register TOM[i]_CH[x]_SR0 for update of compare register TOM[i]_CH[x]_CM0
Loop	–
Bit Range	$[15:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

SR0	
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync

14.15.15 TOM[i]_CH[x]_CM1

Description	TOM[i] channel [x] CCU1 compare register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_CM1
Address	$0x20000 * i + 0x40 * x + 0x1010$
C-Name	GTM.CLS[i].TOM.CH[x].CM1

Interface: MCS[i]

Name	TOM[i]_CH[x]_CM1
Address	$0x40 * x + 0x1010$
C-Name	

CM1	
Description	TOM[i] channel [x] CCU1 compare register
Loop	-
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync

14.15.16 TOM[i]_CH[x]_SR1

Description	TOM[i] channel [x] CCU1 compare shadow register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$

Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_SR1
Address	$0x20000 * i + 0x40 * x + 0x1008$
C-Name	GTM.CLS[i].TOM.CH[x].SR1

Interface: MCS[i]

Name	TOM[i]_CH[x]_SR1
Address	$0x40 * x + 0x1008$
C-Name	

SR1	
Description	TOM[i] channel [x] shadow register TOM[i]_CH[x]_SR1 for update of compare register TOM[i]_CH[x]_CM1
Loop	-
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync

14.15.17 TOM[i]_CH[x]_STAT

Description	TOM[i] channel [x] status register
Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_STAT
Address	$0x20000 * i + 0x40 * x + 0x1018$
C-Name	GTM.CLS[i].TOM.CH[x].STAT

Interface: MCS[i]

Name	TOM[i]_CH[x]_STAT

Address	$0x40 * x + 0x1018$
C-Name	

OL	
Description	Output level of output TOM_OUT[x:x]
Loop	-
Bit Range	$[0 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	TOM_OUT_RST
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
	Note: Reset value is the inverted value of **TOM[i]_CH[x]_CTRL.SL** bit which depends on the hardware configuration chosen by silicon vendor.

OSM_RTF	
Description	One-shot mode retrigger failed flag
Loop	-
Bit Range	$[29 : 29]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
R-Coding	0 : One-shot retrigger operation successful 1 : One-shot retrigger operation failed
W-Coding	0 : No action 1 : Clear status bit to zero
	Note: This bit will be cleared on a configuration interface write access of value 1. A read access leaves the bit unchanged.

14.15.18 TOM[i]_CH[x]_IRQ_NOTIFY

Description	TOM[i] channel [x] interrupt notification register

Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	TOM[i]_CLK_ENABLE == 1

Interface: CPU

Name	TOM[i]_CH[x]_IRQ_NOTIFY
Address	$0x20000 * i + 0x40 * x + 0x101C$
C-Name	GTM.CLS[i].TOM.CH[x].IRQ_NOTIFY

Interface: MCS[i]

Name	TOM[i]_CH[x]_IRQ_NOTIFY
Address	$0x40 * x + 0x101C$
C-Name	

CCU0TC	
Description	CCU0 Trigger condition interrupt for channel [x]
Loop	-
Bit Range	[0 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
R-Coding	0 : No interrupt triggered by software or by the compare event of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** -1 in CCU0 unit was raised. 1 : Interrupt triggered by software or by the compare event of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** -1 in CCU0 unit was raised.
W-Coding	0 : No action 1 : Clear interrupt
	Note: The notification of the interrupt is only triggered once after reaching the condition **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM0.CM0** -1. **Note:** This bit will be cleared on a configuration interface write access of value 1. A read access leaves the bit unchanged.

CCU1TC	
Description	CCU1 Trigger condition interrupt for channel [x]
Loop	-

CCU1TC	
Bit Range	$[1:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
R-Coding	TOM[i]_CH[x]_CTRL.SR0_TRIG == 0 0 : No interrupt triggered by software or the compare event of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1** -1 in CCU1 unit was raised. 1 : Interrupt triggered by software or by the compare event of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1** -1 in CCU1 unit was raised.
W-Coding	TOM[i]_CH[x]_CTRL.SR0_TRIG == 0 0 : No action 1 : Clear interrupt
R-Coding	TOM[i]_CH[x]_CTRL.SR0_TRIG == 1 0 : No interrupt triggered by software or the compare event of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_SR0.SR0** -1 in CCU0 unit was raised. 1 : Interrupt triggered by software or the compare event of **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_SR0.SR0** -1 in CCU0 unit was raised.
W-Coding	TOM[i]_CH[x]_CTRL.SR0_TRIG == 1 0 : No action 1 : Clear interrupt
	Note: If **TOM[i]_CH[x]_CTRL.SR0_TRIG** = 0, the notification of the interrupt is only triggered once after reaching the condition **TOM[i]_CH[x]_CN0.CN0** >= **TOM[i]_CH[x]_CM1.CM1** -1. **Note:** This bit will be cleared on a configuration interface write access of value 1. A read access leaves the bit unchanged.

14.15.19 TOM[i]_CH[x]_IRQ_EN

Description	TOM[i] channel [x] interrupt enable register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_IRQ_EN
Address	$0x20000 * i + 0x40 * x + 0x1020$
C-Name	GTM.CLS[i].TOM.CH[x].IRQ_EN

Interface: MCS[i]

Name	TOM[i]_CH[x]_IRQ_EN
Address	$0x40 * x + 0x1020$
C-Name	

CCU0TC_IRQ_EN	
Description	TOM_CCU0TC[x]_IRQ interrupt enable
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : The interrupt *TOM_CCU0TC[x]_IRQ* is not visible outside GTM-IP 1 : The interrupt *TOM_CCU0TC[x]_IRQ* is visible outside GTM-IP

CCU1TC_IRQ_EN	
Description	TOM_CCU1TC[x]_IRQ interrupt enable
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
RW-Coding	0 : The interrupt *TOM_CCU1TC[x]_IRQ* is not visible outside GTM-IP 1 : The interrupt *TOM_CCU1TC[x]_IRQ* is visible outside GTM-IP

14.15.20 TOM[i]_CH[x]_IRQ_FORCINT

Description	TOM[i] channel [x] force interrupt register
Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	TOM[i]_CLK_ENABLE == 1

Interface: CPU

Name	TOM[i]_CH[x]_IRQ_FORCINT
Address	$0x20000 * i + 0x40 * x + 0x1024$
C-Name	GTM.CLS[i].TOM.CH[x].IRQ_FORCINT

Interface: MCS[i]

Name	TOM[i]_CH[x]_IRQ_FORCINT
Address	$0x40 * x + 0x1024$
C-Name	

TRG_CCU0TC	
Description	Trigger the bit TOM[i]_CH[x]_IRQ_NOTIFY.CCU0TC by software
Loop	-
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 1, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **TOM[i]_CH[x]_IRQ_NOTIFY.CCU0TC** = 1
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

TRG_CCU1TC	
Description	Trigger the bit TOM[i]_CH[x]_IRQ_NOTIFY.CCU1TC by software
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0

TRG_CCU1TC	
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 1, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **TOM[i]_CH[x]_IRQ_NOTIFY.CCU1TC** = 1
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

14.15.21 TOM[i]_CH[x]_IRQ_MODE

Description	TOM[i] channel [x] interrupt mode register
Loop	$i = \{n : 0 <= n <= \text{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{TOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TOM[i]_CH[x]_IRQ_MODE
Address	$0x20000 * i + 0x40 * x + 0x1028$
C-Name	GTM.CLS[i].TOM.CH[x].IRQ_MODE

Interface: MCS[i]

Name	TOM[i]_CH[x]_IRQ_MODE
Address	$0x40 * x + 0x1028$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	-
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TOM[i]_RESET_CH[x]: sync

IRQ_MODE	
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

14.16 TOM Signal Description

14.16.1 TOM Reset

Loop	$j = \{n : 0 <= n <= \mathrm{NTOM} - 1\}; g = \{n : 0 <= n <= 1\}; c = \{n : 0 <= n <= 7\}$
Condition	-
Format	-

TOM[j]_RESET_CH[x]	
Description	TOM[j] channel [x] reset active signal
Loop	$x = \{n : g * 8 + c <= n <= g * 8 + c\}$
Condition	-
Signal Type	ARRAY[NTOM][16]
Assignment	TOM[j]_TGC[g]_GLB_CTRL.RST_CH[c] == 1

14.16.2 TOM Interrupt Signals

Loop	$x = \{n : 0 <= n <= 15\}$
Condition	-
Format	-

TOM_CCU1TC[x]_IRQ	
Description	CCU1 Trigger condition interrupt for channel x of each TOM instance i (i:={0, 1,..., NTOM-1})
Loop	-
Condition	-
Signal Type	ARRAY[16]
Assignment	-

TOM_CCU0TC[x]_IRQ	
Description	CCU0 Trigger condition interrupt for channel x of each TOM instance (i:={0, 1,..., NTOM-1})
Loop	-
Condition	-
Signal Type	ARRAY[16]
Assignment	-

14.16.3 TOM Signals

Loop	-

Condition	–
Format	–

CTRL_TRIG	
Description	Trigger signal to update the TGC[g] registers TOM[i]_TGC[g]_OUTEN_STAT, TOM[i]_TGC[g]_ENDIS_STAT and to activate TOM_FUPD signals
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TOM_EXT_TRIGIN_S	
Description	synchronized TOM_EXT_TRIGIN[x:x] (x:={0, 1,..., 15})signal to selected CMU clock source
Loop	–
Condition	–
Signal Type	–
Assignment	–

TOM_ENDIS	
Description	A bundle of channel enable signals controlled by TGC[g] (g:={0, 1}) unit
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TOM_OUTEN	
Description	A bundle of channel enable signals controlled by TGC[g] (g:={0, 1}) unit
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TOM_FUPD	
Description	A bundle of channel force update event signals controlled by TGC[g] (g:={0, 1}) unit
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TOM_RSTCN0_CH	
Description	A bundle of channel counter reset trigger signal at force update event that are controlled by TGC[g] (g:={0, 1}) unit
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TOM_UPEN	
Description	A bundle of channel update enable signals controlled by TGC[g] (g:={0, 1}) unit

TOM_UPEN	
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SEL_FXCMU_CLKEN	
Description	The selected fixed CMU clock enable signal
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TRIG_CCU0	
Description	Trigger signal for CCU0 compare match
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TRIG_CCU1	
Description	Trigger signal for CCU1 compare match
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TOM_EXT_HRES	
Description	Not internal signal from GTM or TOM. A signal to explain how the external circuit can generate a signal for each channel x that consists of edges from TOM_OUT[x:x] with delay of TOM_OUT_HRES[x].
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TOM_CH_TRIGIN	
Description	Trigger input signal for each channel x. It comes from the preceding channel x−1 or channel 15 of the preceding instance.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TOM_CH_TRIGOUT	
Description	Trigger output signal for each channel x. It is connected to the trigger input signal of the next channel x+1 or channel 0 of the next instance.
Loop	–
Condition	–
Signal Type	INT

TOM_CH_TRIGOUT	
Assignment	–

TOM_CH_HRES_IN[x]	
Description	HRES trigger input signal for each channel x. It is 5 bits value. Together with TOM_CH_TRIGIN[x:x], it comes from the preceding channel x-1 or channel 15 of the preceding instance.
Loop	$x = \{n : 0 <= n <= 15\}$
Condition	–
Signal Type	ARRAY[16]
Assignment	–

TOM_CH_HRES_OUT[x]	
Description	HRES trig out signal for each channel x. It is 5 bits value. Together with TOM_CH_TRIGOUT[x:x], it is connected with the HRES trigger input signal of the next channel x+1 or channel 0 of the next instance.
Loop	$x = \{n : 0 <= n <= 15\}$
Condition	–
Signal Type	ARRAY[16]
Assignment	–

TOM[i]_CH[x]_HRES_PERIOD	
Description	HRES value of TOM i channel x that is calculated with regards to TOM[i]_CH[x]_CM0.CM0.
Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	–
Signal Type	ARRAY[NTOM][16]
Assignment	–

TOM[i]_CH[x]_HRES_PULSEWIDTH	
Description	HRES value of TOM i channel x that is calculated with regards to TOM[i]_CH[x]_CM1.CM1.
Loop	$i = \{n : 0 <= n <= \mathrm{NTOM} - 1\}; x = \{n : 0 <= n <= 15\}$
Condition	–
Signal Type	ARRAY[NTOM][16]
Assignment	–

14.17 TOM Port Description

14.17.1 TOM Signal Interface

Loop	–
Condition	–
Format	–

TOM_EXT_TRIGIN	
Description	TOM_EXT_TRIGIN[7:0] of TOM instance i are connected to TIM_EXT_CAPTURE[7:0] signal from TIM instance i while TOM_EXT_TRIGIN[15:08] of TOM instance i are connected to TIM_EXT_CAPTURE[7:0] signal from TIM instance i
Loop	–

TOM_EXT_TRIGIN	
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	15 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM_TRIGIN	
Description	TOM module trigger chain input
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM_TRIGOUT	
Description	TOM module trigger chain output
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM_TRIGOUT_DEL	
Description	TOM module trigger chain output delayed by 1 cluster clock; will be used only in TOM [0] instance as TOM_TRIGIN
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK

TOM_TRIGOUT_DEL	
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM_OUT	
Description	Output signal TOM_OUT
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	15 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	TOM_OUT_RST

TOM_OUT_T	
Description	Output signal TOM_OUT_T
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	15 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	TOM_OUT_RST

14.17.2 TOM Interrupt Interface

Loop	–
Condition	–
Format	–

TOM_IRQ	
Description	TOM IRQ line signals
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	15 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–

TOM_IRQ	
Operational Reset	GTM_RESET
Reset Value	0

TOM_IRQ_CLR	
Description	TOM IRQ clear signal, hardware clear of ATOM[i]_CH[x]_IRQ_NOTIFY register
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	15 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM_IRQ_OCC	
Description	TOM IRQ occurred signal, corresponds to ATOM[i]_CH[x]_IRQ_NOTIFY register
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	15 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

14.17.3 TOM Signal HRES Interface

Loop	–
Condition	–
Format	–

TOM_OUT_HRES[x]	
Description	TOM_OUT high-resolution output signal of channel x
Loop	$x = \{n : 0 <= n <= 15\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–

TOM_OUT_HRES[x]	
Operational Reset	GTM_RESET
Reset Value	0

TOM_OUT_T_HRES[x]	
Description	TOM_OUT_T high-resolution output signal of channel x
Loop	$x = \{n : 0 <= n <= 15\}$
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	-
Operational Reset	GTM_RESET
Reset Value	0

TOM_HRES_IN	
Description	TOM high-resolution signal trigger chain input;will be driven by previous TOM instance i-1
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	-
Operational Reset	GTM_RESET
Reset Value	0

TOM_HRES_OUT	
Description	TOM high-resolution signal trigger chain output;will be used by following TOM instance i+1
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	-
Operational Reset	GTM_RESET
Reset Value	0

TOM_HRES_OUT_DEL	
Description	TOM high-resolution trigger chain output delayed by 1 cluster clock;will be used only in TOM0 instance as T-OM_HRES_IN

TOM_HRES_OUT_DEL	
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

14.17.4 TOM SPE Interface

Loop	–
Condition	–
Format	–

TOM[i]_SPE_NIPD	
Description	It is special interface signal connected to SPE[i]_NIPD of SPE module i. That means new input pattern detected by SPE module i.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM[i]_SPE_OUT	
Description	It is special interface signal connected to SPE[i]_OUT of SPE module i.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM[i]_CH[x]_SOUR	
Description	It is special interface signals to SPE instance i; driven by TOM instance i channel x.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}; x = \{n : 0 <= n <= 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM[i]_CH[x]_TRIG_CCU1	
Description	It is special interface signal to SPE instance i; connected to the internal signal TRIG_CCU1 of TOM instance i channel x (with x∈{0, 2, 6, 7, 8, 9}.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

TOM[i]_CH[x]_TRIG_CCU0	
Description	It is special interface signals to SPE instance i; connected to the internal signal TRIG_CCU0 of driven by TOM instance i channel x (with x = 0) .
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

15 ARU−connected Timer Output Module (ATOM)

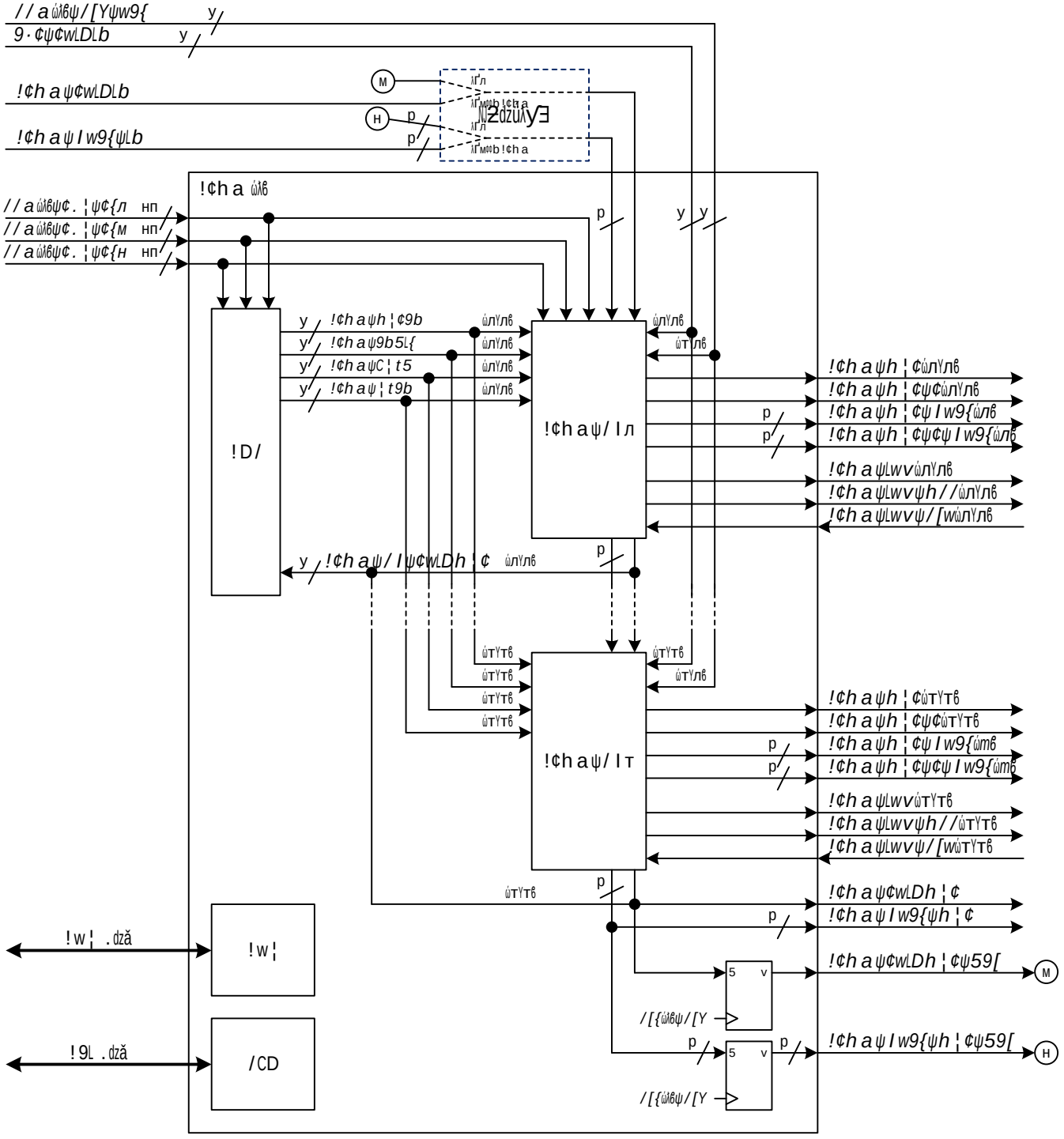
15.1 Overview

The ARU−connected Timer Output Module (ATOM) is able to generate complex output signals without interaction via the configuration interface due to its connectivity to the ARU. Typically, output signal characteristics are provided over the ARU connection through submodules connected to ARU, e.g. the MCS, DPLL or PSM. Each ATOM submodule contains eight output channels which can operate independently from each other in several configurable operation modes. A block diagram of the ATOM submodule is depicted in figure *79 "ATOM block diagram"* .

Indices and their ranges as used inside this chapter are:

- i:={0, 1,..., NATOM−1} instance index of cluster / module
- NATOM= number of ATOM instances in the device
- x:={0, 1,..., 7} index of channels in each ATOM instance
- k:={0, 1,..., 7} index of sub−channels

Figure 79 ATOM block diagram



The architecture of ATOM submodule is similar to TOM submodule with some differences. Firstly, each ATOM integrates only eight channels with outputs *ATOM_OUT* and *ATOM_OUT_T*. There is one ATOM Global Control sub−unit (AGC) for the ATOM channels. Secondly, each ATOM instance is connected to ARU and can set up individual read requests from ARU and write requests to ARU. Furthermore, ATOM channels are able to generate various PWM signals in different modes.

The ATOM output signal routing to DTM or GTM−IP top level is described in chapter *16.9 "DTM Connections on GTM−IP Top Level"*

Each ATOM channel provides five modes of operation:

- ATOM Signal Output Mode Immediate (SOMI)
- ATOM Signal Output Mode Compare (SOMC)
- ATOM Signal Output Mode PWM (SOMP)
- ATOM Signal Output Mode Serial (SOMS)
- ATOM Signal Output Mode Buffered Compare (SOMB)

These modes are described in more detail in section *15.3 "ATOM Channel Modes"* .

The input clocks for the ATOM channels come from the cluster clock *CLS[i]_CLK* with the configurable clock resolution *CCM[i]_CLK_RES* signals of the CCM submodule. This gives the freedom to select a programmable input clock for the ATOM channel counters. The module ATOM receives three timestamp values *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* and *CCM[i]_TBU_TS2* in order to realize synchronized output behavior on behalf of a common time base.

The operation registers (e.g. counter, compare registers) of the ATOM channel are 24-bit wide. Each ATOM channel provides a so called operation register set **ATOM[i]_CH[x]_CM0** , **ATOM[i]_CH[x]_CM1** and shadow register set **ATOM[i]_CH[x]_SR0** , **ATOM[i]_CH[x]_SR1** . With this architecture it is possible to work with the operation register set, while the shadow register set can be reloaded with new parameters over configuration interface and/or ARU.

Especially, when ARU communication is enabled, each ATOM channel can independently request and receive data from other modules via ARU interface to control the output behavior. Furthermore, it can also provide data to the ARU in SOMC mode. Please refer to different mode description for more details.

As shown in figure *79 "ATOM block diagram"* , NATOM-1 ATOM sub-modules are connected together through a trigger chain. The trigger signal *ATOM_TRIGIN* of ATOM instance i is coming from *ATOM_TRIGOUT* of ATOM instance i-1 and the trigger signal *ATOM_TRIGOUT* of ATOM instance i is routed to *ATOM_TRIGIN* of ATOM instance i+1. This trigger chain goes across all channels one by one. The trigger input signal *ATOM_TRIGIN* of ATOM instance i is connected to the trigger input signal *ATOM_CH_TRIGIN* [0:0] of channel 0. The channel 0 can generate a trigger output signal *ATOM_CH_TRIGOUT* [0:0] that is connected to *ATOM_CH_TRIGIN* [1:1] to trigger channel 1. In this way, each channel x is triggered from the preceding channel and then can trigger the next channel (please refer to figure *81 "ATOM channel architecture in SOMP mode"* and chapter *15.2.3 "Internal Trigger Interface"* for the internal trigger chain details). Channel 7 of ATOM instance i is routed to trigger channel 0 of the next instance i+1.
Similarly, there is also a HRES value chain that goes across all channels of each instance along with the trigger chain. The high resolution value input *ATOM_HRES_IN* of ATOM instance i is connected to *ATOM_HRES_OUT* of ATOM instance i-1 and the high resolution value output *ATOM_HRES_OUT* of ATOM instance i is routed to *ATOM_HRES_IN* of ATOM instance i+1. Please refer to chapter *15.3.3.10 "High resolution PWM support (HRES mode)"* for more details.

Especially, the trigger input signal ATOM instance 0 is connected to its own output *ATOM_TRIGOUT_DEL* , i.e. the last channel of ATOM instance 0 can trigger the first channel of ATOM instance 0 (this path is registered, which means delayed by one cluster clock period).
Similarly, the high resolution value input *ATOM_HRES_IN* of ATOM instance 0 is connected to its own output *ATOM_HRES_OUT_DEL* , i.e. the high resolution value output of the last channel of ATOM instance 0 is also fed to the first channel of ATOM instance 0, also with one pipeline register delay.

The function of the interrupt signals *ATOM_IRQ* , *ATOM_IRQ_OCC* and *ATOM_IRQ_CLR* is described in more details in chapter *3.12 "GTM-IP Interrupt Concept"* .

15.1.1 ATOM Global Control (AGC)

Synchronous start, stop and update of work register of up to 8 channels is possible with the AGC sub-unit. This sub-unit has the same functionality as the TGC sub-unit of the TOM sub-module.

15.1.1.1 Overview

One global channel control unit (AGC) exists to drive a number of individual ATOM channels synchronously by external or internal events.

An AGC can drive up to eight ATOM channels.

The ATOM submodule supports four different kinds of signaling mechanisms:

- Global enable/disable mechanism for each ATOM channel with control register **ATOM[i]_AGC_ENDIS_CTRL** and status register **ATOM[i]_AGC_ENDIS_STAT**
- Global output enable mechanism for each ATOM channel with control register **ATOM[i]_AGC_OUTEN_CTRL** and status register **ATOM[i]_AGC_OUTEN_STAT**
- Global force update mechanism of the register **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL.SL** and **ATOM[i]_CH[x]_CTRL.CLK_SRC** for each ATOM channel and optionally reset the counter **ATOM[i]_CH[x]_CN0** with control register **ATOM[i]_AGC_FUPD_CTRL**
- Update enable of the register **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL.SL** and **ATOM[i]_CH[x]_CTRL.CLK_SRC** for each ATOM channel with the control bit field **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]**

15.1.1.2 AGC Sub-unit

Each of the first three individual mechanisms (enable/disable of the channel, output enable and force update) can be driven by three different trigger sources.
The three trigger sources are:

1. over the configuration interface (bit **ATOM[i]_AGC_GLB_CTRL.HOST_TRIG**)
2. the TBU time stamp (signal *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* and *CCM[i]_TBU_TS2* if available)

3. the internal trigger signal *ATOM_CH_TRIGOUT* (bunch of trigger signals *ATOM_CH_TRIGOUT* [x:x]) which can be either the trigger *ATOM_TRIG_CCU0* of channel x, the trigger of preceding channel x-1 (i.e. signal *ATOM_CH_TRIGOUT* [x-1:x-1]) or the external trigger *EXT_TRIGIN* [x:x] of assigned TIM channel x.

The first way (1.) is to trigger the control mechanism by a direct register write access via configuration interface (bit **ATOM[i]_AGC_GLB_CTRL.HOST_TRIG**).

The second way (2.) is provided by a compare match trigger on behalf of a specified time base coming from the module TBU (selected by bits **ATOM[i]_AGC_ACT_TB.TBU_SEL**) and the time stamp compare value defined in the bit field **ATOM[i]_AGC_ACT_TB.ACT_TB** .

In this case, the selected TBU time base *CCM[i]_TBU_TS0* / *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* should be configured to be up-counting. A cyclic event compare of **ATOM[i]_AGC_ACT_TB.ACT_TB** and selected *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , *CCM[i]_TBU_TS2* is performed. Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more information about cyclic event compare strategy.

The third way (3.) is the input *ATOM_CH_TRIGOUT* (bunch of trigger signals *ATOM_CH_TRIGOUT* [x:x]) coming from ATOM channels.

The corresponding trigger signal *ATOM_CH_TRIGOUT* [x:x] coming from channel x can be masked by the register **ATOM[i]_AGC_INT_TRIG** .

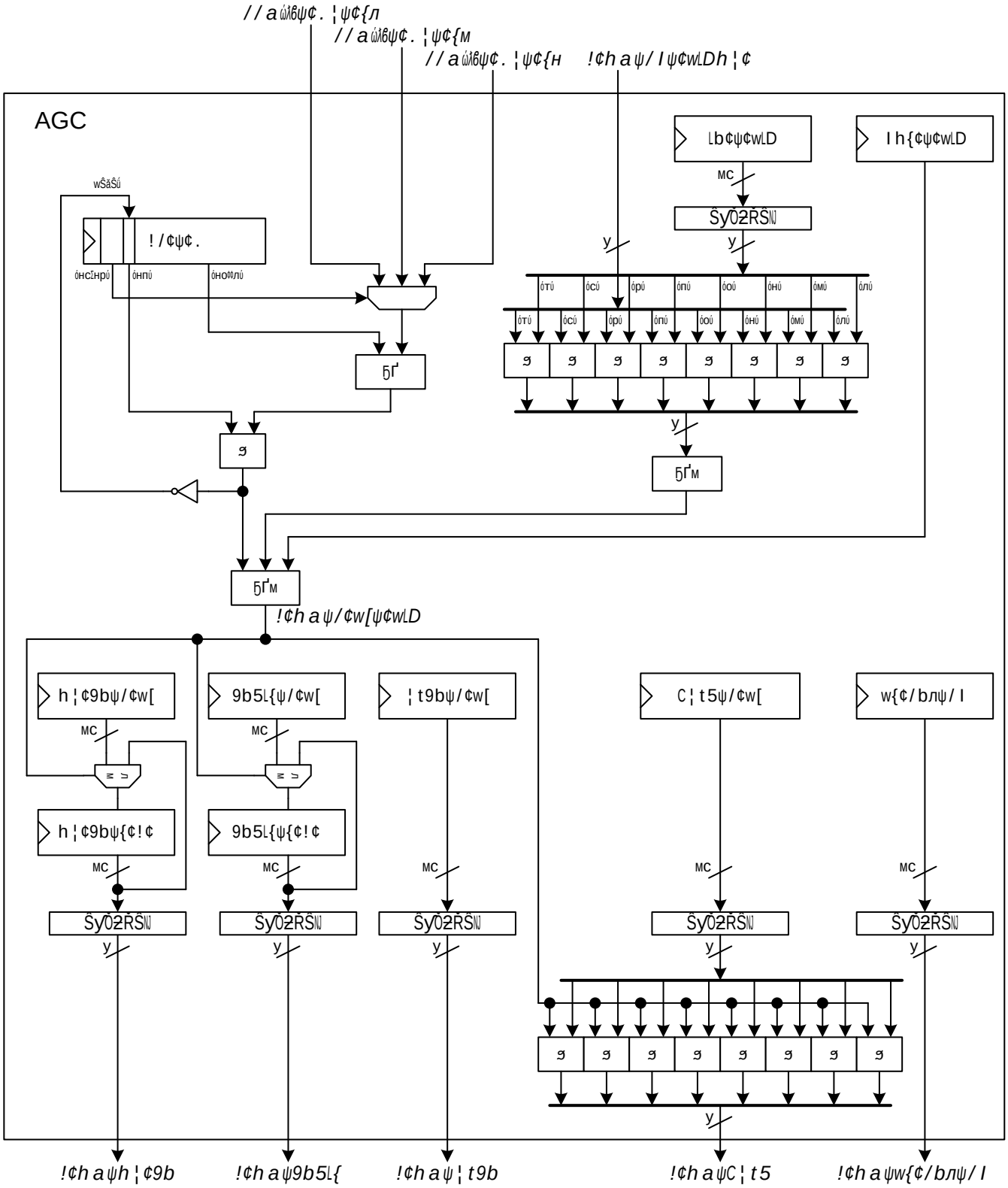
To enable or disable each individual ATOM channel, the registers **ATOM[i]_AGC_ENDIS_CTRL** and/or **ATOM[i]_AGC_ENDIS_STAT** have to be used.

The register **ATOM[i]_AGC_ENDIS_STAT** directly controls the signal *ATOM_ENDIS* [x:x]. A write access to this register is possible.

The register **ATOM[i]_AGC_ENDIS_CTRL** is a shadow register that overwrites the value of register **ATOM[i]_AGC_ENDIS_STAT** if one of the three trigger conditions matches.

To ensure a clean restart of a ATOM channel, it is recommended to raise a channel reset by setting of bit **ATOM[i]_AGC_GLB_CTRL.RST_CH[x]** after a channel was disabled (*ATOM_ENDIS* [x:x]=0) and before it is enabled again (*ATOM_ENDIS* [x:x]=1).

Figure 80 ATOM Global channel control mechanism



The output of the individual ATOM channels can be controlled using the registers **ATOM[i]_AGC_OUTEN_CTRL** and **ATOM[i]_AGC_OUTEN_STAT** .

The register **ATOM[i]_AGC_OUTEN_STAT** directly controls the signal *ATOM_OUTEN* . A write access to this register is possible.

The register **ATOM[i]_AGC_OUTEN_CTRL** is a shadow register that overwrites the value of register **ATOM[i]_AGC_OUTEN_STAT** if one of the three trigger conditions matches.

The trigger condition has always priority over the bus write access to the **ATOM[i]_AGC_OUTEN_STAT** and **ATOM[i]_AGC_ENDIS_STAT** registers, even if **ATOM[i]_AGC_OUTEN_CTRL.OUTEN_CTRL[k]** / **ATOM[i]_AGC_OUTEN_CTRL.ENDIS_CTRL[k]** is set to 0b00. This means

that the bus write access to **ATOM[i]_AGC_OUTEN_STAT** and **ATOM[i]_AGC_ENDIS_STAT** register is ignored in the clock cycle when the trigger condition is active.

If an ATOM output is disabled by the register **ATOM[i]_AGC_OUTEN_STAT** , the actual value of the channel output at *ATOM_OUT* [x:x] is set as ! **ATOM[i]_CH[x]_CTRL.SL** .
If the output is enabled, the output at *ATOM_OUT* [x:x] depends on the current operation of different mode, **ATOM[i]_AGC_ENDIS_STAT** and **ATOM[i]_CH[x]_CTRL.FREEZE** . (Please check **ATOM[i]_CH[x]_CTRL_SOMP.SL** / **ATOM[i]_CH[x]_CTRL_SOMI.SL** / **ATOM[i]_CH[x]_CTRL_SOMB.SL** / **ATOM[i]_CH[x]_CTRL_SOMS.SL** for more information)

The register **ATOM[i]_AGC_FUPD_CTRL.FUPD_CTRL** defines which of the ATOM channels receive a force update event if the trigger signal *ATOM_CTRL_TRIG* is raised while **ATOM[i]_AGC_FUPD_CTRL.RSTCN0_CH[k]** determines whether the force update event also reset counter **ATOM[i]_CH[x]_CN0.CN0** of the channel. If in SOMP continuous counting up mode with **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0 the force update mechanism is configured with **ATOM[i]_AGC_FUPD_CTRL.RSTCN0_CH[x]** =1, the force update event will simultaneously update the operation registers, reset the counter and also set the output to be **ATOM[i]_CH[x]_CTRL_SOMP.SL** . That means the force update mechanism can be used to stop the current PWM signal generation and restart a new PWM signal generation with new PWM parameters.

In SOMP, SOMI and SOMS modes, the force update request is synchronized to the selected *CCM[i]_CLK_RES* as shown in figure *81 "ATOM channel architecture in SOMP mode"* and then executed. In other modes, the force update request is executed immediately.

The register bits **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** defines, for which ATOM channel the update of the working register **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL.SL** and **ATOM[i]_CH[x]_CTRL.CLK_SRC** is enabled by the corresponding shadow register **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** , **ATOM[i]_CH[x]_CTRL_SR.SL_SR** and **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** . If update is enabled in SOMP continuous counting up mode, the register **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL.SL** and **ATOM[i]_CH[x]_CTRL.CLK_SRC** will be updated on reset of counter register **ATOM[i]_CH[x]_CN0.CN0** (see figure *81 "ATOM channel architecture in SOMP mode"*).

15.1.2 ATOM Channel Mode Overview

Each ATOM channel offers the following different operation modes:

In ATOM Signal Output Mode Immediate (SOMI), the ATOM channels generate an output signal according to **ATOM[i]_CH[x]_CTRL_SOMI.ACB0** when ARU communication is disabled or according to the two signal level output bits of the ARU word received through the **ATOM[i]_CH[x]_STAT.ACBI** when ARU communication is enabled. Due to the fact, that the ARU destination channels are served in a Round Robin order, the output signal can jitter in this mode with a jitter of the ARU round trip time.

In ATOM Signal Output Mode Compare (SOMC), the ATOM channel generates an output signal on behalf of time stamps that are located in the ATOM operation registers **ATOM[i]_CH[x]_CM0** and **ATOM[i]_CH[x]_CM1** . The output generates edges when the TBU time stamps reach the time stamps stored in **ATOM[i]_CH[x]_CM0** and **ATOM[i]_CH[x]_CM1** . The ATOM is able to receive new time stamps either by configuration interface or via the ARU. The new time stamps are directly loaded into the channels operation register. The shadow registers are used as capture registers for two time base values, when a compare match of the channels operation registers occurs.

In ATOM Signal Output Mode PWM (SOMP), the ATOM channel is able to generate simple and complex PWM output signals like the TOM submodule by comparing its operation registers with a submodule internal counter. Unlike the TOM, the ATOM shadow registers can be reloaded over the configuration interface and by the ARU in the background, while the channel operates on the operation registers. Please refer to section *15.3.3 "ATOM Signal Output Mode PWM (SOMP)"* for further details.

In ATOM Signal Output Mode Serial (SOMS), the ATOM channel generates a serial output bit stream on behalf of a shift register. The number of bits shifted and the shift direction is configurable. The shift frequency is determined by one of the *CCM[i]_CLK_RES* clock resolution signals. Please refer to section *15.3.4 "ATOM Signal Output Mode Serial (SOMS)"* for further details.

In ATOM Signal Output Buffered Compare (SOMB), the ATOM channel generates an output signal on behalf of time stamps that are located in the ATOM operation registers **ATOM[i]_CH[x]_CM0** and **ATOM[i]_CH[x]_CM1** . The output generates edges when the TBU time stamps reach the time stamps stored in **ATOM[i]_CH[x]_CM0** and **ATOM[i]_CH[x]_CM1** . The ATOM is able to receive new compare values either over the configuration interface or via the ARU. The new compare values received via configuration interface or via ARU are stored first in the shadow register. Only if previous compare match has occurred, the operation registers are updated with the content of the shadow registers. Please refer to section *15.3.5 "ATOM Signal Output Mode Buffered Compare(SOMB)"* for further details.

15.2 ATOM Channel Architecture

Each ATOM channel is able to generate output signals according to five operation modes. The architecture of the ATOM channels is similar to the architecture of the TOM channels. The architecture of an ATOM channel in SOMP mode is depicted in figure *81 "ATOM channel architecture in SOMP mode"* whereas the ARU interface is shown in picture *94 "ARU Data input stream pipeline structure for SOMP mode"* .

Note:

Figure *81 "ATOM channel architecture in SOMP mode"* doesn't reflect the real implementation of the circuit but is intended to clearly explain the functionality of the channel, especially for SOMP mode.

Figure 81 ATOM channel architecture in SOMP mode

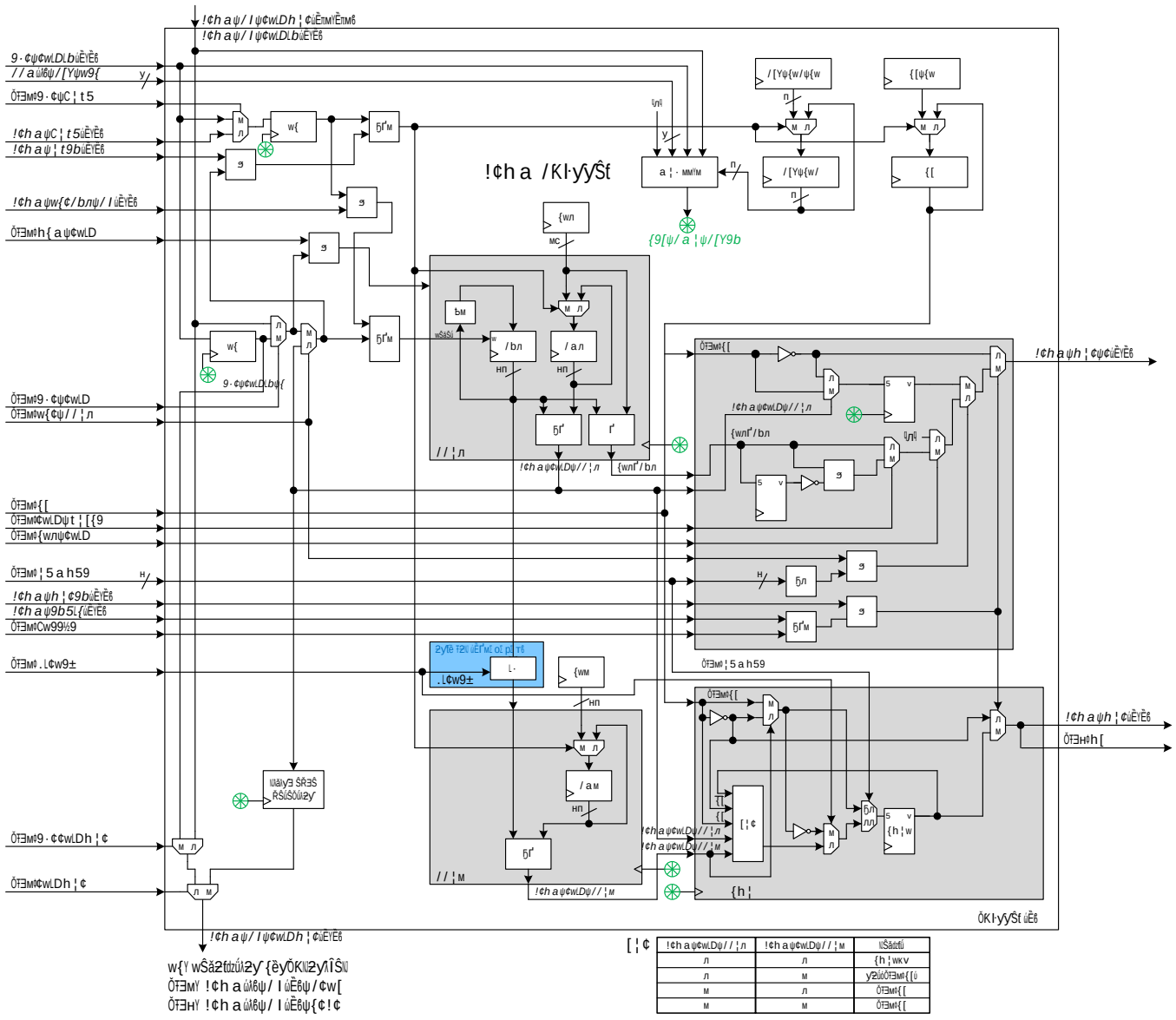
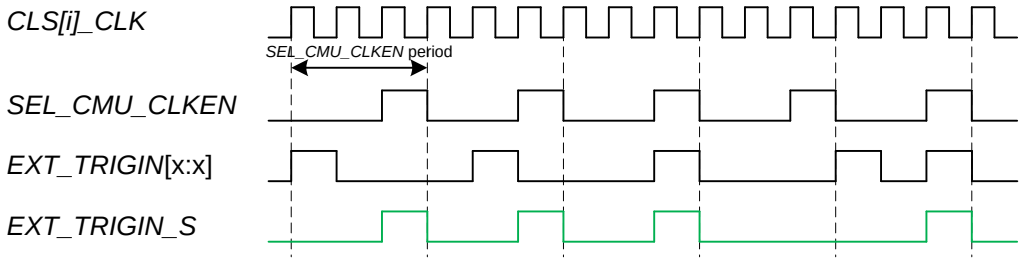


Figure 82 Timing diagram of RS unit



As shown in figure *81 "ATOM channel architecture in SOMP mode"* , in SOMP and SOMS mode, the operating clock resolution signal *SEL_CMU_CLKEN* can be configured by **ATOM[i]_CH[x]_CTRL.CLK_SRC** . This configuration register can select the clock source from CMU clock enable signals *CCM[i]_CLK_RES* that are provided from CCM submodule.
In addition, with the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , it is possible to change the selected clock resolution from CMU under different conditions. The signal *SEL_CMU_CLKEN* looks like clock enable signal that is shown in figure *82 "Timing diagram of RS unit"* . Most functionalities in SOMP and SOMS modes are only active when *SEL_CMU_CLKEN* = 1 and the consequent operation registers and outputs are always synchronously updated at the falling edge of *SEL_CMU_CLKEN* pulse. The functionalities synchronous to the selected clock resolution *SEL_CMU_CLKEN* are depicted with the green round symbol in figure *81 "ATOM channel architecture in SOMP mode"* .

Due to a fixed clock resolution scheme from CMU, the first clock resolution period may differ from the expected value, depending on the point of time, where the **ATOM[i]_CH[x]_CTRL.CLK_SRC** bit field is updated from its shadow register bit field **ATOM[i]_CH[x]_CTRL_SR.-CLK_SRC_SR** .

In all ATOM channels the operation registers **ATOM[i]_CH[x]_CN0.CN0** , **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** and the shadow registers **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are 24−bit wide. As shown in figure *81 "ATOM channel*

architecture in SOMP mode" , the unit CCU0 provides different counting up/down functionalities in various modes. The comparators inside CCU0 and CCU1 perform selectable signed greater-equal or less-equal comparison against different parameters depending on different ATOM modes and then generate ATOM_TRIG_CCU0=1 when the compare match event in CCU0 occurs or generate ATOM_TRIG_CCU1=1 when the compare match event in CCU1 occurs.

The Signal Output Unit (SOU) generates the output signal for each ATOM channel. In SOMI mode, this output signal level depends on the bit **ATOM[i]_CH[x]_CTRL_SOMI.SL** in combination with two control bits **ATOM[i]_CH[x]_CTRL_SOMI.ACB0** (if controlled by configuration interface) or **ATOM[i]_CH[x]_STAT.ACBI** [0:0] (if controlled by ARU communication).

The Signal Output Unit (SOU) generates the output signal for each ATOM channel. In SOMC mode, this output signal level depends on the bit **ATOM[i]_CH[x]_CTRL_SOMC.SL** in combination with five control bits **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** (if controlled by configuration interface) or **ATOM[i]_CH[x]_STAT.ACBI** (if controlled by ARU communication).

The Signal Output Unit (SOU) generates the output signal for each ATOM channel. In SOMS mode, this output signal level depends on the bit **ATOM[i]_CH[x]_CTRL_SOMS.SL** in combination with two control bits **ATOM[i]_CH[x]_CTRL_SOMS.ACB0** (if controlled by configuration interface) or **ATOM[i]_CH[x]_STAT.ACBI** [0:0] (if controlled by ARU communication).

The Signal Output Unit (SOU) generates the output signal for each ATOM channel. In SOMP mode, this output signal level depends on the bit **ATOM[i]_CH[x]_CTRL_SOMP.SL** in combination with the counter behavior and its compare events in CCU0 and CCU1.

The Signal Output Unit (SOU) generates the output signal for each ATOM channel. In SOMB mode, this output signal level depends on the bit **ATOM[i]_CH[x]_CTRL_SOMB.SL** in combination with five control bits **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** (if controlled by configuration interface) or **ATOM[i]_CH[x]_STAT.ACBI** (if controlled by ARU communication).

In SOMP continuous counting up mode, **ATOM[i]_CH[x]_CM0.CM0** represents PWM period while **ATOM[i]_CH[x]_CM1.CM1** is PWM pulse width. In order to generate various PWM signals, CCU0 counts up **ATOM[i]_CH[x]_CN0.CN0** and compares it with **ATOM[i]_CH[x]_CM0.CM0** while CCU1 compares it with **ATOM[i]_CH[x]_CM1.CM1** . Depending on the configuration bit **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** , the counter **ATOM[i]_CH[x]_CN0.CN0** can be reset, when the counter value is equal to the compare value **ATOM[i]_CH[x]_CM0.CM0** (i.e. **ATOM[i]_CH[x]_CN0.CN0** counts only from 0 to **ATOM[i]_CH[x]_CM0.CM0** -1 and is then reset to 0) or when it is triggered by the ATOM[i] trigger signal *ATOM_CH_TRIGIN* [x:x] (*ATOM_CH_TRIGOUT* [x-1:x-1]) of the preceding channel x-1 which can also be the last channel of preceding instance ATOM[i-1]) or when it is triggered by the external trigger signal *EXT_TRIGIN* [x:x] of the assigned TIM channel x.

The external trigger signal *EXT_TRIGIN* [x:x] is synchronous to the clock source selected by RS (Resolution Synchronizer) unit and then taken into use.
With such a RS unit, in each *SEL_CMU_CLKEN* period (i.e. selected clock resolution signal), any event (or pulses) of *EXT_TRIGIN* will be synchronized by RS unit on *EXT_TRIGIN_S* that looks exactly the same as *SEL_CMU_CLKEN* . Figure *82 "Timing diagram of RS unit"* shows the timing diagram of RS unit. If *EXT_TRIGIN* has one or more pulses in a *SEL_CMU_CLKEN* period, the synchronized output *EXT_TRIGIN_S* will have 1 pulse at the end of this period. Otherwise, *EXT_TRIGIN_S* is always 0.

The trigger signal *ATOM_CH_TRIGIN* [x:x] coming from the preceding channel may not be synchronous to the selected clock resolution of channel x.

In SOMP continuous counting up mode, once the comparison event of **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM0.CM0** -1 in unit CCU0 occurs, *ATOM_TRIG_CCU0* = 1 will be set to trigger the output generation of *ATOM_OUT* [x:x] in the SOU sub-unit together with the corresponding interrupt signals. In order to trigger the next channel, the rising edge detection unit can detect each rising edge of *ATOM_TRIG_CCU0* (i.e. each comparison event) and generate a single cluster clock pulse on *ATOM_CH_TRIGOUT* [x:x] when the selected clock resolution signal *SEL_CMU_CLKEN* =1 (i.e. synchronous to the selected clock resolution). In addition, *ATOM_CH_TRIGOUT* [x:x] can also be configured to be *ATOM_CH_TRIGIN* [x:x] (*ATOM_CH_TRIGOUT* [x-1:x-1] of the preceding channel x-1) or the external trigger signal *EXT_TRIGIN* [x:x] of the assigned TIM channel x. Similarly, once the comparison event of **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM1.CM1** -1 in unit CCU1 occurs, *ATOM_TRIG_CCU1* = 1 will be set to trigger the output generation of *ATOM_OUT* [x:x] in the SOU sub-unit together with the corresponding interrupt signals.

When **ATOM[i]_CH[x]_CN0.CN0** < **ATOM[i]_CH[x]_CM0.CM0** -1, *ATOM_TRIG_CCU0* will be reset as 0. Similarly, when **ATOM[i]_CH[x]_CN0.CN0** < **ATOM[i]_CH[x]_CM1.CM1** -1, *ATOM_TRIG_CCU1* will be reset to 0.

As shown in figure *82 "Timing diagram of RS unit"* , in SOMP continuous counting up mode according to the LUT with *ATOM_TRIG_CCU0* , *ATOM_TRIG_CCU1* and the signal level configuration **ATOM[i]_CH[x]_CTRL.SL** , SOU unit can generate *ATOM_OUT* [x:x] with various PWM timing characteristics when the channel and the output are both enabled (*ATOM_OUTEN* [x:x]=1 and *ATOM_ENDIS* [x:x]=1). Please refer to the chapter of different modes for more details on PWM signal generation.

In SOMC/SOMB mode, CCU0 and CCU1 compare **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** against the GTM time bases *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , and if available, *CCM[i]_TBU_TS2* generates the compare events *ATOM_TRIG_CCU0* and *ATOM_TRIG_CCU1* . Please refer to TBU chapter *12 "Time Base Unit (TBU)"* for further details. In SOMC/SOMB mode, the two compare units CCU0/CCU1 can be used in combination with each other. When used in combination, the signals *ATOM_TRIG_CCU0* and *ATOM_TRIG_CCU1* can be used to enable/disable the other compare unit when a match event occurs. Different compare strategy can be configured in **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** / **ATOM[i]_CH[x]_CTRL_SOMB.ACB42** . Based on the signal level register bit **ATOM[i]_CH[x]_CTRL.SL** and the two control bits **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** / **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** , the output *ATOM_OUT* [x:x] reflects the configured compare strategy. This behavior is not really illustrated in figure *81 "ATOM channel architecture in SOMP mode"* . Please refer to the sections *15.3.2 "ATOM Signal Output Mode Compare (SOMC)"* and *15.3.5 "ATOM Signal Output Mode Buffered Compare(SOMB)"* for further details.

In SOMC/SOMB mode, CCU0/CCU1 unit always performs cyclic event compare between the selected time base *CCM[i]_TBU_TS0* / *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* and the compare value **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** . When the selected time base is "cyclically greater than or equal to" the compare value of **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** , the compare match event specified in **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** is considered to have occurred. Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more information about cyclic event compare strategy.

For a correct behavior of cyclic event compare in SOMC/SOMB mode, when a compare match event with the specified compare value **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** occurs, the new compare value in **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** for the next cyclic event compare must be specified in the range of the half total time base value (i.e. 0x7FFFFF) greater/smaller than the old compare value.

In SOMI mode, the output signal level depends on **ATOM[i]_CH[x]_CTRL.SL** and **ATOM[i]_CH[x]_CTRL_SOMI.ACB0** . In SOMS mode, the output signal level is defined by the bit pattern that has to be shifted out by the ATOM channel. The bit pattern is located inside the **ATOM[i]_CH[x]_CM1.CM1** register.

When the output is disabled (*ATOM_OUTEN* [x:x]=0), the output signals *ATOM_OUT* [x:x] and *ATOM_OUT_T* [x:x] are the inverse value of the **ATOM[i]_CH[x]_CTRL.SL** bit. When the output is enabled (*ATOM_OUTEN* [x:x]=1), the output signals *ATOM_OUT* [x:x] and *ATOM_OUT_T* [x:x] are dependent on different operation modes, the channel enable signal *ATOM_ENDIS* [x:x] and the freeze mode enable register **ATOM[i]_CH[x]_CTRL.FREEZE** .(Please refer to register description of **ATOM[i]_CH[x]_CTRL_SOMI.SL** / **ATOM[i]_CH[x]_CTRL_SOMB.SL** / **ATOM[i]_CH[x]_CTRL_SOMC.SL** / **ATOM[i]_CH[x]_CTRL_SOMP.SL** / **ATOM[i]_CH[x]_CTRL_SOMS.SL** for more details.) Especially, in SOMB, SOMC and SOMI modes, the output signal *ATOM_OUT_T* [x:x] is not supported. So it is always clamped to 0 if the output and the channel are both enabled (i.e. *ATOM_OUTEN* [x:x]=1 and *ATOM_ENDIS* [x:x]=1). When the output is enabled (*ATOM_OUTEN* [x:x]=1) but the channel is disabled (*ATOM_ENDIS* [x:x]=0), if **ATOM[i]_CH[x]_CTRL.FREEZE** =0, *ATOM_OUT_T* [x:x] is ! **ATOM[i]_CH[x]_CTRL.SL** . When the output is enabled (*ATOM_OUTEN* [x:x]=1) but the channel is disabled (*ATOM_ENDIS* [x:x]=0), if **ATOM[i]_CH[x]_CTRL.FREEZE** =1, *ATOM_OUT_T* [x:x] is clamped to 0.

When the channel is disabled (*ATOM_ENDIS* [x:x]=0), the counting and comparing functionalities in CCU0 and CCU1 are deactivated in different modes and thus *ATOM_TRIG_CCU0* = 0, *ATOM_TRIG_CCU1* = 0. Therefore, *ATOM_CH_TRIGOUT* [x:x] is not generated if **ATOM[i]_CH[x]_CTRL.TRIGOUT** = 1, regardless of *ATOM_OUTEN* [x:x].

Similar to **ATOM[i]_CH[x]_CTRL.CLK_SRC** , **ATOM[i]_CH[x]_CTRL.SL** , **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** also have their shadow registers **ATOM[i]_CH[x]_CTRL_SR.SL_SR** , **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** . In SOMP and SOMS mode, it is possible to synchronously update the clock resolution register **ATOM[i]_CH[x]_CTRL.CLK_SRC** and the operation registers **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** from their shadow registers. In addition, synchronous updating of the signal level register **ATOM[i]_CH[x]_CTRL.SL** is possible in SOMP mode. There are different update mechanisms in SOMP and SOMS mode. Please refer to the chapters SOMP and SOMS modes for more details.

In addition, the operation registers mentioned above can also be controlled by ARU communication interface (ACI). Each ATOM channel can request data from other modules through ARU to control different operations in SOMI, SOMP, SOMB, SOMC and SOMS modes and additionally can provide data to the ARU in SOMC mode.

15.2.1 ARU Communication Interface

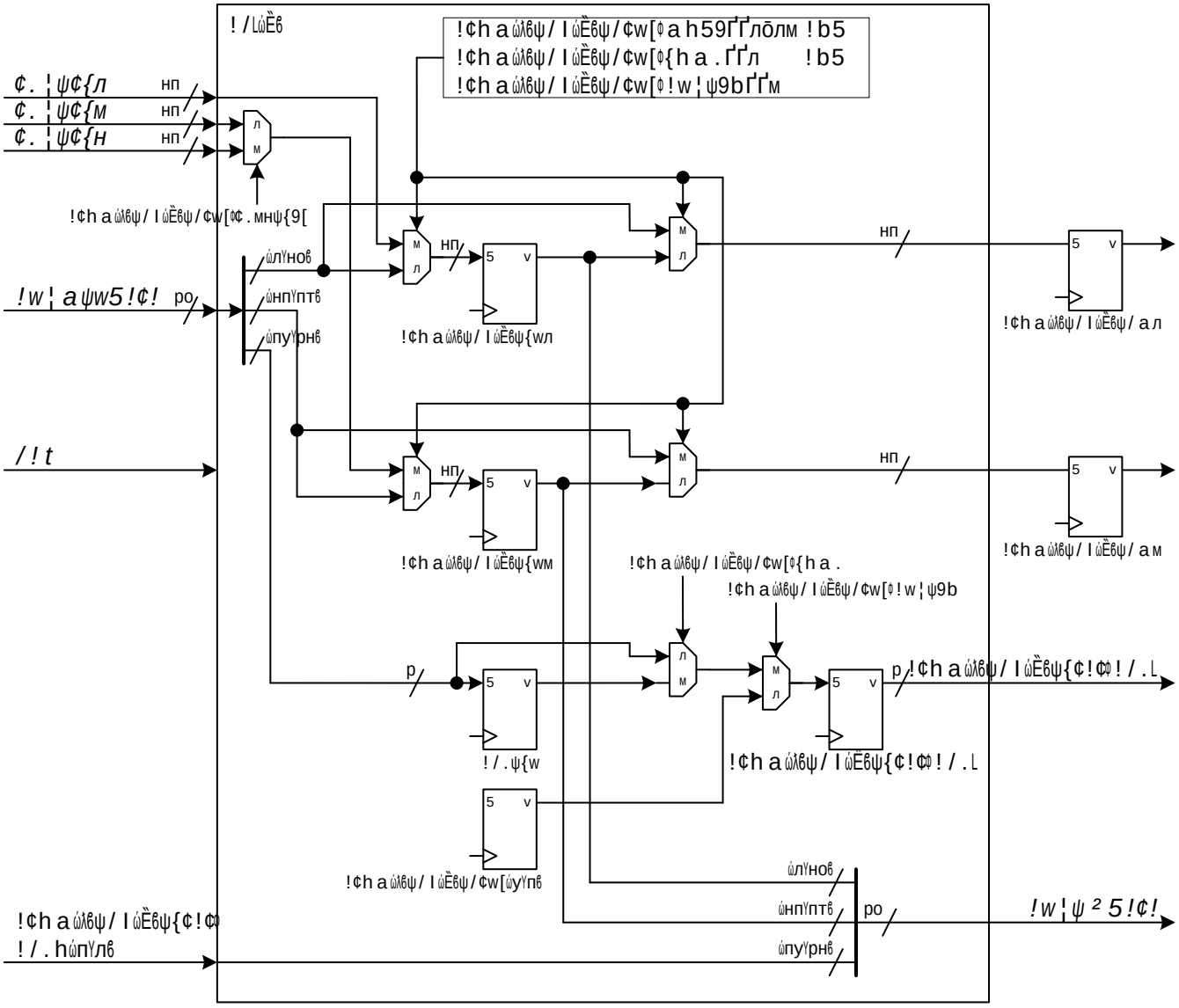
The ATOM channels have an ARU Communication Interface (ACI) sub-unit. This sub-unit is responsible for data exchange from and to ARU. This is done with the two implemented registers **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** , and the **ATOM[i]_CH[x]_STAT.ACBI** and **ATOM[i]_CH[x]_STAT.ACBO** bit fields that are part of the **ATOM[i]_CH[x]_STAT** register. The ACI architecture is shown in figure *83 "ACI architecture overview"* .

If the **ATOM[i]_CH[x]_CTRL.ARU_EN** bit is set, the ATOM channel is enabled by setting the enable bits **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[k]** and no data unequal to zero were written over configuration interface into the **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** register, the ATOM channel will first request data from the ARU before the signal generation starts in SOMP, SOMS, SOMC and SOMB mode.

In SOMP mode, if data in the **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_SR0.SR0** registers is not equal to 0, the channel counter **ATOM[i]_CH[x]_CN0.CN0** immediately starts counting, regardless of whether the channel has received ARU data yet.

In SOMS mode, if data in the **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_SR0.SR0** registers is not equal to 0, the channel immediately starts shifting, regardless of whether the channel has received ARU data yet.

Figure 83 ACI architecture overview



The incoming ARU data (53−bit wide signal *ARUM_RDATA*) is split into three parts by the ACI and they communicate to the ATOM channel registers. In SOMI, SOMP, SOMS and SOMB modes, the incoming ARU data *ARUM_RDATA* is split in such a way that the lower 24−bits *ARUM_RDATA* [23:0] of the ARU data are stored in the **ATOM[i]_CH[x]_SR0.SR0** register, the upper bits *ARUM_RDATA* [47:24] are stored in the **ATOM[i]_CH[x]_SR1.SR1** register. In SOMI, SOMP, SOMS and SOMC modes, the bits *ARUM_RDATA* [52:48] are stored in the **ATOM[i]_CH[x]_STAT.ACBI** bit field, while in SOMB mode these bits are stored in the internal **ACB_SR** register. In SOMB mode, if ARU communication is not enabled in a channel, **ATOM[i]_CH[x]_STAT.ACBI** of this channel will be updated with the ACB bits of **ATOM[i]_CH[x]_CTRL** [8:4].

In a case, when the channel operation registers **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** are updated with the contents of **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** and ARU transfer to these shadow registers happen in parallel, the ATOM channel has to ensure that either the old values from both shadow registers or both new values from the ARU are transferred to the operation registers.

In SOMC mode with enabled ARU communication, the incoming ARU data *ARUM_RDATA* is written directly to the ATOM channel operation register in such a way that the lower 24−bits [23:0] are written to **ATOM[i]_CH[x]_CM0.CM0** , and the upper bits [47:24] are written to register **ATOM[i]_CH[x]_CM1.CM1** . The bits [52:48] are stored in the **ATOM[i]_CH[x]_STAT.ACBI** bit field and control the behavior of the compare units and the output signal of the ATOM channel.

In SOMC mode, the **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** registers serve as capture registers for the time stamps coming from TBU whenever a compare match event is signaled by the CCU0 and/or CCU1 sub−units via the *CAP* signal line. These two time stamps are then provided together with the actual ATOM channel status information located in the **ATOM[i]_CH[x]_STAT.ACBO** bit field to the ARU at the dedicated ARU write address of the ATOM channel when the ARU is enabled. The data provided to ARU is marked as *ARU_WDATA* in the figure.

The encoding of the ARU control bits in the different ATOM operation modes is described in more detail in the following chapters.

15.2.2 External Trigger Interface

Each channel x of ATOM instance i has an input signal *EXT_TRIGIN* [x:x] that is connected to *TIM_EXT_CAPTURE* [x:x] of TIM instance i. Via this signal, events (high pulses with the length of 1 *CLS[i]_CLK* period) generated by the TIM module can be used to control the functionality of the ATOM channel in multiple ways:

- *EXT_TRIGIN* [x:x] events can be used to define the clock source of the selected channel mode (**ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** = 0b1110; **ATOM[i]_CH[x]_CTRL_SOMS.CLK_SRC** = 0b1110).
- *EXT_TRIGIN* [x:x] events can be used to update the operation registers with the content of the shadow registers in SOMP, SOMI, SOMS, SOMB mode (activated by **ATOM[i]_CH[x]_CTRL_SOMP.EXT_FUPD** = 1; **ATOM[i]_CH[x]_CTRL_SOMI.EXT_FUPD** = 1; **ATOM[i]_CH[x]_CTRL_SOMS.EXT_FUPD** = 1; **ATOM[i]_CH[x]_CTRL_SOMB.EXT_FUPD** = 1). Any *EXT_TRIGIN* [x:x] event is synchronized in the ATOM channel as specified in figure *82 "Timing diagram of RS unit"* .
- *EXT_TRIGIN* [x:x] events can be used to reset the counter register **ATOM[i]_CH[x]_CN0** in SOMP continuous counting up mode or change the counting direction in counting up-down mode or trigger the start of counting in one-shot mode (activated by **ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG** = 1 && **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** = 1). In this case, if **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 0b10 is configured, *EXT_TRIGIN* [x:x] events can also trigger the update of the operation registers from their shadow registers. Any *EXT_TRIGIN* [x:x] event is synchronized in the ATOM channel as specified in figure *82 "Timing diagram of RS unit"* .
- *EXT_TRIGIN* [x:x] can be used as a trigger event for the following channels (activated by **ATOM[i]_CH[x]_CTRL_SOMI.EXTTRIGOUT** = 1 && **ATOM[i]_CH[x]_CTRL_SOMI.TRIGOUT** = 0; **ATOM[i]_CH[x]_CTRL_SOMS.EXTTRIGOUT** = 1 && **ATOM[i]_CH[x]_CTRL_SOMS.TRIGOUT** = 0; **ATOM[i]_CH[x]_CTRL_SOMP.EXTTRIGOUT** = 1 && **ATOM[i]_CH[x]_CTRL_SOMP.TRIGOUT** = 0; **ATOM[i]_CH[x]_CTRL_SOMC.EXTTRIGOUT** = 1 && **ATOM[i]_CH[x]_CTRL_SOMC.TRIGOUT** = 0; **ATOM[i]_CH[x]_CTRL_SOMB.EXTTRIGOUT** = 1 && **ATOM[i]_CH[x]_CTRL_SOMB.TRIGOUT** = 0). Any *EXT_TRIGIN* [x:x] event is synchronized in the ATOM channel as specified in figure *82 "Timing diagram of RS unit"* .

In certain configurations of the TIM module, the *EXT_TRIGIN* [x:x] signal pulses can have a length of more than 1 cluster clock cycle. In this case, the functionality of the ATOM is not specified and it is highly recommended to prevent this configuration in applications.

Before triggering the functionalities mentioned above, the *EXT_TRIGIN* [x:x] signal is always synchronized with the current selected clock enable signal in RS unit as shown in figure *82 "Timing diagram of RS unit"* .

15.2.3 Internal Trigger Interface

As shown in figure *81 "ATOM channel architecture in SOMP mode"* , each ATOM channel x has an input signal *ATOM_CH_TRIGIN* [x:x] that is coming from *ATOM_CH_TRIGOUT* [x-1:x-1] from the preceding channel. Via this signal, events (high pulses with the length of 1 *CLS[i]_CLK* period) generated by the preceding ATOM channel, can be used to control the functionality of ATOM channel in multiple ways:

- *ATOM_CH_TRIGIN* [x:x] events can be used to define the clock source of the selected channel mode (**ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** = 0b1101; **ATOM[i]_CH[x]_CTRL_SOMS.CLK_SRC** = 0b1101).
- *ATOM_CH_TRIGIN* [x:x] events can be used to reset the counter register **ATOM[i]_CH[x]_CN0** in SOMP continuous counting up mode or change the counting direction in counting up-down mode or trigger the start of counting in one-shot mode (activated by **ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG** = 0 && **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** = 1). In this case, if **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[k]** = 0b10 is configured, *ATOM_CH_TRIGIN* [x:x] events can also trigger the update of the operation registers from their shadow registers.
- *ATOM_CH_TRIGIN* [x:x] can be used as a trigger event for following channels (activated by **ATOM[i]_CH[x]_CTRL_SOMI.EXTTRIGOUT** = 0 && **ATOM[i]_CH[x]_CTRL_SOMI.TRIGOUT** = 0; **ATOM[i]_CH[x]_CTRL_SOMS.EXTTRIGOUT** = 0 && **ATOM[i]_CH[x]_CTRL_SOMS.TRIGOUT** = 0; **ATOM[i]_CH[x]_CTRL_SOMP.EXTTRIGOUT** = 0 && **ATOM[i]_CH[x]_CTRL_SOMP.TRIGOUT** = 0; **ATOM[i]_CH[x]_CTRL_SOMC.EXTTRIGOUT** = 0 && **ATOM[i]_CH[x]_CTRL_SOMC.TRIGOUT** = 0; **ATOM[i]_CH[x]_CTRL_SOMB.EXTTRIGOUT** = 0 && **ATOM[i]_CH[x]_CTRL_SOMB.TRIGOUT** = 0).

Especially, for channel 0 of ATOM module i, i ∈ {1, 2, ..., NATOM-1}, *ATOM_CH_TRIGIN* [0:0] is connected to *ATOM_TRIGIN* of instance [i] that comes from *ATOM_CH_TRIGOUT* [7:7] of the preceding ATOM instance [i-1] and may be delayed by a register with the cluster clock *CLS[i]_CLK* in channel 0 before triggering the functionalities as mentioned above. This delay is controlled by the device configuration variable ATOM_TRIG_CHAIN (indicated by **CCM[i]_HW_CONF.ATOM_TRIG_CHAIN**). The delay exists after every N clusters (N = ATOM_TRIG_CHAIN). For ATOM0, *ATOM_CH_TRIGIN* [0:0] is connected to *ATOM_TRIGOUT_DEL* via the signal *ATOM_CH_TRIGOUT* [7:7] of ATOM0 itself as shown in figure *79 "ATOM block diagram"* . Furthermore, the cluster i can operate on faster GTM-IP clock *CLK* , if the device is defined with FAST_CLS_CLK[i] = 1. In this case, it is allowed to have one more register that is clocked with *CLK* to delay *ATOM_CH_TRIGIN* [4:4] in channel 4 before triggering the functionalities. This is controlled by another ATOM device configuration parameter ATOM_TRIG_INTCHAIN (it can only be valid if FAST_CLS_CLK[i] = 1 and indicated by **CCM[i]_HW_CONF.ATOM_TRIG_INTCHAIN**). The registers for these special cases are not shown in figure *81 "ATOM channel architecture in SOMP mode"* .

Unlike *EXT_TRIGIN* [x:x], *ATOM_CH_TRIGIN* [x:x] is not synchronized with the selected clock enable signal of channel x. For the application of 2 channels generating PWM signals synchronously, **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** of the triggering channel and the triggered channel should be configured with the same clock resolution and the same cluster clock.

15.3 ATOM Channel Modes

As described above, each ATOM channel can operate independently from each other in one of five dedicated output modes:

- ATOM Signal Output Mode Immediate (SOMI)
- ATOM Signal Output Mode Compare (SOMC)
- ATOM Signal Output Mode PWM (SOMP)

- ATOM Signal Output Mode Serial (SOMS)
- ATOM Signal Output Mode Buffered Compare (SOMB)

The Signal Output Mode PWM (SOMP) is principally the same as the output mode for the TOM submodule. In addition, it is possible to reload the shadow registers via the ARU without the need of a configuration interface interaction. The other modes provide additional functionality for signal output control. All operation modes are described in more detail in the following sections.

In any output mode, if a channel is enabled, one-shot mode is disabled (**ATOM[i]_CH[x]_CTRL.OSM** =0; only used in modes SOMP and SOMS) and **ATOM[i]_CH[x]_CM0.CM0** >= **ATOM[i]_CH[x]_CN0.CN0** , the counter **ATOM[i]_CH[x]_CN0.CN0** is incrementing until it reaches **ATOM[i]_CH[x]_CM0.CM0** .
To avoid unintended counting of **ATOM[i]_CH[x]_CN0.CN0** after enabling a channel, it is recommended to reset a channel (or at least **ATOM[i]_CH[x]_CN0.CN0** and **ATOM[i]_CH[x]_CM0.CM0**) before any change on the mode bits **ATOM[i]_CH[x]_CTRL.MODE** , **ATOM[i]_CH[x]_CTRL.ARU_EN** and **ATOM[i]_CH[x]_CTRL.OSM** .

15.3.1 ATOM Signal Output Mode Immediate (SOMI)

In ATOM Signal Output Mode Immediate (SOMI), the ATOM channel generates output signals on the *ATOM_OUT* of instance i and channel x output port immediately after the update of the bit **ATOM[i]_CH[x]_STAT.ACBI** [0:0] or **ATOM[i]_CH[x]_CTRL_SOMI.ACB0** bit.

If ARU access is enabled by setting bit **ATOM[i]_CH[x]_CTRL_SOMI.ARU_EN** , the update of the output *ATOM_OUT* of instance i and channel x depends on the bit **ATOM[i]_CH[x]_STAT.ACBI** [0:0] received at the ACI sub-unit and the bit **ATOM[i]_CH[x]_CTRL_SOMI.SL** . The remaining 48 ARU bits [47:0] have no meaning in this mode.

If ARU access is disabled, the update of the output *ATOM_OUT* of instance i and channel x depends on the bit **ATOM[i]_CH[x]_CTRL_SOMI.ACB0** and the bit **ATOM[i]_CH[x]_CTRL_SOMI.SL** .

The initial ATOM channel port pin *ATOM_OUT* signal level has to be specified by the **ATOM[i]_CH[x]_CTRL_SOMI.SL** bit when **ATOM[i]_AGC_OUTEN_CTRL.OUTEN_CTRL[k]** (x = k) is disabled (see chapter **ATOM[i]_AGC_OUTEN_CTRL**) for details.

In SOMI mode the output behavior depends on the **ATOM[i]_CH[x]_CTRL_SOMI.SL** bit and the bit **ATOM[i]_CH[x]_STAT.ACBI** or the bit **ATOM[i]_CH[x]_CTRL_SOMI.ACB0** :

Table 27 Output behavior in SOMI mode

ATOM[i]_CH[x]_CTRL_SOMI.SL	ATOM[i]_CH[x]_STAT.ACBI [0:0]/ ATOM[i]_CH[x]_CTRL_SOMI.ACB0	Output behavior
0	0	Set output to inverse of **ATOM[i]_CH[x]_CTRL_SOMI.SL** (1)
0	1	Set output to **ATOM[i]_CH[x]_CTRL_SOMI.SL** (0)
1	0	Set output to inverse of **ATOM[i]_CH[x]_CTRL_SOMI.SL** (0)
1	1	Set output to **ATOM[i]_CH[x]_CTRL_SOMI.SL** (1)

The signal level bit **ATOM[i]_CH[x]_STAT.ACBI** [0:0] is transferred to the SOU sub-unit of the ATOM and made visible at the output port according to the table above immediately after the data was received by the ACI. This can introduce a jitter on the output signal since the ARU channels are served in a time multiplexed fashion.

15.3.2 ATOM Signal Output Mode Compare (SOMC)

15.3.2.1 Overview

In ATOM Signal Output Mode Compare (SOMC) the output action is performed depending on the comparison between input values located in **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** registers and the two (three) time base values *CCM[i]_TBU_TS0* or *CCM[i]_TBU_TS1* (or *CCM[i]_TBU_TS2*) provided by the TBU. For a description of the time base generation please refer to the TBU specification in chapter *12 "Time Base Unit (TBU)"* . It is configurable, which of the two (three) time bases is to be compared with one or both values in **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** .

The behavior of the two compare units CCU0 and CCU1 is controlled either with the bits **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** when the ARU connection is disabled, or with the bit field **ATOM[i]_CH[x]_STAT.ACBI** [4:2] when the ARU is enabled. In the latter case the **ATOM[i]_CH[x]_STAT.ACBI** bit field is updated via the ARU control bits [52:48].

Based on the configured compare strategy **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** or **ATOM[i]_CH[x]_STAT.ACBI** [4:2], CCU0 unit always performs cyclic event compare between **ATOM[i]_CH[x]_CM0.CM0** and the selected TBU time base *CCM[i]_TBU_TS0* / *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* while CCU1 unit always performs cyclic event compare between **ATOM[i]_CH[x]_CM1.CM1** and the selected TBU time base *CCM[i]_TBU_TS0* / *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* . Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more information about cyclic event compare strategy. When the selected time base is "cyclically greater than or equal to" the compare value of **ATOM[i]_CH[x]_CM0.CM0** , CCU0 compare match event is considered to have occurred. When the selected time base is "cyclically greater than or equal to" the compare value of **ATOM[i]_CH[x]_CM1.CM1** , CCU1 compare match event is considered to have occurred. Whenever CCU0 or CCU1

compare match event has occurred, an edge is generated on output *ATOM_OUT* [x:x], depending on the predefined signal level in **ATOM[i]_CH[x]_CTRL_SOMC.SL** bit in combination with two control bits **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** or **ATOM[i]_CH[x]_STAT.ACBI** [1:0].

In SOMC mode the channel is always disabled after the specified compare match event has occurred. The shadow registers are used to store two time stamp values at the match time. The channel compare can be re-enabled by first reading the shadow registers, either over configuration interface or ARU and by providing new data for **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** registers through configuration interface or ARU. For a detailed description please refer to the sections *15.3.2.2 "SOMC Mode controlled by configuration interface"* and *15.3.2.3 "SOMC Mode under ARU control"* .

If three time bases exist for the GTM-IP there must be a preselection between *CCM[i]_TBU_TS1* and *CCM[i]_TBU_TS2* for the ATOM channel. This can be done with **ATOM[i]_CH[x]_CTRL_SOMC.TB12_SEL** bit.

If *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* is selected for cyclic event compare in CCU0 or CCU1 unit, **ATOM[i]_CH[x]_CTRL_SOMC.CMP_CTRL** must be configured to assure correct cyclic event compare. If the selected TBU time base *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* performs up-counting, **ATOM[i]_CH[x]_CTRL_SOMC.CMP_CTRL** must be configured as 0. If the selected TBU time base *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* performs down-counting, **ATOM[i]_CH[x]_CTRL_SOMC.CMP_CTRL** must be configured as 1. The bit **ATOM[i]_CH[x]_CTRL_SOMC.CMP_CTRL** has no impact on cyclic event compare against the TBU time base *CCM[i]_TBU_TS0* .

When configured in SOMC mode, the channel port pin has to be initialized to an initial signal level. This initial level after enabling the ATOM channel is determined by the **ATOM[i]_CH[x]_CTRL_SOMC.SL** bit. If the output is disabled, the signal level is set to the inverse level of the **ATOM[i]_CH[x]_CTRL_SOMC.SL** bit.

If the channel is disabled and the output is still enabled with **ATOM[i]_CH[x]_CTRL_SOMC.FREEZE** =0, the register SOUR and the output *ATOM_OUT* [x:x]is set to the **ATOM[i]_CH[x]_CTRL_SOMC.SL** bit.

On a compare match event the shadow register **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are used to capture the TBU time stamp values. **ATOM[i]_CH[x]_SR0.SR0** always holds *CCM[i]_TBU_TS0* and **ATOM[i]_CH[x]_SR1.SR1** either holds *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* depending on the **ATOM[i]_CH[x]_CTRL_SOMC.TB12_SEL** bit.

When the channel is disabled and the compare registers are written, the compare registers **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** are loaded with the written value and the channel starts with the comparison against these values, when the channel is enabled.

15.3.2.2 SOMC Mode controlled by configuration interface

As already mentioned above the ATOM channel can be controlled either from configuration interface or by ARU. When the channel should be controlled via the configuration interface, the **ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN** bit has to be reset.

The output of the ATOM channel is set at a compare match event depending on the **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** in combination with the **ATOM[i]_CH[x]_CTRL_SOMC.SL** bit. The comparison is performed according to different compare strategy specified by **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** . The output behavior according to **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** bit field in the control register is shown in the following table:

Table 28 Output behavior according to the ATOM[i]_CH[x]_CTRL_SOMC.ACB10 bit field in the control register

ATOM[i]_CH[x]_CTRL.SL	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [1:1]	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [0:0]	Output behavior
0	0	0	No signal level change at output (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001)
0	0	1	Set output signal level to 1 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001)
0	1	0	Set output signal level to 0 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001)
0	1	1	Toggle output signal level (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001)

ATOM[i]_CH[x]_CTRL.SL	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [1:1]	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [0:0]	Output behavior
1	0	0	No signal level change at output (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001)
1	0	1	Set output signal level to 0 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001)
1	1	0	Set output signal level to 1 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001)
1	1	1	Toggle output signal level (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** = 001)

The capture/compare strategy of the two units CCU0 and CCU1 can be controlled with the **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** bit field. The CCU0 unit always compares **ATOM[i]_CH[x]_CTRL.CM0** with the selected TBU time base. The CCU1 unit always compares **ATOM[i]_CH[x]_CTRL.CM1** with the selected TBU time base. Different CCU0/CCU1 comparison strategies controlled by **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** are shown in the following table:

Table 29 Capture/compare strategy of the two CCUx units controlled by ATOM[i]_CH[x]_CTRL_SOMC.ACB42 bit field

ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [2:2]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [1:1]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [0:0]	CCU0/CCU1 comparison control
0	0	0	Serve First: Compare in CCU0 using *CCM[i]_TBU_TS0* and in parallel in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Disable the other comparison at one compare match event. Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** . Details see table *30 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b000 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b000"* .
0	0	1	Serve First: Compare in CCU0 using *CCM[i]_TBU_TS0* and in parallel in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Disable the other comparison at one compare match event. Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** . Details see table *31 "ATOM Serve first definition ATOM[i]_CH[x]_C-* ▽

ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [2:2]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [1:1]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [0:0]	CCU0/CCU1 comparison control
			△ *TRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* .
0	1	0	Compare in CCU0 only, use time base *CCM[i]_TBU_TS0* . Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** (details see table *28 "Output behavior according to the ATOM[i]_CH[x]_CTRL_SOMC.ACB10 bit field in the control register"*).
0	1	1	Compare in CCU1 only, use time base *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** (details see table *28 "Output behavior according to the ATOM[i]_CH[x]_CTRL_SOMC.ACB10 bit field in the control register"*).
1	0	0	Serve Last: Compare in CCU0 and then in CCU1 using *CCM[i]_TBU_TS0* . Output signal level at the CCU0 compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** (details see table *28 "Output behavior according to the ATOM[i]_CH[x]_CTRL_SOMC.ACB10 bit field in the control register"*). At the CCU1 compare match event the output level is toggled.
1	0	1	Serve Last: Compare in CCU0 and then in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level at the CCU0 compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** (details see table *28 "Output behavior according to the ATOM[i]_CH[x]_CTRL_SOMC.ACB10 bit field in the control register"*). At the CCU1 compare match event the output level is toggled.
1	1	0	Serve Last: Compare in CCU0 using *CCM[i]_TBU_TS0* and then in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level at the CCU1 compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** (details see table *28 "Output behavior according to the ATOM[i]_CH[x]_CTRL_SOMC.ACB10 bit field in the control register"*).

ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [2:2]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [1:1]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [0:0]	CCU0/CCU1 comparison control
1	1	1	Cancels pending comparison independent on **ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN** .

The behavior of the **ATOM[i]_CH[x]_STAT.ACBI** [4:2]/ **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** bit combinations 0b000 and 0b001 is described in more detail in tables *30 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b000 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b000"* and *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* .

Table 30 ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b000 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b000

ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [2:2] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [4:4]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [1:1] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [3:3]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [0:0] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [2:2]	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [1:1] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [1:1]	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [0:0] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [0:0]	ATOM[i]_CH[x]_CTRL_SOMC.SL	CCU0 match	CCU1 match	Pin level new
0	0	0	0	0	0	0	1	hold
0	0	0	0	0	0	1	0	hold
0	0	0	0	0	0	1	1	hold
0	0	0	0	1	0	0	1	1
0	0	0	0	1	0	1	0	1
0	0	0	0	1	0	1	1	1
0	0	0	1	0	0	0	1	0
0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	1	1	0
0	0	0	1	1	0	0	1	toggle
0	0	0	1	1	0	1	0	toggle
0	0	0	1	1	0	1	1	toggle
0	0	0	0	0	1	0	1	hold
0	0	0	0	0	1	1	0	hold
0	0	0	0	0	1	1	1	hold
0	0	0	0	1	1	0	1	0
0	0	0	0	1	1	1	0	0
0	0	0	0	1	1	1	1	0
0	0	0	1	0	1	0	1	1
0	0	0	1	0	1	1	0	1
0	0	0	1	0	1	1	1	1
0	0	0	1	1	1	0	1	toggle
0	0	0	1	1	1	1	0	toggle
0	0	0	1	1	1	1	1	toggle

Table 31 ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001

ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [2:2] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [4:4]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [1:1] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [3:3]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [0:0] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [2:2]	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [1:1] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [1:1]	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [0:0] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [0:0]	ATOM[i]_CH[x]_CTRL_SOMC.SL	CCU0 match	CCU1 match	Pin level new
0	0	1	0	0	0	0	1	hold
0	0	1	0	0	0	1	0	toggle
0	0	1	0	0	0	1	1	hold
0	0	1	0	1	0	0	1	0
0	0	1	0	1	0	1	0	1

BOSCH

ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [2:2] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [4:4]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [1:1] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [3:3]	ATOM[i]_CH[x]_CTRL_SOMC.ACB42 [0:0] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [2:2]	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [1:1] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [1:1]	ATOM[i]_CH[x]_CTRL_SOMC.ACB10 [0:0] / ATOM[i]_CH[x]_CTRL_STAT.ACBI [0:0]	ATOM[i]_CH[x]_CTRL_SOMC.SL	CCU0 match	CCU1 match	Pin level new
0	0	1	0	1	0	1	1	0
0	0	1	1	0	0	0	1	1
0	0	1	1	0	0	1	0	0
0	0	1	1	0	0	1	1	1
0	0	1	1	1	0	0	1	toggle
0	0	1	1	1	0	1	0	hold
0	0	1	1	1	0	1	1	toggle
0	0	1	0	0	1	0	1	hold
0	0	1	0	0	1	1	0	toggle
0	0	1	0	0	1	1	1	hold
0	0	1	0	1	1	0	1	1
0	0	1	0	1	1	1	0	0
0	0	1	0	1	1	1	1	1
0	0	1	1	0	1	0	1	0
0	0	1	1	0	1	1	0	1
0	0	1	1	0	1	1	1	0
0	0	1	1	1	1	0	1	toggle
0	0	1	1	1	1	1	0	hold
0	0	1	1	1	1	1	1	toggle

If the ATOM channel is enabled, the **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** registers and the **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** bit field can be updated over the configuration interface as long as the first match event occurs in case of a 'serve first' compare strategy or as long as the overall match event occurs in case of the other compare strategies.

After a compare match event that causes an update of the shadow registers **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** and before reading the **ATOM[i]_CH[x]_SR0.SR0** and/or **ATOM[i]_CH[x]_SR1.SR1** register via ARU or the configuration interface, the update of the registers **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** is possible but has no effect.

To set up a new compare action, first the **ATOM[i]_CH[x]_SR0.SR0** and/or **ATOM[i]_CH[x]_SR1.SR1** register containing captured values have to be read and then new compare values have to be written into the register **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** .

Which register **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** has to be updated depends on the compare strategy defined in the **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** bit field of the channel control register. Since the channel immediately starts with the comparison after the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register was/were written, the compare strategy has to be updated before the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** registers are written.

For the 'serve last' compare strategies, if the register **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** are updated, it can happen that one or both compare values are already located in the past. In any case, the ATOM channel will first wait until both compare values are written before it starts the time base comparisons to avoid a deadlock.

The configuration interface can be used to check at any time, whether at least one of the capture compare registers of the ATOM channel contains valid data, and to wait for a compare event to occur. This is signaled by the **ATOM[i]_CH[x]_STAT.DV** bit inside the **ATOM[i]_CH[x]_STAT** register.

Note:

For 'serve last' compare strategies, if **ATOM[i]_CH[x]_STAT.DV** bit is currently not set, writing to **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** immediately sets the **ATOM[i]_CH[x]_STAT.DV** bit although the compare is only started if both values are written.

An exception for update of register **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** exists in SOMC mode and CCUx control mode 'serve last'. If in this mode the CCU0 compare match event occurred, the update of register **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** over the configuration interface is blocked until the CCU1 compare match event.

In the 'serve last' mode (**ATOM[i]_CH[x]_CTRL_SOMC.ACB42** = 0b100 or **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =0b101) it is possible to generate very small spikes on the output pin by loading **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** with two time stamp values for *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* close together. The output pin will then be set or reset depending on the **ATOM[i]_CH[x]_CTRL_SOMC.SL** bit and **ATOM[i]_CH[x]_CTRL_SOMC.ACB10** bit field on the first match event and the output will toggle on the second compare event in the CCU1 compare unit.

Note:

Since the CCU0 will enable the CCU1 once it reaches its comparison time stamp, the bigger (smaller) time stamp has to be loaded into the **ATOM[i]_CH[x]_CM1.CM1** register. The order of the comparison time stamps depends on the defined greater-equal or less-equal comparison of the CCU0/CCU1 units.

In addition to storing the captured time stamps in the shadow registers, the ATOM channel provides the result of the compare match event in the **ATOM[i]_CH[x]_STAT.ACBO** [4:4] and **ATOM[i]_CH[x]_STAT.ACBO** [3:3] bits. The meaning of the bits is shown in the following table:

Table 32 Compare match event ATOM[i]_CH[x]_STAT.ACBO[4:4] and ATOM[i]_CH[x]_STAT.ACBO[3:3] bits of

ATOM[i]_CH[x]_STAT.ACBO [4:4]	ATOM[i]_CH[x]_STAT.ACBO [3:3]	Indication
0	1	CCU0 compare match occurred
1	0	CCU1 compare match occurred

Note:

In case of the 'serve last' compare strategy, if the bit **ATOM[i]_CH[x]_CTRL_SOMC.SLA** is 0, the **ATOM[i]_CH[x]_STAT.ACBO** [4:3] will be always 0b10 after the CCU1 compare match event and provided with the captured data in the shadow registers to ARU.

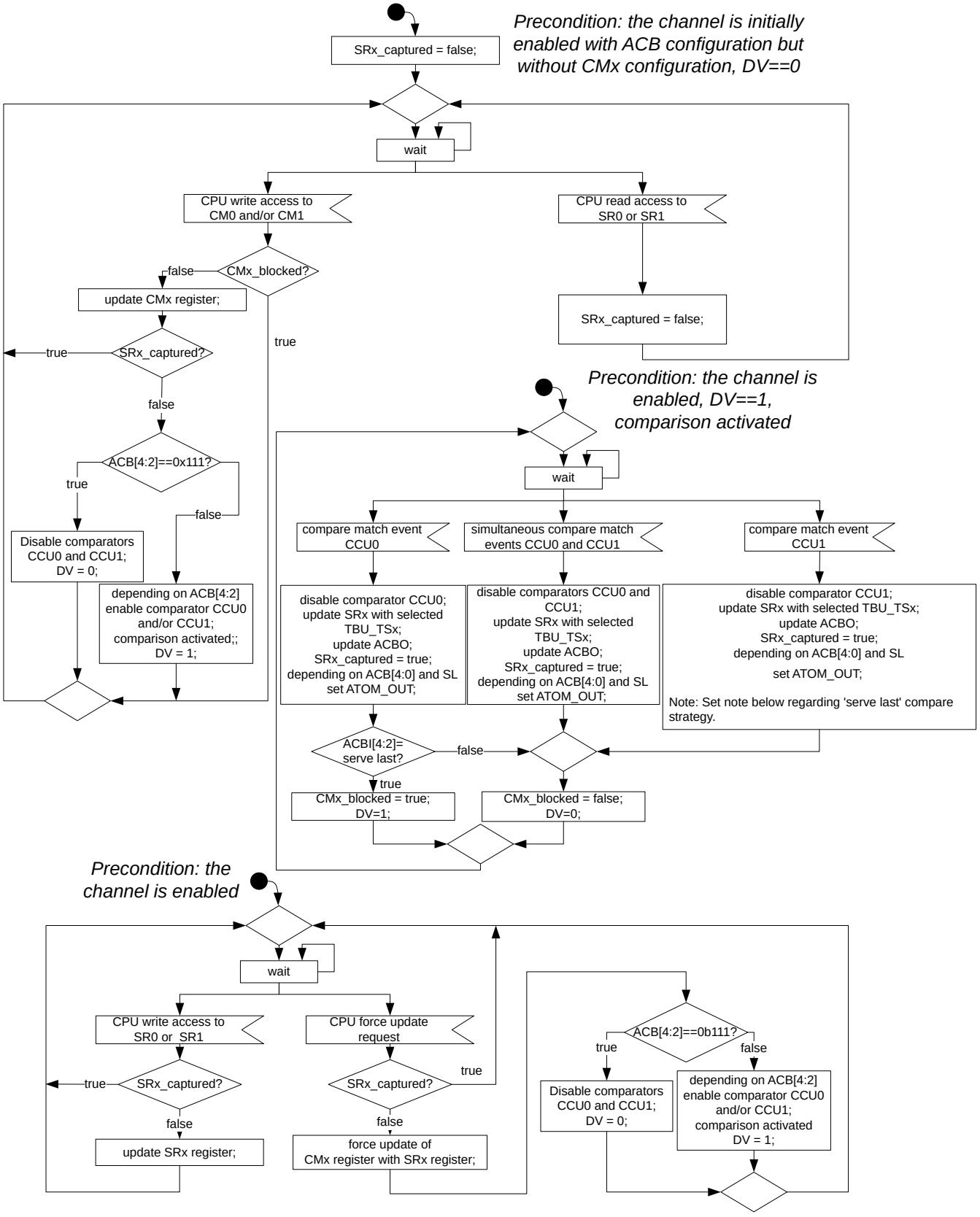
The **ATOM[i]_CH[x]_STAT.ACBO** bit field is reset, when the **ATOM[i]_CH[x]_STAT.DV** bit is set.

Depending on the capture compare unit where the compare match event occurred, the interrupt *ATOM_CCU0TC[x]_IRQ* or *ATOM_CCU1TC[x]_IRQ* is raised.

In case of 'serve first' compare strategy, if both events CCU0 and CCU1 occur at the same point in time, both interrupts will be raised and **ATOM[i]_CH[x]_STAT.ACBO** will be 0b10 due to higher priority.

The behavior of an ATOM channel in SOMC mode with **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =0 controlled via configuration interface is depicted in figure *84 "SOMC state diagram for channel controlled via configuration interface if ATOM[i]_CH[x]_CTRL_SOMC.EUPM=0"* .

Figure 84 SOMC state diagram for channel controlled via configuration interface if ATOM[i]_CH[x]_CTRL_SOMC.EUPM=0



Note:

In case of 'serve last' compare strategy the compare match in CCU0 is expected before the compare match in CCU1. But if the CCU1 compare match occurs first, **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** as well as **ATOM[i]_CH[x]_STAT.ACBO** will not be updated and also the output signal will not be set. After first occurance of CCU0 match and afterwards a CCU1 match the ordering is as expected and captures and signal level output changes will be performed on both matches.

BOSCH

15.3.2.3 SOMC Mode under ARU control

When the channel should be controlled by ARU, **ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN** has to be set to 1.

In case, the ATOM channel is under ARU control, the content for the compare registers **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** as well as the update of the compare strategy can be loaded via the 53-bit ARU word.

The ARU word [23:0] is loaded into the **ATOM[i]_CH[x]_CM0.CM0** register while the ARU word [47:24] is loaded into the **ATOM[i]_CH[x]_CM1.CM1** register. The five ARU control bits [52:48] are loaded into the **ATOM[i]_CH[x]_STAT.ACBI** bit field and control the channel compare strategy as well as the output behavior in case of compare match events.

For the five ARU control bits [52:48] the bits [49:49] and [48:48] are loaded into **ATOM[i]_CH[x]_STAT.ACBI** [1:0]. The output of the ATOM channel is set at a compare match event depending on **ATOM[i]_CH[x]_STAT.ACBI** [1:0] in combination with **ATOM[i]_CH[x]_CTRL_SOMC.SL** bit. The comparison is performed according to different compare strategy specified by **ATOM[i]_CH[x]_STAT.ACBI** [4:2]. The output behavior according to **ATOM[i]_CH[x]_STAT.ACBI** [1:0] bit field is shown in the following table:

Table 33 Output behavior depends on ATOM[i]_CH[x]_CTRL_SOMC.SL bit and ATOM[i]_CH[x]_STAT.ACBI bits 1 and 0

SL	ATOM[i]_CH[x]_STAT.ACBI [1:1]	ATOM[i]_CH[x]_STAT.ACBI [0:0]	Output behavior
0	0	0	No signal level change at output (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_STAT.ACBI** [4:2]=001)
0	0	1	Set output signal level to 1 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_STAT.ACBI** [4:2]=001)
0	1	0	Set output signal level to 0 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_STAT.ACBI** [4:2]=001)
0	1	1	Toggle output signal level (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_STAT.ACBI** [4:2]=001)
1	0	0	No signal level change at output (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_STAT.ACBI** [4:2]=001)
1	0	1	Set output signal level to 0 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_STAT.ACBI** [4:2]=001)
1	1	0	Set output signal level to 1 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_STAT.ACBI** [4:2]=001)

SL	ATOM[i]_CH[x]_STAT.ACBI [1:1]	ATOM[i]_CH[x]_STAT.ACBI [0:0]	Output behavior
1	1	1	Toggle output signal level (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_STAT.ACBI** [4:2]=001)

For the five ARU control bits [52:50] are loaded into the **ATOM[i]_CH[x]_STAT.ACBI** [4:2] bits. The capture/compare strategy of the two units CCU0 and CCU1 can be controlled with **ATOM[i]_CH[x]_STAT.ACBI** [4:2]. The CCU0 unit always compares **ATOM[i]_CH[x]_CTRL.CM0** with the selected TBU time base. The CCU1 unit always compares **ATOM[i]_CH[x]_CTRL.CM1** with the selected TBU time base. Different CCU0/CCU1 comparison strategies controlled by **ATOM[i]_CH[x]_STAT.ACBI** [4:2] are shown in the following table:

Table 34 Capture/compare units CCUx controlled by ATOM[i]_CH[x]_STAT.ACBI[4:2] bits

ATOM[i]_CH[x]_STAT.ACBI [4:4]	ATOM[i]_CH[x]_STAT.ACBI [3:3]	ATOM[i]_CH[x]_STAT.ACBI [2:2]	CCU0/CCU1 comparison control
0	0	0	Serve First: Compare in CCU0 using *CCM[i]_TBU_TS0* and in parallel in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Disable the other comparison at one compare match event. Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0]. Details see table *30 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b000 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b000"*
0	0	1	Serve First: Compare in CCU0 using *CCM[i]_TBU_TS0* and in parallel in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Disable the other comparison at one compare match event. Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0]. Details see table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"*
0	1	0	Compare in CCU0 only, use time base *CCM[i]_TBU_TS0* . Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0] (details in table *33 "Output behavior depends on ATOM[i]_CH[x]_CTRL_SOMC.SL bit and ATOM[i]_CH[x]_STAT.ACBI bits 1 and 0"*).
0	1	1	Compare in CCU1 only, use time base *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0] (details in table *33 "Output behavior depends on ATOM[i]_CH[x]_CTRL_SOMC.SL bit and ATOM[i]_CH[x]_STAT.ACBI bits 1 and 0"*).

ATOM[i]_CH[x]_STAT.ACBI [4:4]	ATOM[i]_CH[x]_STAT.ACBI [3:3]	ATOM[i]_CH[x]_STAT.ACBI [2:2]	CCU0/CCU1 comparison control
1	0	0	Serve Last: Compare in CCU0 and then in CCU1 using *CCM[i]_TBU_TS0* . Output signal level at the CCU0 compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0] (details in table *33 "Output behavior depends on ATOM[i]_CH[x]_CTRL_SOMC.SL bit and ATOM[i]_CH[x]_STAT.ACBI bits 1 and 0"*). At the CCU1 compare match event the output level is toggled.
1	0	1	Serve Last: Compare in CCU0 and then in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level at the CCU0 compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0] (details in table *33 "Output behavior depends on ATOM[i]_CH[x]_CTRL_SOMC.SL bit and ATOM[i]_CH[x]_STAT.ACBI bits 1 and 0"*). At the CCU1 compare match event the output level is toggled.
1	1	0	Serve Last: Compare in CCU0 using *CCM[i]_TBU_TS0* and then in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level at the CCU1 match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMC.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0] (details in table *33 "Output behavior depends on ATOM[i]_CH[x]_CTRL_SOMC.SL bit and ATOM[i]_CH[x]_STAT.ACBI bits 1 and 0"*).
1	1	1	Change ARU read address to **ATOM[i]_CH[x]_RDADDR.RDADDR1** . **ATOM[i]_CH[x]_STAT.DV** flag is not set. Neither **ATOM[i]_CH[x]_STAT.ACBI** [1:1] nor **ATOM[i]_CH[x]_STAT.ACBI** [0:0] is evaluated.

The bit combination 0b111 for the **ATOM[i]_CH[x]_STAT.ACBI** [4:4], **ATOM[i]_CH[x]_STAT.ACBI** [3:3] and **ATOM[i]_CH[x]_STAT.ACBI** [2:2] bits force the channel to request new compare values from another destination read address defined in the **ATOM[i]_CH[x]_RDADDR.RDADDR1** bit field. After data is successfully received and the compare event occurs, the ATOM channel will switch back to **ATOM[i]_CH[x]_RDADDR.RDADDR0** to receive the next data from there.

After the specified compare match event, the captured time stamps are stored in **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** and the compare result is stored in the **ATOM[i]_CH[x]_STAT.ACBO** bit field. The meaning of the **ATOM[i]_CH[x]_STAT.ACBO** [4:4] and **ATOM[i]_CH[x]_STAT.ACBO** [3:3] bits are shown in the following table:

Table 35 Compare match event ATOM[i]_CH[x]_STAT.ACBO[4:4] and ATOM[i]_CH[x]_STAT.ACBO[3:3] bits of ATOM[i]_CH[x]_STAT

ATOM[i]_CH[x]_STAT.ACBO [4:4]	ATOM[i]_CH[x]_STAT.ACBO [3:3]	Return value to ARU
0	1	CCU0 compare match occurred
1	0	CCU1 compare match occurred

In case of the 'serve last' compare strategy, if the bit **ATOM[i]_CH[x]_CTRL_SOMC.SLA** is 0, the **ATOM[i]_CH[x]_STAT.ACBO** [4:3] will be always 0b10 after the CCU1 compare match event and provided with the captured data in the shadow registers to ARU.

The **ATOM[i]_CH[x]_STAT.ACBO** bit field is reset, when the **ATOM[i]_CH[x]_STAT.DV** bit is set.

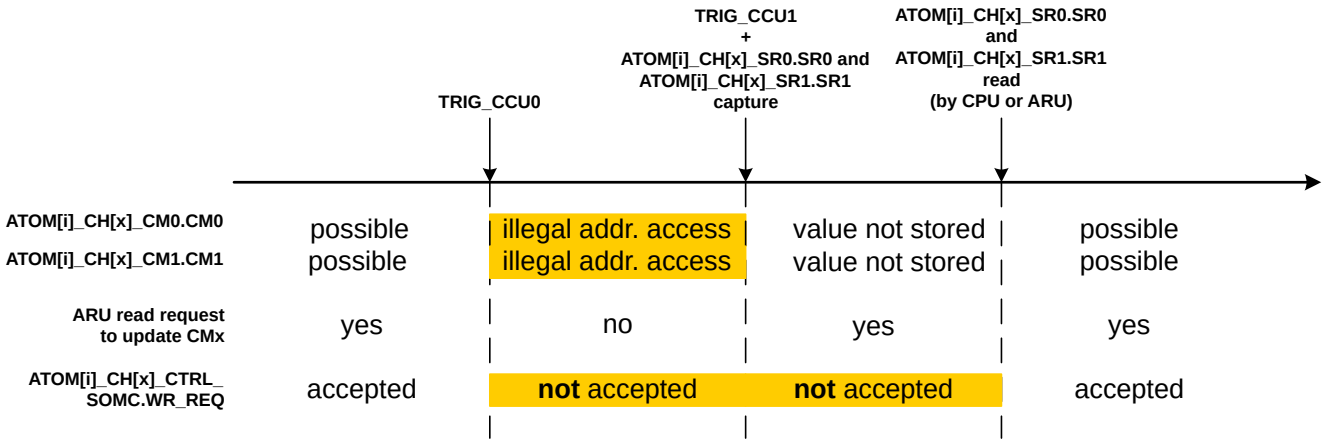
Depending on the capture compare unit where the compare match event occurred, the interrupt *ATOM_CCU0TC[x]_IRQ* or *ATOM_CCU1TC[x]_IRQ* is raised.

When CCU0 and CCU1 are used for comparison it is possible to generate very small spikes on the output pin by loading **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** with two time stamp values for *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* close together. The output pin will then be set or reset depending on the **ATOM[i]_CH[x]_CTRL_SOMC.SL** bit and the specified **ATOM[i]_CH[x]_STAT.ACBI** [1:0] bits at the first match event and the output will toggle at the second match event.

Since the CCU0 will enable the CCU1 once it reaches its comparison time stamp, the bigger (smaller) time stamp has to be loaded into the **ATOM[i]_CH[x]_CM1.CM1** register. The order of the comparison time stamps depends on the defined greater-equal or less-equal comparison of the CCU0/CCU1 units.

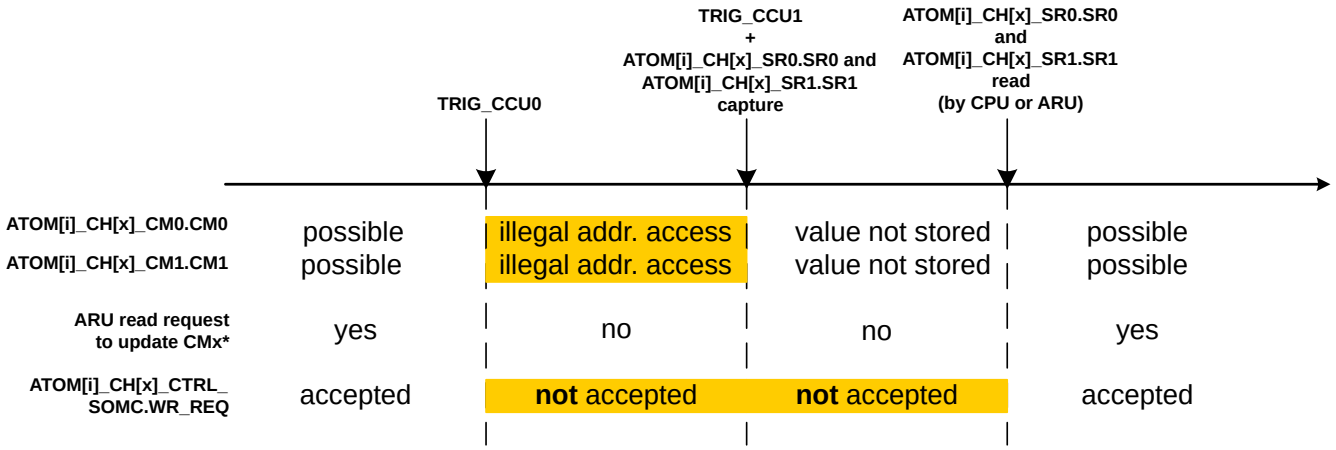
For compare strategy 'serve last' the CCU0 and CCU1 compare match may occur sequentially. During different phases of compare match the configuration interface access rights to registers **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** as well as to **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit are different.
For the case of bit **ATOM[i]_CH[x]_CTRL_SOMC.ABM** =0 and **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =0 these access rights over the configuration interface to register **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** and the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** are depicted in the following figure.

Figure 85 Configuration interface access rights in case of compare strategy 'serve last', ATOM[i]_CH[x]_CTRL_SOMC.ABM=0 and ATOM[i]_CH[x]_CTRL_SOMC.EUPM=0



For the bits **ATOM[i]_CH[x]_CTRL_SOMC.ABM** =1 and **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =0, these access rights from configuration interface to registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** are depicted in the following figure.

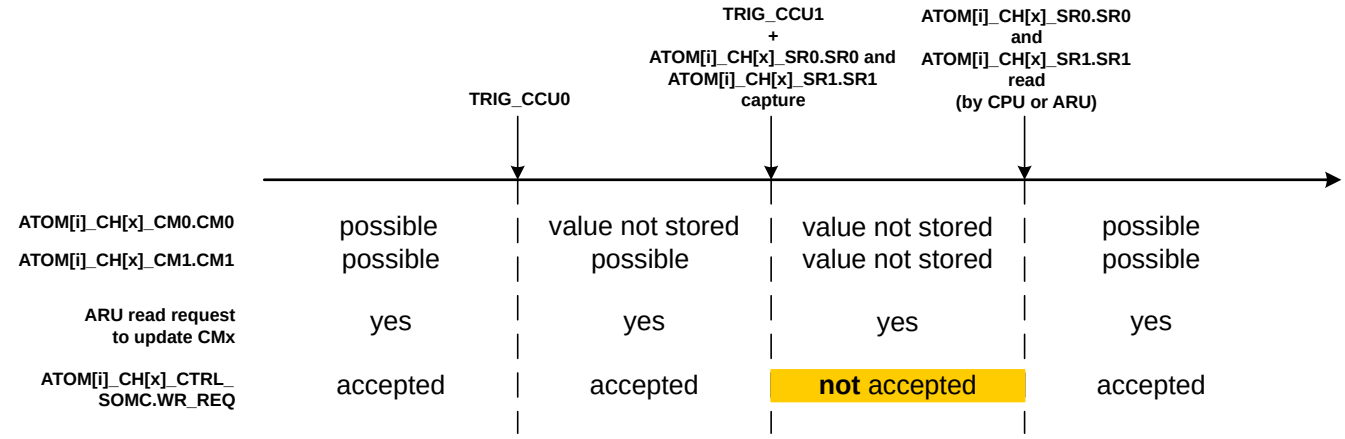
Figure 86 Configuration interface access rights in case of compare strategy 'serve last', ATOM[i]_CH[x]_CTRL_SOMC.ABM=1 and ATOM[i]_CH[x]_CTRL_SOMC.EUPM=0



* ARU read request until valid data is stored in **ATOM[i]_CH[x]_CM0.CM0/ ATOM[i]_CH[x]_CM1.CM1** (i.e. indicated by DV bit in register ATOM[i]_CH[x]_STAT set)

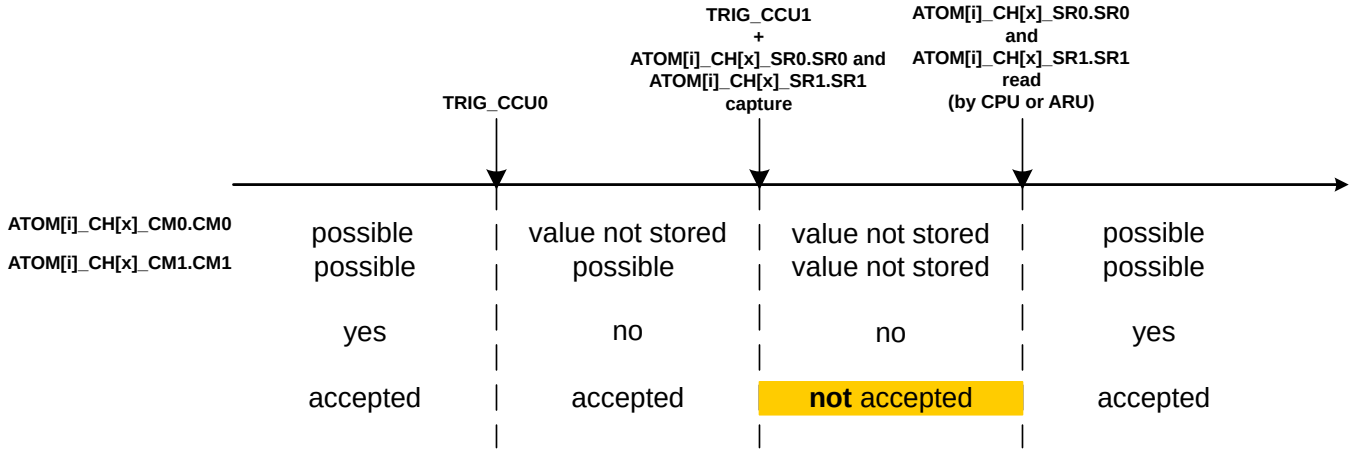
In case bit **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =1, after CCU0 compare match (and before CCU1 compare match) an update of **ATOM[i]_CH[x]_CM1.CM1** as well as a late update via **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** is possible. The value is used for compare. After CCU0 compare match an update of **ATOM[i]_CH[x]_CM0.CM0** is not possible, that means the value is not stored.
The ARU read request is not paused between the compare matches.
This behavior is depicted in the following figures.

Figure 87 Configuration interface access rights in case of compare strategy 'serve last', ATOM[i]_CH[x]_CTRL_SOMC.ABM=0 and ATOM[i]_CH[x]_CTRL_SOMC.EUPM=1



The behavior in case of EUPM=1 and ABM=1 is depicted in the following figure:

Figure 88 Configuration interface access rights in case of compare strategy 'serve last', ATOM[i]_CH[x]_CTRL_SOMC.ABM=1 and ATOM[i]_CH[x]_CTRL_SOMC.EUPM=1



* ARU read request until valid data is stored in **ATOM[i]_CH[x]_CM0.CM0/ATOM[i]_CH[x]_CM1.CM1** (i.e. indicated by DV bit in register ATOM[i]_CH[x]_STAT set)

In case of **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =1 a write access to **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** never causes an AEI write status 0b10.

15.3.2.3.1 ARU Non–Blocking mode

When the compare registers are updated via ARU, the update behavior of the channel is configurable with the ABM bit inside the **ATOM[i]_CH[x]_CTRL_SOMC** register. When the **ATOM[i]_CH[x]_CTRL_SOMC.ABM** bit is reset, the ATOM channel is in ARU non–blocking mode.

In the ARU non–blocking mode, the ARU read request raised by the channel is always 1 and so ARU data can be continuously transferred to the registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and the bit field **ATOM[i]_CH[x]_STAT.ACBI** as long as no specified compare match event occurs.

When the specified compare match event occurs, the shadow registers **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are updated together with the **ATOM[i]_CH[x]_STAT.ACBO** bits. The data in the shadow registers holding the captured TBU time stamp values is marked as valid for the ARU and the **ATOM[i]_CH[x]_STAT.DV** bit is reset. After the specified compare match event occurs but before the shadow registers are read via configuration interface or ARU, the registers **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** can be continuously updated via configuration interface or ARU but the data is not accepted to be valid (**ATOM[i]_CH[x]_STAT.DV** = 0).

To set up a new compare action, firstly, the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register containing captured values have to be read and then new compare values have to be written into the register **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** . This can be done either by ARU or over configuration interface.

When the registers are accessed through the configuration interface, only one of the shadow registers must be read. Depending on the compare strategy, one or both of the compare registers has to be written via the configuration interface or ARU interface to restart a new comparison.

An exception for update of register **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** in SOMC mode is CCUx control mode 'serve last' if **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =0. In this mode, if the CCU0 compare match event occurs, the update of register **ATOM[i]_-**

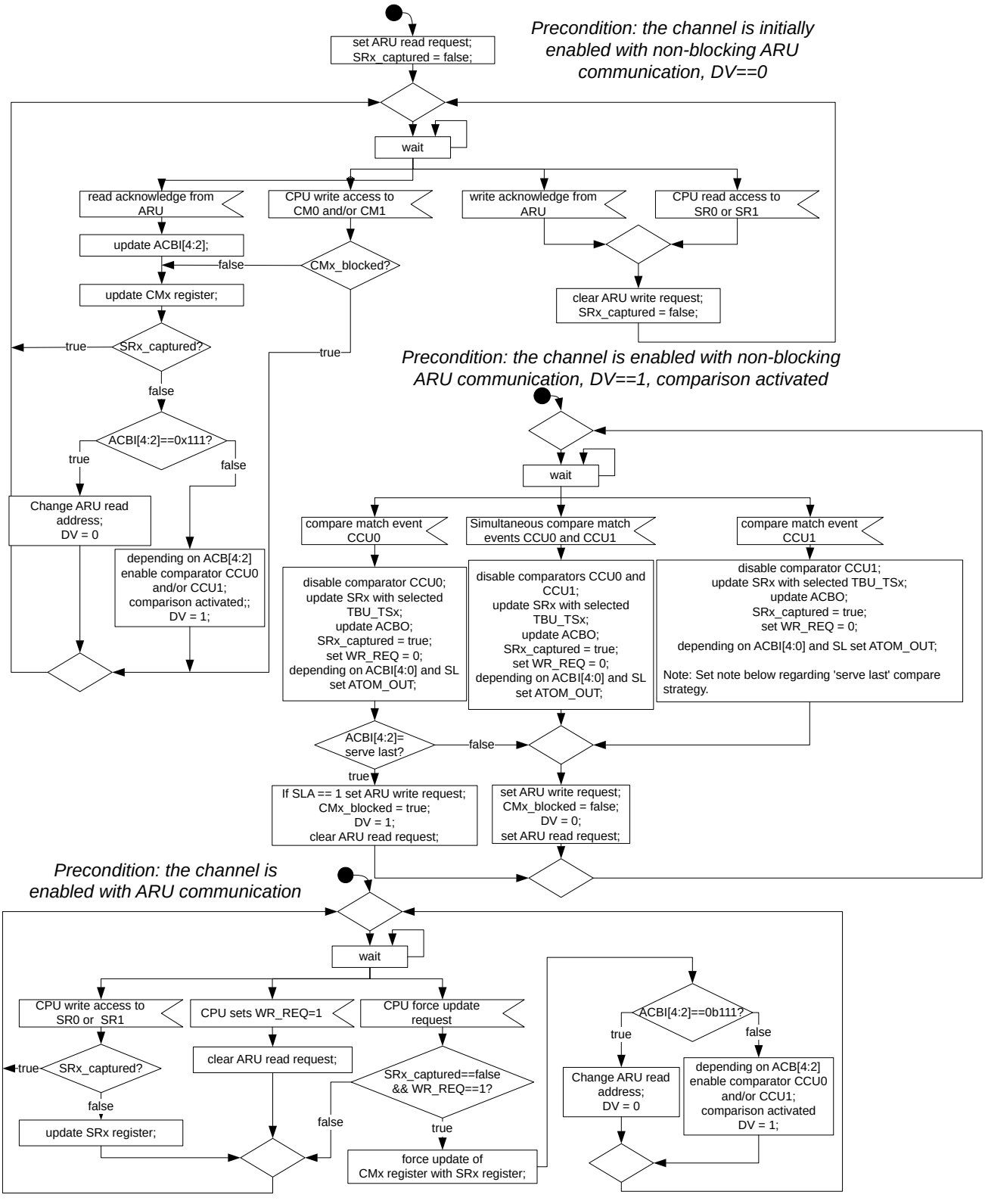
CH[x]_CM0.CM0 / **ATOM[i]_CH[x]_CM1.CM1** over configuration interface or ARU is not possible until the CCU1 compare match event occurs.

A write access to either **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** in this case leads to a write status 0b10.

It can be checked through the configuration interface at any time, if the ATOM channel has received valid data from the ARU and waits for a compare event to occur. This is signaled by the **ATOM[i]_CH[x]_STAT.DV** bit inside the **ATOM[i]_CH[x]_STAT** register.

The behavior of an ATOM channel in SOMC mode with **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =0, when ARU is enabled and ARU blocking mode is disabled is shown in figure *89 "SOMC State diagram for SOMC mode, ARU enabled, ABM disabled, ATOM[i]_CH[x]_CTRL_SOMC.EUPM=0"* .

Figure 89 SOMC State diagram for SOMC mode, ARU enabled, ABM disabled, ATOM[i]_CH[x]_CTRL_SOMC.EUPM=0



Note:

In case of 'serve last' compare strategy the compare match in CCU0 is expected before the compare match in CCU1. But if the CCU1 compare match occurs first, **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** as well as **ATOM[i]_CH[x]_STAT.ACBO** will not be updated and also the output signal will not be set. After first occurance of CCU0 match and afterwards a CCU1 match the ordering is as expected and captures and signal level output changes will be performed on both matches.

In ARU non-blocking mode, if **ATOM[i]_CH[x]_CTRL_SOMC.FREEZE** =0 and the channel is disabled (*ATOM_ENDIS* =0), the comparison is stopped, the current compare values are no more valid and the ARU read request will be reset regardless of the current compare state. Therefore, no more ARU data can be further transferred to the ATOM channel. Afterwards, if the channel is enabled (*ATOM_ENDIS* =1) again, the comparison that was stopped before by disabling the channel cannot be continued. The channel must raise a new ARU read request again and then can start comparing after receiving new data.

In ARU non-blocking mode, if **ATOM[i]_CH[x]_CTRL_SOMC.FREEZE** =1 and the channel is disabled (*ATOM_ENDIS* =0), the comparison is stopped, but the current compare values are still valid and there will be no change to the ARU read request. Therefore, it is possible to continuously transfer more ARU data to the registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and the bit field **ATOM[i]_CH[x]_STAT.ACBI** if the ARU read request is 1 before the channel is disabled. That means, if the activated comparison has not matched its compare values stored in **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** before the channel is disabled, it is possible to overwrite **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** via ARU while the channel is disabled. Afterwards, when the channel is enabled again, it can start comparing the new values overwritten in **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , or it can continue comparison with the old compare values (i.e. before the channel was disabled), if no new ARU data was transferred to **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** .

15.3.2.3.2 ARU Blocking mode

When the compare registers are updated by ARU, the ATOM channel can be configured in a blocking manner to receive the ARU data. This can be configured by setting the **ATOM[i]_CH[x]_CTRL_SOMC.ABM** bit.

If the **ATOM[i]_CH[x]_CTRL_SOMC.ABM** and **ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN** bits are set, depending on compare strategy, **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** can be updated via ARU with new compare values. If the compare registers **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** are accepting these new data to be valid (indicated by bit **ATOM[i]_CH[x]_STAT.DV**), the ATOM channel stops requesting new data via ARU and waits for the compare match event to happen.

When the specified compare match event occurs, the shadow registers **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are updated together with the **ATOM[i]_CH[x]_STAT.ACBO** bits. The data in the shadow registers is marked as valid for the ARU and the **ATOM[i]_CH[x]_STAT.DV** bit or register **ATOM[i]_CH[x]_CTRL_SOMC** is reset.

If the registers **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** which hold the captured TBU time stamp values are read either by the ARU or through the configuration interface, the ATOM channel request new data again and the next write access to or update of the registers **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** via ARU or via configuration interface enables the new compare match check again. Furthermore, **ATOM[i]_CH[x]_STAT.DV** is again set to 1 which indicates the received new compare data is valid.

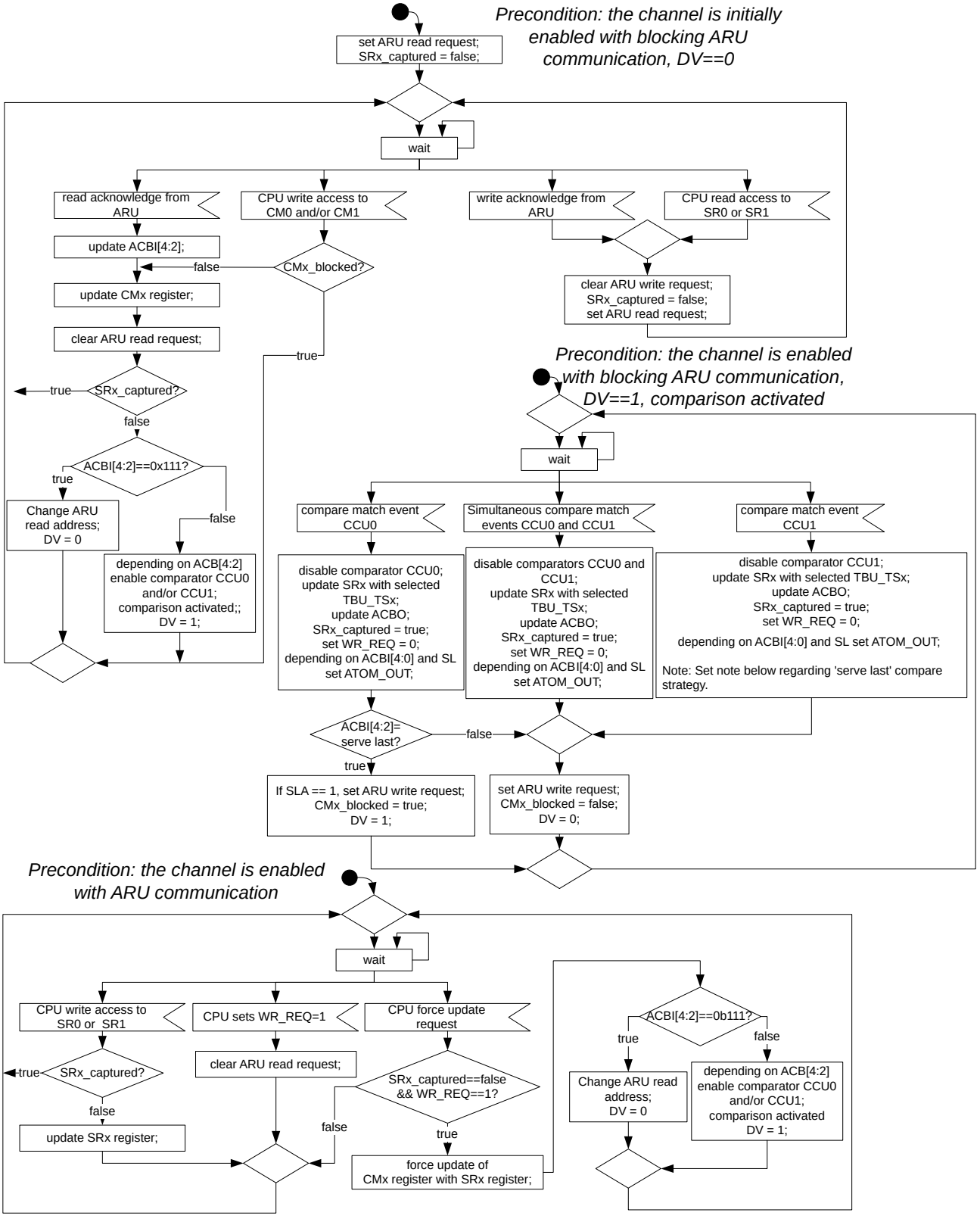
At least one of the registers **ATOM[i]_CH[x]_SR0.SR0** or **ATOM[i]_CH[x]_SR1.SR1** has to be read either via ARU or via configuration interface, before new data is requested via ARU.

In case of ABM=1, the application has to handle the situation that the ATOM does not request new data to update **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** until the captured values are read.
E.g. if an MCS task starts to write via ARU new data (with AWR(I) command) after capture of data in **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** , the task remains in the command until captured data is read by another task via ARU or the configuration interface.

It can be checked through the configuration interface at any time, if the ATOM channel has received valid data from the ARU and waits for a compare event to occur. This is signaled by a set **ATOM[i]_CH[x]_STAT.DV** bit.

The behavior of an ATOM channel in SOMC mode with **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =0, when ARU is enabled and ARU blocking mode is enabled is shown in figure *90 "SOMC State diagram for SOMC mode, ARU enabled and ABM enabled, ATOM[i]_CH[x]_CTRL_SOMC.EUPM=0"* .

Figure 90 SOMC State diagram for SOMC mode, ARU enabled and ABM enabled, ATOM[i]_CH[x]_CTRL_SOMC.EUPM=0



Note:

In case of 'serve last' compare strategy the compare match in CCU0 is expected before the compare match in CCU1. But if the CCU1 compare match occurs first, **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** as well as **ATOM[i]_CH[x]_STAT.ACBO** will not be updated and also the output signal will not be set. After first occurance of CCU0 match and afterwards a CCU1 match the ordering is as expected and captures and signal level output changes will be performed on both matches.

In ARU blocking mode, if **ATOM[i]_CH[x]_CTRL_SOMC.FREEZE** =0 and the channel is disabled (*ATOM_ENDIS* =0), the comparison is stopped, the current compare values are no more valid and the ARU read request will be reset regardless of the current compare state. Therefore, no more ARU data can be further transferred to the ATOM channel. Afterwards, if the channel is enabled (*ATOM_ENDIS* =1) again, the comparison that was stopped before by disabling the channel cannot be continued. The channel must raise a new ARU read request again and then can start comparing after receiving new data.

In ARU blocking mode, if **ATOM[i]_CH[x]_CTRL_SOMC.FREEZE** =1 and the channel is disabled (*ATOM_ENDIS* =0), the comparison is stopped, but the current compare values are still valid and there will be no change to the ARU read request. Therefore, it is possible to transfer one more ARU data to the registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and the bit field **ATOM[i]_CH[x]_STAT.ACBI** , if the ARU read request is 1 before the channel is disabled. However, in this mode, only one ARU data can be transferred while the channel is disabled because the ATOM channel will stop requesting new data until the received data is used. Afterwards, when the channel is enabled again (*ATOM_ENDIS* =0), the channel can start comparing the new values overwritten in **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , or it can continue comparison with the old compare values (i.e. before the channel was disabled), if no new ARU data was transferred to **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** .

15.3.2.3.3 ATOM SOMC Late update mechanism

Although the ATOM channel may be controlled by data received via the ARU, it is possible to request a late update of the compare register at any time through the configuration interface. This can be initiated by setting the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit. By doing this, the ATOM will request no further data from ARU (if ARU access was enabled). The channel will in any case continue to compare against the values stored inside the compare registers (if bit **ATOM[i]_CH[x]_STAT.DV** was set). Before the compare match event occurs, new compare values can be updated through the configuration interface by writing to the shadow registers and the ATOM channel can be forced to update the compare registers by writing to the **ATOM[i]_AGC_FUPD_CTRL** register.

If the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit is set and a compare match event occurs, any further access to the shadow registers **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** will be blocked and the force update of this channel will be blocked. In addition, the **ATOM[i]_CH[x]_STAT.WRF** bit is set. Thus, it can be determined that the late update failed by reading the **ATOM[i]_CH[x]_STAT.WRF** bit through the configuration interface.

In case bit **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =0, the following statements are true:

If a compare match event already occurred, the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit cannot be set to 1 until the channel is unlocked for a new compare match event by reading the shadow registers. In addition, the **ATOM[i]_CH[x]_STAT.WRF** bit is set, if a write access to the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit is executed through/from the configuration interface.

In case bit **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =1, the following statements are true:

If in case of 'serve last' strategy a CCU1 compare match event or in any other compare strategy a CCU0 or CCU1 compare match event has already occurred, the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit cannot be set to 1 until the channel is unlocked for a new compare match event by reading the shadow registers. In addition, the **ATOM[i]_CH[x]_STAT.WRF** bit is set, if a write access to the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit is tried through/from the configuration interface.

In general, for a late update the following has to be taken into account:

If after writing new values to shadow registers and setting **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** =1 but before the force update triggered by **ATOM[i]_AGC_FUPD_CTRL** , a compare event occurs with the old compare values, the **ATOM[i]_CH[x]_STAT.WRF** bit will be set. The force update will be blocked.

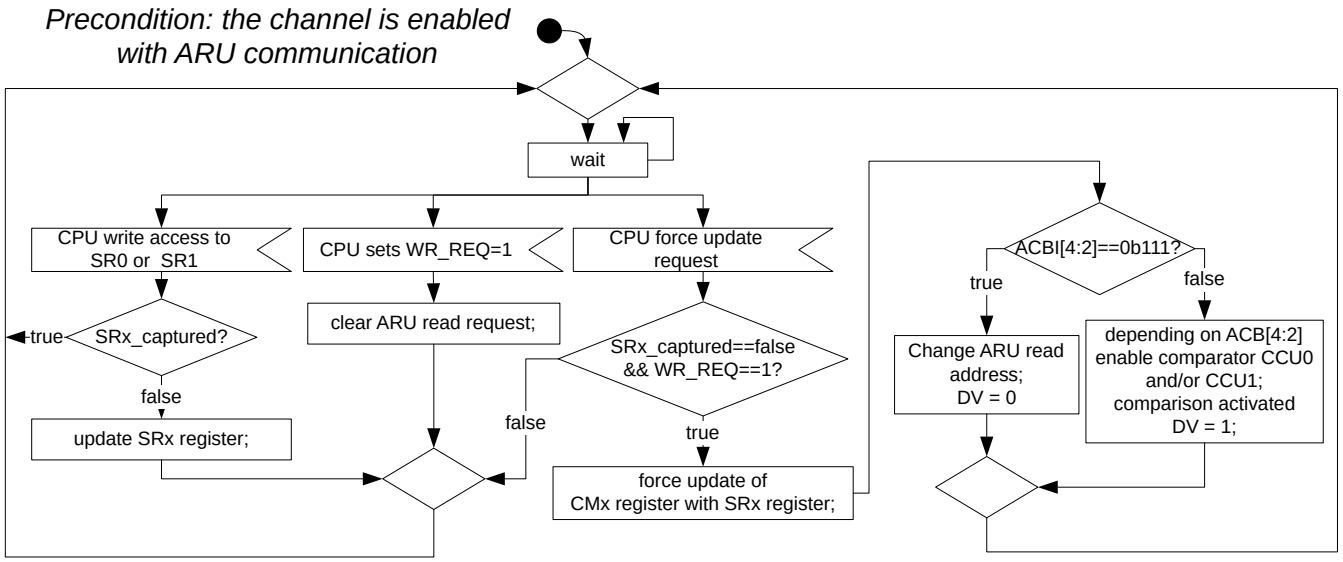
The **ATOM[i]_CH[x]_STAT.WRF** bit will be set in any case, if a write access to a blocked shadow register is attempted through the configuration interface.

The **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit will be reset at a compare match event. The bit **ATOM[i]_CH[x]_STAT.DV** will be reset at the CCU1 compare event in 'serve last' strategy or reset at CCU0/CCU1 compare event in other compare strategies.

After a capture event for register **ATOM[i]_CH[x]_SR0.SR0** and/or **ATOM[i]_CH[x]_SR1.SR1** the force update mechanism will be blocked until a read access to the register **ATOM[i]_CH[x]_SR0.SR0** or **ATOM[i]_CH[x]_SR1.SR1** by either the ARU or via the configuration interface takes place.
Writing to **ATOM[i]_CH[x]_SR0.SR0** or **ATOM[i]_CH[x]_SR1.SR1** after compare match causes an AEI write status 0b10.

The ATOM SOMC late update mechanism from configuration interface is shown in figure *91 "SOMC State diagram for late update requests by configuration interface"* .

Figure 91 SOMC State diagram for late update requests by configuration interface



15.3.3 ATOM Signal Output Mode PWM (SOMP)

In ATOM Signal Output Mode PWM (SOMP), the ATOM submodule channel is able to generate complex PWM signals with different pulse widths and periods by configuring different operation registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and the control register **ATOM[i]_CH[x]_CTRL_SOMP** . ATOM SOMP mode without considering ARU communication interface has almost the same functions as the TOM module.

The signal level of the pulse generated inside the period can be configured inside the channel control register (**ATOM[i]_CH[x]_CTRL_SOMP.SL**). The initial signal output level for the channel is the inverse pulse level defined by the **ATOM[i]_CH[x]_CTRL_SOMP.SL** bit. Figure *95 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0"* depicts this behavior.

The counter **ATOM[i]_CH[x]_CN0.CN0** of each channel can run in two different modes depending on configuration of **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** .
By default, the counter counts only up until it reaches a specified value and it is then reset to 0.
In the up down counter mode **ATOM[i]_CH[x]_CN0.CN0** switches between counting up and counting down.

Like TOM module, in SOMP mode, each ATOM channel can further work in one-shot mode by configuring **ATOM[i]_CH[x]_CTRL_SOMP.OSM** =1. Pulse count modulation mode and high resolution mode are also available depending on the configuration of the control register **ATOM[i]_CH[x]_CTRL_SOMP** .

15.3.3.1 Pulse width, period, signal level and clock frequency update mechanisms

Since ATOM shows a behavior similar to TOM module in SOMP mode, their update mechanisms of pulse width, period, signal level and clock frequency are also almost same. The two operation registers **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** can be updated with the content of the shadow register **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** . The register **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** that determines the clock resolution of the counter **ATOM[i]_CH[x]_CN0.CN0** can be updated with its shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** . The signal level defined in bit field **ATOM[i]_CH[x]_CTRL_SOMP.SL** can be updated with its shadow register **ATOM[i]_CH[x]_CTRL_SR.SL_SR** .

The update of the configuration register **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** with the content of their shadow registers can be performed under three conditions. Firstly, when **ATOM[i]_CH[x]_CN0.CN0** is reset by comparing **ATOM[i]_CH[x]_CN0.CN0** greater or equal than **ATOM[i]_CH[x]_CM0.CM0** -1, the update will be performed. Secondly, when the channel receives the trigger signal *ATOM_CH_TRIGIN* [x:x] (= *ATOM_CH_TRIGOUT* [x-1:x-1]) from another ATOM channel [x-1] or the external trigger signal *EXT_TRIGIN* of the assigned TIM channel [x] after it is synchronized to the selected *CCM[i]_CLK_RES* of channel x, the update will also be performed. These two update mechanisms must be enabled by configuring **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 0b10. Thirdly, these configuration registers can be force updated via a trigger signal via *ATOM_FUPD* or via *EXT_TRIGIN* .

Based on the update mechanisms above, these registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** , **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** can be updated synchronously and asynchronously. Synchronous update means that the pulse width, period duration, signal level and the counter operating clock frequency can be changed at the end of the running PWM period. Asynchronous update means that the pulse width, period duration, signal level and the counter operating clock frequency can also be changed during the actual running PWM period. The following two sections provide examples of synchronous update and asynchronous update of pulse width in continuous up mode with **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** = 0.

An update of pulse width, period and the counter clock frequency becoming effective synchronously with start of a new period can easily be reached by performing the below steps:

1. Disable the update of the action register with the content of the corresponding shadow register by setting the channel specific configuration bit field **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 0b01. (Configuring **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** =0b01 if in counting up-down mode.)
2. Write new desired values to **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** , **ATOM[i]_CH[x]_CTRL_SR.SL_SR** , **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** .
3. Enable update of the action register by setting the channel specific configuration bit field **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 0b10.

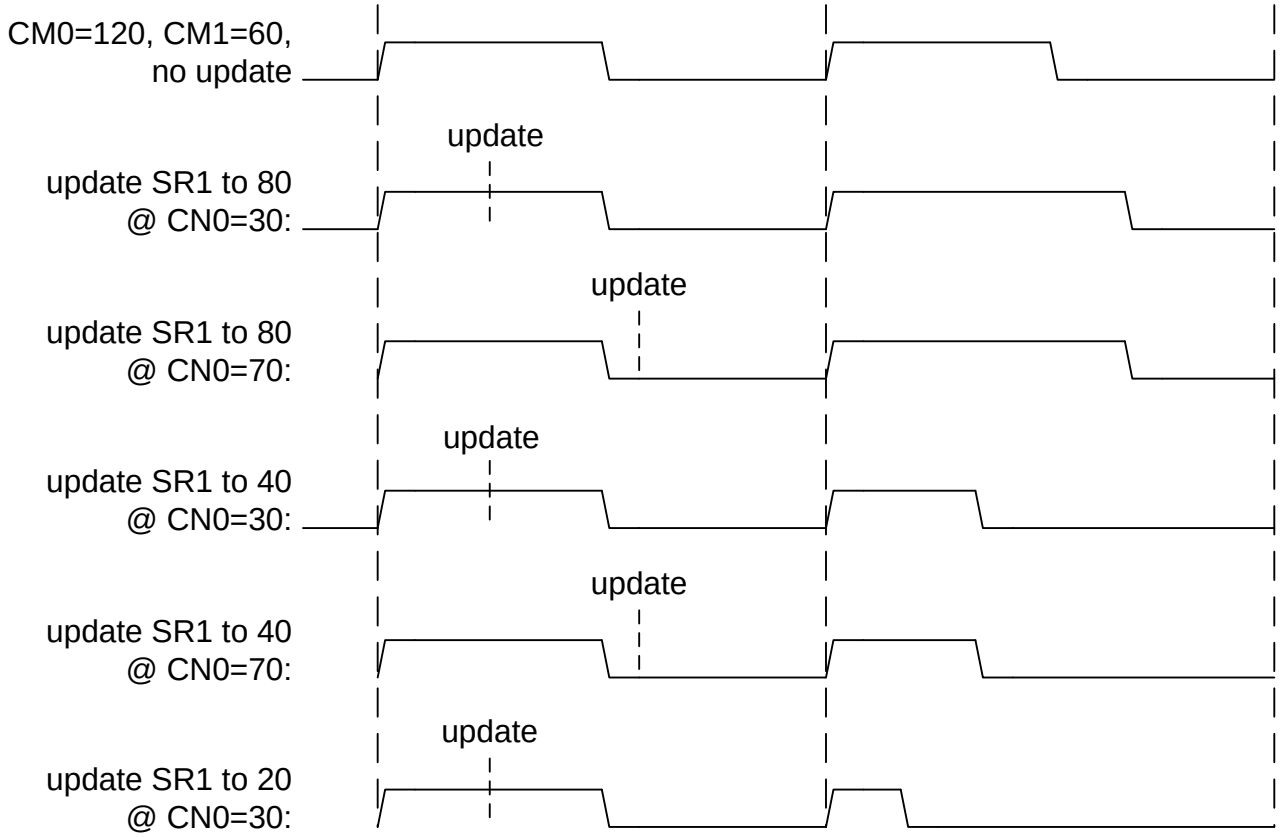
Asynchronous update of the operation registers can be realized by hot reconfiguring directly the registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** , **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** or hot reconfiguring their shadow registers and then using the force update mechanism. Such asynchronous update mechanism is sometimes helpful to change the PWM parameters of the current running PWM signal. For example, in continuous counting up mode, the current running PWM period can be reduced by asynchronously updating the value of **ATOM[i]_CH[x]_CM0.CM0** with a smaller value before the end of the current period. However, since the behavior depends on the access or update time of registers and the current running operation phase of the channel, which is often difficult to control, then the behavior after an asynchronous update can be unpredictable and unexpected. Therefore, such an asynchronous update must be carefully performed in real application. The following steps are recommended to apply the force update mechanism on channel x:

1. Disable the update of the action register with the content of the corresponding shadow register by setting the channel specific configuration bit field **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 0b01.
2. Write new desired values to **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** , **ATOM[i]_CH[x]_CTRL_SR.SL_SR** , **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** .
3. Enable the force update of the operation registers by setting the channel specific configuration bit field **ATOM[i]_AGC_FUPD_CTRL.FUPD_CTRL[x]** = 0b10, **ATOM[i]_AGC_FUPD_CTRL.RSTCN0_CH[x]** = 0b01.
4. Write **ATOM[i]_AGC_GLB_CTRL.HOST_TRIG** = 1 to trigger the force update.

15.3.3.1.1 Synchronous Update Of Pulse Width Only

A synchronous update of the pulse width only can be done by simply writing the desired new value to register **ATOM[i]_CH[x]_SR1.SR1** without preceding disable of the update mechanism (as described in the chapter above). As an example in SOMP continuous counting up mode with **ATOM[i]_CH[x]_CTRL.RST_CCU0** =0 is shown in figure *92 "Synchronous update of pulse width only in SOMP continuous counting up mode with ATOM[i]_CH[x]_CTRL.RST_CCU0=0"* , the new pulse width is then applied in the period following the period where the update of register **ATOM[i]_CH[x]_SR1.SR1** is done.

Figure 92 Synchronous update of pulse width only in SOMP continuous counting up mode with ATOM[i]_CH[x]_CTRL.RST_CCU0=0

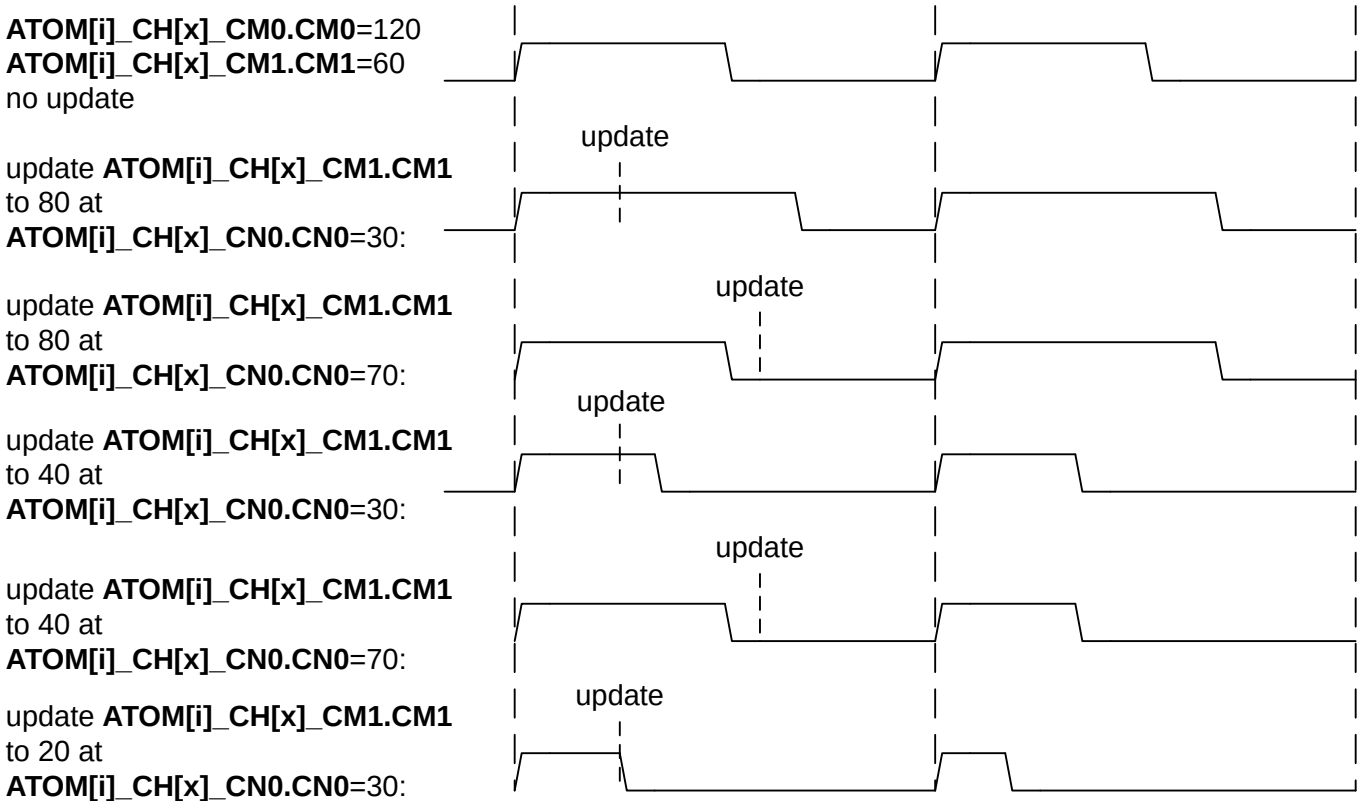


15.3.3.1.2 Asynchronous Update Of Pulse Width Only

If the update of the pulse width should be performed independent of the start of a new period (asynchronous), it is mandatory to additionally disable the synchronous update mechanism as a whole (i.e. configuring bits **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =0b01 of corresponding channel x) and then update the operation register. On the one hand, the desired new value can be written directly to register **ATOM[i]_CH[x]_CM1.CM1** . On the other hand, it can be realized by writing the new value to **ATOM[i]_CH[x]_SR1.SR1** and afterwards updating the value of **ATOM[i]_CH[x]_CM1.CM1** by using the asynchronous force update mechanism (configuring **ATOM[i]_AGC_FUPD_CTRL.FUPD_CTRL[x]** = 0b10, **ATOM[i]_AGC_FUPD_CTRL.RSTCN0_CH[x]** = 0b01 and then writing bit **ATOM[i]_AGC_GLB_CTRL.HOST_TRIG** = 1).

Depending on the point of time of the update of **ATOM[i]_CH[x]_CM1.CM1** in relation to the actual value of **ATOM[i]_CH[x]_CN0.CN0** and **ATOM[i]_CH[x]_CM1.CM1** , the new pulse width is applied in the current period or the following period. That means asynchronous update of **ATOM[i]_CH[x]_CM1.CM1** in the current period can at least change the pulse width in the next PWM signal. The behavior of the output signal due to the different possibilities of an asynchronous update during a PWM period in SOMP continuous counting up mode with **ATOM[i]_CH[x]_CTRL.RST_CCU0** =0 is shown in figure *93 "Asynchronous Update Of Pulse Width Only in SOMP continuous counting up mode with ATOM[i]_CH[x]_CTRL.RST_CCU0=0"* . The new pulse width may jitter from update to update by maximum one period (given by **ATOM[i]_CH[x]_CM0.CM0**). However, the period remains unchanged. On an asynchronous update, it is guaranteed, that no spike occurs at the output port of the channel due to a too late update of the operation registers.

Figure 93 Asynchronous Update Of Pulse Width Only in SOMP continuous counting up mode with ATOM[i]_CH[x]_CTRL.RST_CCU0=0



15.3.3.2 ARU controlled update

ARU Data input stream pipeline structure for SOMP mode is shown in figure *94 "ARU Data input stream pipeline structure for SOMP mode"* .

After the channel is enabled by setting **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[x]** =1, **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =1 and ARU input stream is enabled by setting of **ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** =1, the first ARU read request is set up in the next active clock cycle that is selected by **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** .

For the synchronous update mechanism, the generation of a complex PWM output waveform is possible without interaction over configuration interface by reloading the shadow registers **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** and **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** or the **ATOM[i]_CH[x]_STAT.ACBI** bit field over the ACI sub-unit from the ARU, while the ATOM channel operates on the registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** and **ATOM[i]_CH[x]_CTRL_SOMP.SL** .

If ARU access is enabled (**ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** =1) and after the first ARU data is transferred, the internal signal *ATOM_ARU_VALID* will be 1. As shown in figure *94 "ARU Data input stream pipeline structure for SOMP mode"* , the bits **ATOM[i]_CH[x]_STAT.ACBI** [4:1] received via ARU will be used as shadow register for the update of the CMU clock source register **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** . If ARU access is enabled but the first ARU data is not transferred yet (indicated by the internal signal *ATOM_ARU_VALID* = 0), **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** is still updated from **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** .

Similarly, if **ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** =1, after the first ARU data is transferred, **ATOM[i]_CH[x]_STAT.ACBI** will work as a shadow register for the update of the signal level register **ATOM[i]_CH[x]_CTRL_SOMP.SL** . Otherwise, **ATOM[i]_CH[x]_CTRL_SOMP.SL** is updated from **ATOM[i]_CH[x]_CTRL_SR.SL_SR** .

In continuous counting up mode under ARU control, this internal update mechanism is established at the end of each PWM period. The shadow registers **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** are loaded into the operation registers **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** , the counter register is reset, the new clock source is selected according to **ATOM[i]_CH[x]_STAT.ACBI[4:1]** , the signal level is set according to **ATOM[i]_CH[x]_STAT.ACBI[0:0]** and the new PWM generation starts.

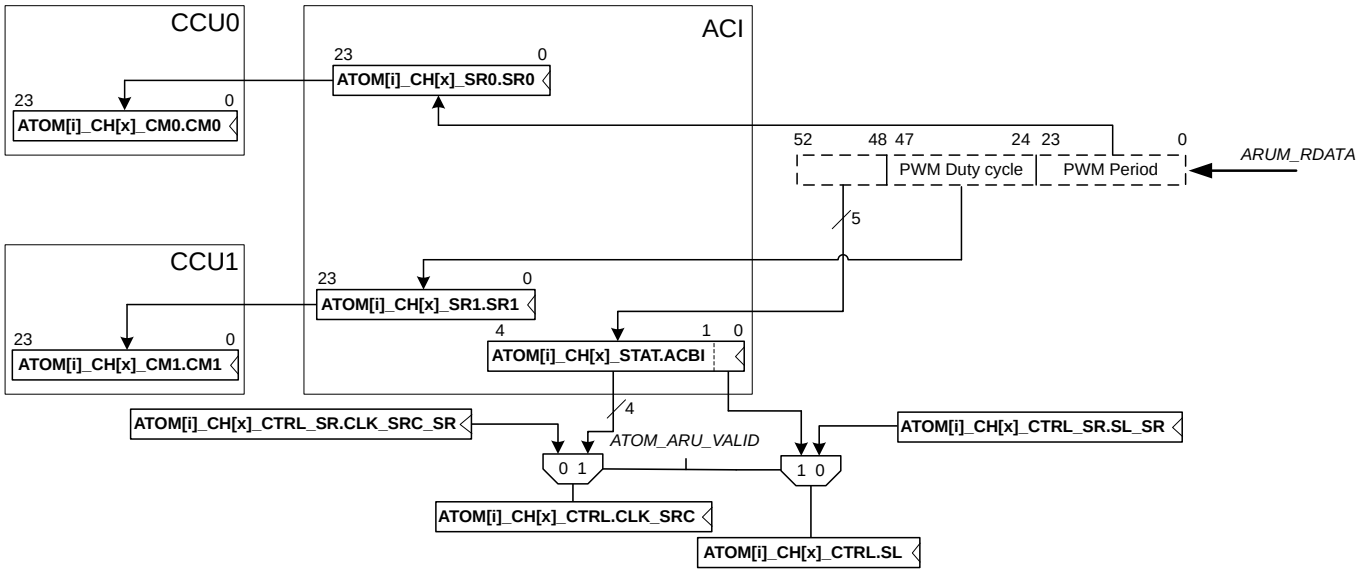
In parallel, the ATOM channel raises a read request to ARU to reload the shadow registers with new values while the ATOM channel operates on the current operation registers. To guarantee the reloading, the PWM period must not be smaller than the worst case of ARU round trip time plus two clock cycles for the ARU internal pipeline stages to load data to shadow registers and plus one clock cycle for reloading operation registers from their shadow registers inside ATOM. Furthermore, the ARU data source for new PWM parameters must provide data within this time. Otherwise, the old PWM parameters are used for the next PWM signal because the shadow registers are not updated with new ARU data.

Similarly, in continuous counting up-down mode, this internal update mechanism is established at two possible points of time according to **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** . If **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** = 0b01, the update happens at the end of each period when the counter **ATOM[i]_CH[x]_CN0.CN0** = 0. The operation registers are updated from their shadow registers and **ATOM[i]_CH[x]_STAT.ACBI** , and the next PWM generation starts together with new request to ARU for new data. In this case, in order to guarantee continuous PWM signals generation with different parameters, the ARU data source must provide data within the current running PWM period and the PWM period must be not shorter than the worst case of ARU round trip time plus three clock cycles as mentioned before. If **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** = 0b10, the update happens when the counter **ATOM[i]_CH[x]_CN0.CN0** is equal to the maximum

value. The operation registers are updated and the counter starts counting down together with new request to ARU for new data. In this case, like the update without ARU communication, it is only allowed to change **ATOM[i]_CH[x]_CM1.CM1** . Therefore, the bit field of ARU data for period, clock source and signal level should always be the same. If **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** = 0b11, both update options above are possible. Therefore, it is possible to raise 2 requests in one PWM period together with two times operation registers update.

When updated over the ARU, the user has to ensure that the new period duration is located in the lower (bits [23:0]) ARU data word and the pulse width duration is located in the upper (bits [47:24]) ARU data word. The new clock source is specified in the ARU control bits [52:49] and the new signal level **ATOM[i]_CH[x]_CTRL_SOMP.SL** is specified in the ARU control bit 48. In addition, it is possible to only update the period (**ATOM[i]_CH[x]_SR0.SR0**) by the lower ARU data word (bits [23:0]) or only update the pulse width (**ATOM[i]_CH[x]_SR1.SR1**) by the upper ARU data word(bits [47:24]). It is controlled by the ARU data select register **ATOM[i]_CH[x]_CTRL_SOMP.ADL** (please refer to the register description for more details).

Figure 94 ARU Data input stream pipeline structure for SOMP mode



15.3.3.3 Update controlled by configuration interface

Without ARU communication, update of operation registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** can also be controlled by the AEI bus interface. In this case, **ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** has to be set to 0. On the one hand, they can be synchronously updated from their shadow registers **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** , **ATOM[i]_CH[x]_CTRL_SR.SL_SR** and **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** when *ATOM_UPEN* [x:x] = 1 in the AGC sub-unit. Normally, synchronous update happens at the end of PWM period when the counter is reset. On the other hand, they can be asynchronously updated from the shadow registers triggered by *ATOM_FUPD* [x:x] signal in AGC sub-unit. Furthermore, **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_CTRL_SOMP.SL** can be directly asynchronously written via the configuration interface. Such an asynchronous update happens during one PWM period that should be carefully applied to avoid unexpected and unpredicted behavior. For asynchronous update, the synchronous update mechanism has to be locked via the **ATOM[i]_AGC_GLB_CTRL** register with the *ATOM_UPEN* [x:x] signal in the AGC sub-unit.

On a compare match of **ATOM[i]_CH[x]_CN0.CN0** and **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** , the output signal level of *ATOM_OUT* is toggled according to the signal level output bit **ATOM[i]_CH[x]_CTRL_SOMP.SL** . Please refer to different mode sections for more output generation details.

Thus, the output level of PWM pulse width can be changed during runtime by synchronous update or asynchronous update of **ATOM[i]_CH[x]_CTRL_SOMP.SL** bit of the channel configuration register.

15.3.3.4 Continuous Counting Up Mode

In SOMP continuous counting up mode the ATOM channel starts incrementing the counter **ATOM[i]_CH[x]_CN0.CN0** once it is enabled by setting the corresponding bits in register **ATOM[i]_TGC[g]_ENDIS_STAT** (refer to chapter *80 "ATOM Global channel control mechanism"* for details of enabling a ATOM channel).
In this mode with **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** =0b00 (i.e. **ATOM[i]_CH[x]_CN0.CN0** counts only up), depending on configuration bits **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** the counter register **ATOM[i]_CH[x]_CN0.CN0** can be reset either when the counter value is equal to the compare value **ATOM[i]_CH[x]_CM0.CM0** (i.e. **ATOM[i]_CH[x]_CN0.CN0** counts only from 0 to **ATOM[i]_CH[x]_CM0.CM0** -1 and is then reset to 0) or when the trigger signal *ATOM_CH_TRIGOUT* [x-1:x-1] of the preceding channel x-1 (which can also be the last channel of preceding instance ATOM[i-1]) is captured by the ATOM[i] or when there is trigger signal *EXT_TRIGIN* [x:x] of the assigned TIM channel x after it was synchronized to the selected clock resolution signal.
In this case, if **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =0b10, also the operation register **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** are updated from their shadow registers together with **ATOM[i]_CH[x]_CN0.CN0** reset.

As an exception, the input *ATOM_CH_TRIGIN* [0:0] of instance ATOM0 is triggered by its own last channel 7 via signal *ATOM_CH_TRIGOUT* [7:7].

The duration of the pulse high or low time and period is measured with the counter in sub-unit CCU0. The trigger of the counter is one of the eight CMU clock signals configurable in the channel control register **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** . The register **ATOM[i]_CH[x]_CM0.CM0** holds the duration of the period and the register **ATOM[i]_CH[x]_CM1.CM1** holds the duration of the pulse width in clock ticks of the selected CMU clock *SEL_CLK_EN* . The signal level of the generated output signal can be configured with the configuration bit **ATOM[i]_CH[x]_CTRL_SOMP.SL** .

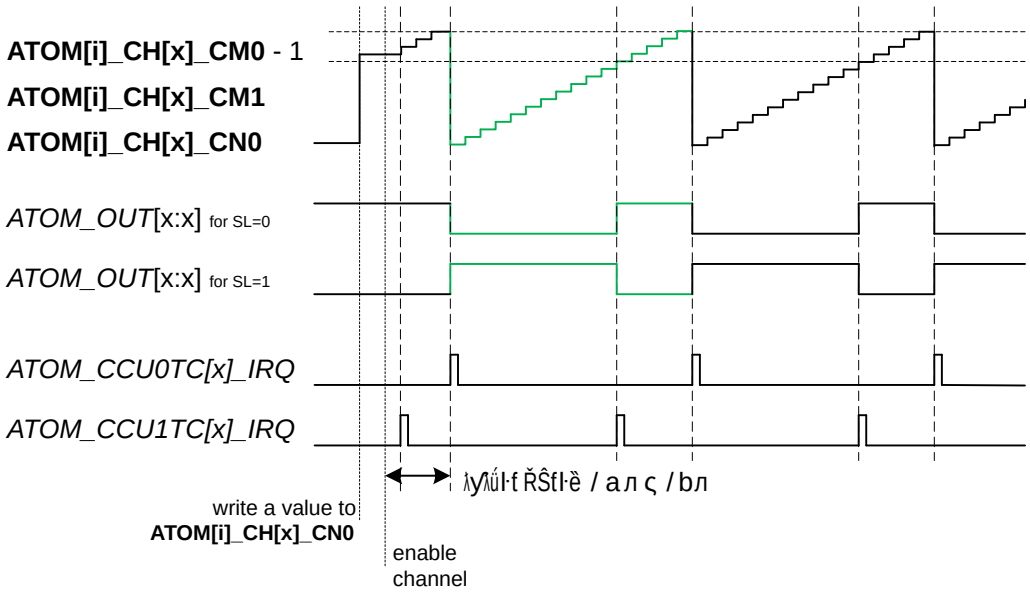
If counter register **ATOM[i]_CH[x]_CN0.CN0** of channel x is reset by its own CCU0 unit (i.e. the compare match of **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM0.CM0** -1 configured by **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0), the PWM output behavior is shown in figure *95 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0"* and the following statements are valid:

- After ATOM i channel x is enabled, **ATOM[i]_CH[x]_CN0.CN0** counts firstly from its initial value to **ATOM[i]_CH[x]_CM0.CM0** -1 and is then reset to 0. This phase is defined as initial delay that can be flexibly configured as by the value of (**ATOM[i]_CH[x]_CM0.CM0** - **ATOM[i]_CH[x]_CN0.CN0**) multiplied with the selected CMU clock period.
- When **ATOM[i]_CH[x]_CN0.CN0** is reset from **ATOM[i]_CH[x]_CM0.CM0** -1 to 0, an edge to **ATOM[i]_CH[x]_CTRL_SOMP.SL** is generated.
- When **ATOM[i]_CH[x]_CN0.CN0** is incrementing and reaches **ATOM[i]_CH[x]_CM1.CM1** (**ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM1.CM1**), an edge to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** is generated.
- **ATOM[i]_CH[x]_CN0.CN0** always counts from 0 to **ATOM[i]_CH[x]_CM0.CM0** -1 and is then reset to 0.
- If **ATOM[i]_CH[x]_CM0.CM0** =0 or **ATOM[i]_CH[x]_CM0.CM0** =1, the counter **ATOM[i]_CH[x]_CN0.CN0** is constant 0. It is highly recommended to configure **ATOM[i]_CH[x]_CM0.CM0** >1.
- If **ATOM[i]_CH[x]_CM1.CM1** =0 and **ATOM[i]_CH[x]_CM0.CM0** >1, the output is ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** (i.e. 0% duty cycle PWM signal).
- If **ATOM[i]_CH[x]_CM1.CM1** >= **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM0.CM0** >1, the output is **ATOM[i]_CH[x]_CTRL_SOMP.SL** (i.e. 100% duty cycle PWM signal).

Since the *ATOM_OUT* signal level is defined as ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** when the ATOM channel is disabled without setting freeze mode, the first PWM period can be shifted earlier by writing an initial offset value to **ATOM[i]_CH[x]_CN0.CN0** register. By doing this, the ATOM channel first counts until **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** and then it toggles the output signal at *ATOM_OUT* .

In this case, as shown in figure *95 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0"* , 2 interrupt signals *ATOM_CCU0TC[x]_IRQ* and *ATOM_CCU1TC[x]_IRQ* of channel x can be generated together with output generation *ATOM_OUT* [x:x] and both of them are shown on *ATOM_IRQ[x:x]* if **ATOM[i]_CH[x]_IRQ_EN** =0b11. When **ATOM[i]_CH[x]_CN0.CN0** is reset from **ATOM[i]_CH[x]_CM0.CM0** -1 to 0, *ATOM_CCU0TC[x]_IRQ* of channel x is generated. When **ATOM[i]_CH[x]_CN0.CN0** is incrementing to reach **ATOM[i]_CH[x]_CM1.CM1** , *ATOM_CCU1TC[x]_IRQ* of channel x is generated. Especially, if the initial configuration before enabling the channel fulfills the condition **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM1.CM1** - 1, an interrupt is generated after 1 selected CMU clock period on *CCU1TC[x]_IRQ* . Similarly but not shown in the figure, if the initial configuration before enabling the channel fulfills the condition **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM0.CM0** - 1, an interrupt is generated after 1 selected CMU clock period on *CCU0TC[x]_IRQ* . The interrupt signals are always aligned with the edges of the output *ATOM_OUT* [x:x].

Figure 95 PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0



If the counter register **ATOM[i]_CH[x]_CN0.CN0** of channel x is reset by the trigger signal coming from another channel or the assigned TIM module (configured by **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1), the PWM output behavior is shown in figure *96 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1"* and the following statements are valid (with MAX=number of selected *CCM[i]_CLK_RES* cycle between two trigger signals):

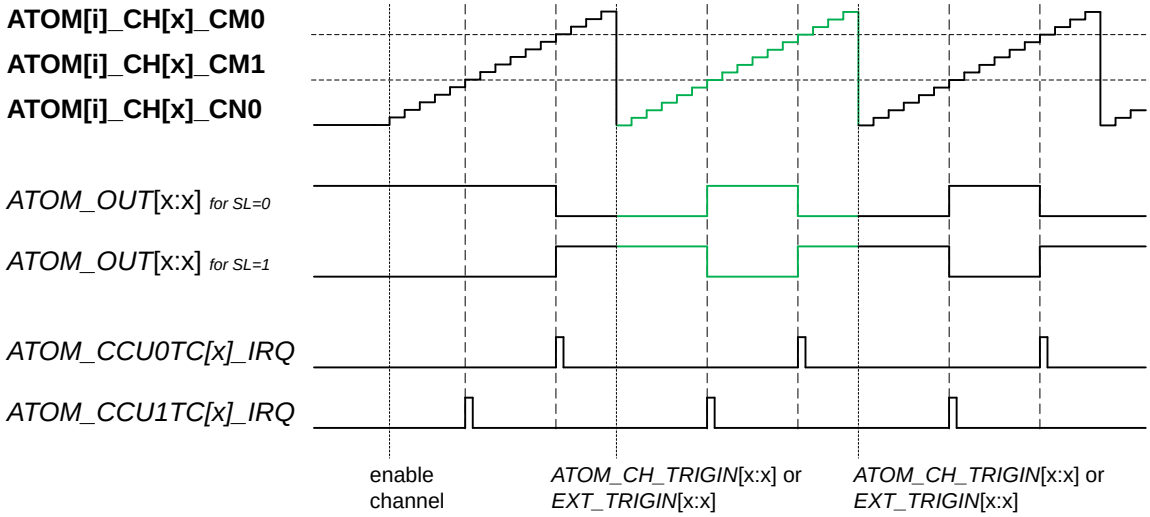
- **ATOM[i]_CH[x]_CN0.CN0** counts from 0 to MAX-1 and is then reset to 0 at the falling edge of *ATOM_CH_TRIGIN* [x:x] or at falling edge of *EXT_TRIGIN_S* (the synchronized signal of of *EXT_TRIGIN* [x:x], see chapter *15.2.2 "External Trigger Interface"*).
- If **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM1.CM1** , an edge to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** is generated.
- If **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** , an edge to **ATOM[i]_CH[x]_CTRL_SOMP.SL** is generated.
- As soon as **ATOM[i]_CH[x]_CN0.CN0** reaches the value of **ATOM[i]_CH[x]_CM0.CM0** while **ATOM[i]_CH[x]_CM1.CM1** is equal to **ATOM[i]_CH[x]_CM0.CM0** , an edge to **ATOM[i]_CH[x]_CTRL_SOMP.SL** is generated at the output or the output remains at **ATOM[i]_CH[x]_CTRL_SOMP.SL** level depending on the former level of the output (**ATOM[i]_CH[x]_CM0.CM0** has higher priority). Please note that this configuration is not suitable for generating 100% duty cycle.
- If **ATOM[i]_CH[x]_CM0.CM0** =0 and **ATOM[i]_CH[x]_CM1.CM1** >MAX, the output is a pulse of **ATOM[i]_CH[x]_CTRL_SOMP.SL** for the period MAX (i.e. 100% duty cycle PWM signal).
- If **ATOM[i]_CH[x]_CM0.CM0** > MAX and **ATOM[i]_CH[x]_CM1.CM1** =0, the output is ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** (i.e. 0% duty cycle PWM signal).

In this case, as shown in figure *96 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1"* , 2 interrupt signals *ATOM_CCU0TC[x]_IRQ* and *ATOM_CCU1TC[x]_IRQ* of channel x can be generated together with output generation *ATOM_OUT* [x:x] and both of them are shown on *ATOM_IRQ[x:x]* if **ATOM[i]_CH[x]_IRQ_EN** =0b11. When **ATOM[i]_CH[x]_CN0.CN0** is incrementing to reach **ATOM[i]_CH[x]_CM0.CM0** , *ATOM_CCU0TC[x]_IRQ* of channel x is generated. When **ATOM[i]_CH[x]_CN0.CN0** is incrementing to reach **ATOM[i]_CH[x]_CM1.CM1** , *ATOM_CCU1TC[x]_IRQ* of channel x is generated. The interrupt signals are always aligned with the edges of the output *ATOM_OUT* [x:x].

According to internal trigger interface specification *15.2.3 "Internal Trigger Interface"* , more than one channel (maybe also in different clusters) can be triggered by a common preceding channel to synchronously generate consistent PWM signals. In such applications, the channels and instances between the triggering channel to the triggered channels build up a trigger chain and the triggered channels should be configured with same PWM parameters (i.e. period, duty cycle, signal level and clock source). However, possible pipeline registers according to *15.2.3 "Internal Trigger Interface"* may introduce delay of several cluster clock periods *CLS[i]_CLK* or GTM clock periods *CLK* in the trigger chain. As a result, the triggered channels in the trigger chain may be asynchronously triggered and have different behavior even though they are triggered by a common channel and they are configured with same PWM parameters. Therefore, if in real applications there are pipeline registers in the trigger chain, it is necessary to configure the selected CMU clock source to be a slower frequent clock enable signal to avoid the influence of the introduced delay so that the triggered channels can be synchronously triggered and have consistent behavior. The period between two enable pulses of the selected CMU clock source (i.e. the distance between 2 falling edges of *SEL_CMU_CLKEN*) should be greater than the total delay caused by all the pipeline registers in the trigger chain. Especially, if the triggering channel and other triggered channels are in a common instance i where *CLS[i]_CLK* is enabled with clock divider of 2 (**GTM_CLS_CLK_CFG.CLS[i]_CLK_DIV** =2) and there is only one pipeline register (with delay of one *CLK* period) between channel 3 and channel 4 in the trigger chain, then such a delay less than *CLS[i]_CLK* period can be ignored regardless of the selection of CMU clock source.

If there are pipeline registers between the triggering channel and the triggered channel and the introduced delay is greater than or equal to one cluster clock period, it is not recommended to configure **ATOM[i]_CH[x]_CM0.CM0** =0 or **ATOM[i]_CH[x]_CM1.CM1** =0. In this case, 0% duty cycle or 100% duty cycle PWM signal can be realized by configuring **ATOM[i]_CH[x]_CM0.CM0** = **ATOM[i]_CH[x]_CM1.CM1** = MAX with **ATOM[i]_CH[x]_CTRL_SOMP.SL** =0 or 1.

Figure 96 PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1



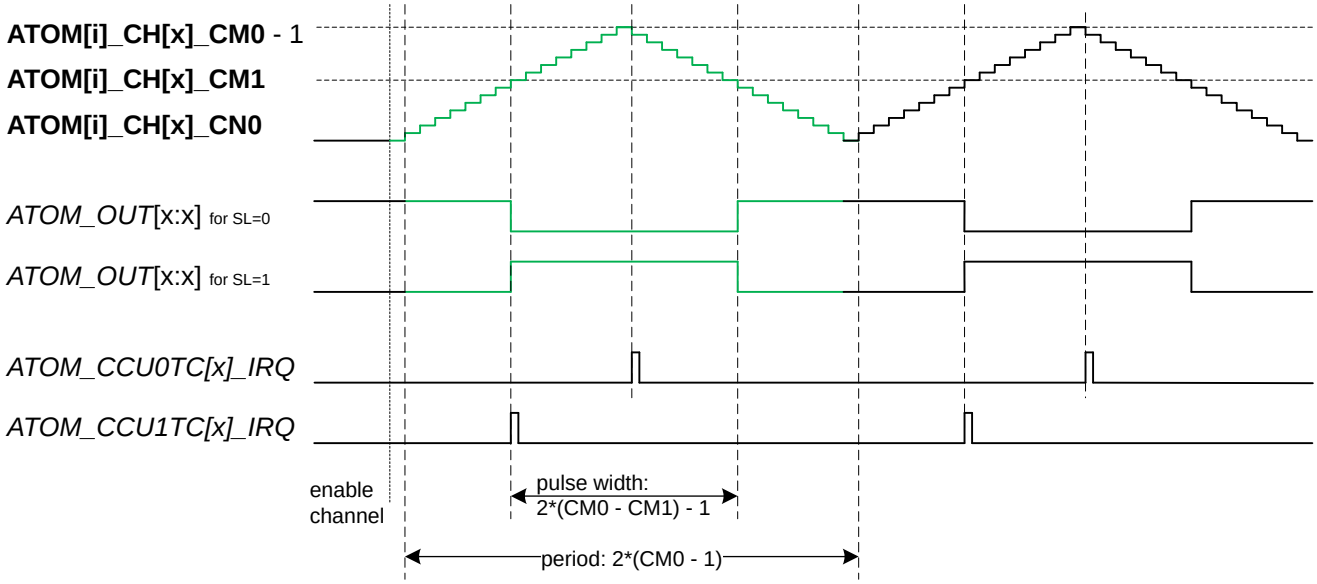
In case of **ATOM[i]_AGC_GLB_CTRL.UPEN[x]=0b10** , when the counter value **ATOM[i]_CH[x]_CN0.CN0** reaches the compare value in register **ATOM[i]_CH[x]_CM0.CM0** (in fact **ATOM[i]_CH[x]_CM0.CM0** -1) if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0 or the channel captures the falling edge of the trigger signal if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1, the operation registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** are updated with the content of their shadow registers **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** , **ATOM[i]_CH[x]_CTRL_SR.SL_SR** and **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** .

The operation registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** can also be asynchronously updated from their shadow registers with the force update mechanism. Please refer to chapter *15.3.3.1 "Pulse width, period, signal level and clock frequency update mechanisms"* for the force update configuration steps and more details. In continuous counting up mode, the force update mechanism can be also used to stop the current running PWM signal generation and restart a new PWM period with new PWM parameters by additionally setting **ATOM[i]_AGC_FUPD_CTRL.RSTCN0_CH[x]** = 0b10 together with **ATOM[i]_AGC_FUPD_CTRL.FUPD[x]** = 0b10. That means the force update event will not only update the operation registers from their shadow registers, but also simultaneously reset the counter to restart the next new PWM period with setting the output as **ATOM[i]_CH[x]_CTRL_SOMP.SL** if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0 or without changing the output if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1.

15.3.3.5 Continuous Counting Up-Down Mode

In SOMP mode, if **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** != 0b00, the counter **ATOM[i]_CH[x]_CN0.CN0** will count up and down. Depending on the configuration bit **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** , the counter register **ATOM[i]_CH[x]_CN0.CN0** changes the direction either when the counter value is equal to the compare value **ATOM[i]_CH[x]_CM0.CM0** -1 or when triggered by the internal trigger signal *ATOM_CH_TRIGIN* [x:x] (i.e. *ATOM_CH_TRIGOUT* [x-1:x-1] from the preceding channel [x-1]) or the external trigger signal *EXT_TRIGIN* [x:x] from TIM module [i].

Figure 97 PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in in continuous counting up-down mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0



The clock of the counter register **ATOM[i]_CH[x]_CN0.CN0** can be one of the CMU enable clock resolutions *CCM[i]_CLK_RES* .
If **ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** =0, the clock for **ATOM[i]_CH[x]_CN0.CN0** is defined by **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** value.
If **ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** =1, the clock for **ATOM[i]_CH[x]_CN0.CN0** is defined by **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** value received via ARU.

If **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0, the counter register **ATOM[i]_CH[x]_CN0.CN0** of channel x will change its counting direction by the compare match of **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM0.CM0** -1 in its own CCU0 unit and the following statements are valid:

- In each PWM period, **ATOM[i]_CH[x]_CN0.CN0** firstly counts continuously up from 1 to **ATOM[i]_CH[x]_CM0.CM0** -1 and then down to 0. So period = 2*(**ATOM[i]_CH[x]_CM0.CM0** -1).
- If **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM1.CM1** , the output is *ATOM_OUT* [x:x] set to **ATOM[i]_CH[x]_CTRL_SOMP.SL** .
- If **ATOM[i]_CH[x]_CN0.CN0** < **ATOM[i]_CH[x]_CM1.CM1** , the output is *ATOM_OUT* [x:x] set to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** . So pulse width = 2*(**ATOM[i]_CH[x]_CM0.CM0** - **ATOM[i]_CH[x]_CM1.CM1**)-1.
- If **ATOM[i]_CH[x]_CM0.CM0** <= 1, the counter behavior of **ATOM[i]_CH[x]_CN0.CN0** is unexpected (not recommended).
- If **ATOM[i]_CH[x]_CM1.CM1** = 0 and also **ATOM[i]_CH[x]_CM0.CM0** > 1, the output *ATOM_OUT* [x:x] is **ATOM[i]_CH[x]_CTRL_SOMP.SL** (i.e. 100% duty cycle).
- If **ATOM[i]_CH[x]_CM1.CM1** >= **ATOM[i]_CH[x]_CM0.CM0** and also **ATOM[i]_CH[x]_CM0.CM0** > 1, the output *ATOM_OUT* [x:x] is ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** (i.e. 0% duty cycle).

This behavior is depicted in figure *97 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in in continuous counting up-down mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0"* . In this case, if **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =1, the operation registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** can be updated when **ATOM[i]_CH[x]_CN0.CN0** = 0 (i.e. counting direction changes from down to up) or **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** -1 (i.e. counting direction changes from up to down) depending on various configurations of **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** (see the register description for more details). Especially, it is not recommended to change the value

of **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** when **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** -1. Otherwise, the behavior may be unexpected.

Like continuous counting up mode, interrupt signals *ATOM_CCU0TC[x]_IRQ* and *ATOM_CCU1TC[x]_IRQ* of channel x are generated together with PWM signal output edges. When **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM1.CM1** at the first time and the first edge is generated, an interrupt on *ATOM_CCU1TC[x]_IRQ* is generated. When **ATOM[i]_CH[x]_CN0.CN0** changes its counting direction from up to down, an interrupt on *ATOM_CCU0TC[x]_IRQ* is generated.

If **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1, the counter register **ATOM[i]_CH[x]_CN0.CN0** of channel x will change its counting direction by the trigger signal coming from another channel or the assigned TIM module and the following statements are valid:

- **ATOM[i]_CH[x]_CN0.CN0** firstly counts continuously up. At the falling edge of the trigger signal pulse *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN_S* (the synchronized signal of *EXT_TRIGIN* [x:x], see chapter *15.2.2 "External Trigger Interface"*), the counter switches to count down mode. If **ATOM[i]_CH[x]_CN0.CN0** reaches 0, it will count up again.
- If **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM1.CM1** , the output *ATOM_OUT* [x:x] is set to **ATOM[i]_CH[x]_CTRL_SOMP.SL** .
- If **ATOM[i]_CH[x]_CN0.CN0** < **ATOM[i]_CH[x]_CM1.CM1** , the output *ATOM_OUT* [x:x] is set to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** .
- If **ATOM[i]_CH[x]_CM1.CM1** = 0, the output *ATOM_OUT* [x:x] is **ATOM[i]_CH[x]_CTRL_SOMP.SL** (i.e. 100% duty cycle).
- If **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM0.CM0** , the output *ATOM_OUT_T* [x:x] is set to **ATOM[i]_CH[x]_CTRL_SOMP.SL** .
- If **ATOM[i]_CH[x]_CN0.CN0** < **ATOM[i]_CH[x]_CM0.CM0** , the output *ATOM_OUT_T* [x:x] is set to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** .
- If **ATOM[i]_CH[x]_CM0.CM0** = 0, the output *ATOM_OUT_T* [x:x] is **ATOM[i]_CH[x]_CTRL_SOMP.SL** (i.e. 100% duty cycle).

This behavior is depicted in figure *98 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in in continuous counting up-down mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1"* . In this case, if **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =1, the operation registers **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** can also be updated when **ATOM[i]_CH[x]_CN0.CN0** = 0 (i.e. counting direction changes from down to up) or **ATOM[i]_CH[x]_CN0.CN0** reaches its maximal value (i.e. counting direction changes from up to down) depending on various configurations of **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** (see the register description for more details). Especially, it is not recommended to change the value of **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** when **ATOM[i]_CH[x]_CN0.CN0** reaches its maximal value (i.e. counting direction changes from up to down). Otherwise, the behavior may be unexpected.

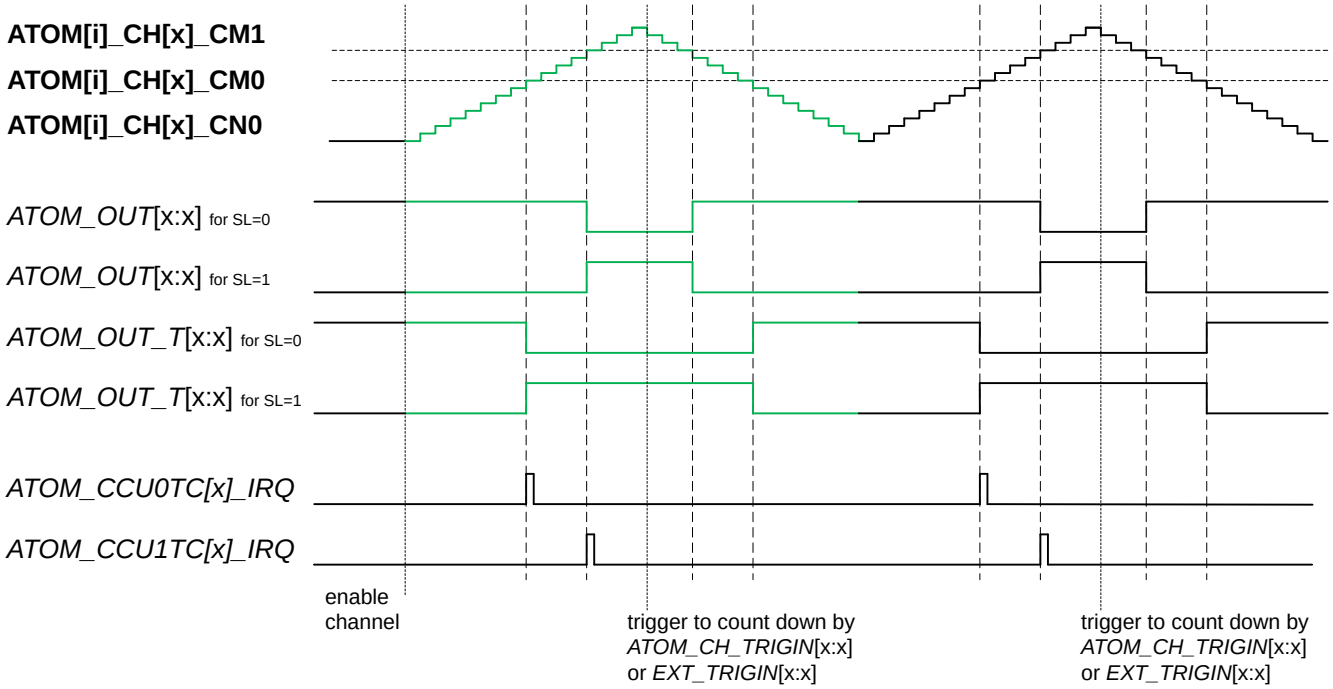
If **ATOM[i]_CH[x]_IRQ_EN** =0b11, interrupt signals *ATOM_CCU0TC[x]_IRQ* and *ATOM_CCU1TC[x]_IRQ* of channel x are generated together with PWM signal output. When **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM1.CM1** at the first time and the first edge of *ATOM_OUT* [x:x] is generated, an interrupt on *ATOM_CCU1TC[x]_IRQ* is generated. When **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** at the first time and the first edge of *ATOM_OUT_T* [x:x] is generated, an interrupt on *ATOM_CCU0TC[x]_IRQ* is generated.

In case of continuous counting up-down mode and **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1, it is recommended to configure the ATOM channel according to the following statements in order to realize several cyclic central aligned PWM signals:

- The triggering channel and the triggered channel are both running in up-down mode.
- The time between two trigger signals is equal to the time needed for **ATOM[i]_CH[x]_CN0.CN0** of triggered channel to count back to 0 and again up to the same upper value.

The second recommendation can be fulfilled by synchronizing the start of the triggering channel and the triggered channel, i.e. let both channels start with **ATOM[i]_CH[x]_CN0.CN0** = 0 and same clock source.
If there is a pipeline register in the trigger chain, the additional delay of one clock period has to be taken into account by starting the triggering channel with **ATOM[i]_CH[x]_CN0.CN0** = 1 (i.e. 1 greater than **ATOM[i]_CH[x]_CN0.CN0** of the triggered channel).

Figure 98 PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in in continuous counting up-down mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1



15.3.3.6 One-shot Counting Up Mode

The ATOM channel can operate in one-shot mode when the **ATOM[i]_CH[x]_CTRL_SOMP.OSM** bit is set. One-shot mode means that a single pulse with the pulse level defined in bit **ATOM[i]_CH[x]_CTRL_SOMP.SL** is generated on the output signal. In this case, ARU interface communication is not supported, i.e. **ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** must be configured as 0.

Firstly, the channel has to be enabled by setting the corresponding **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[k]** .

Unlike in the continuous counting modes, the counter **ATOM[i]_CH[x]_CN0.CN0** will not be incremented once the channel is enabled. Instead, according to the configuration bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG** , the start of counting and pulse generation is triggered by a write access of **ATOM[i]_CH[x]_CN0.CN0** or the internal trigger signal *ATOM_CH_TRIGIN* [x:x] or the external trigger signal *EXT_TRIGIN* [x:x].

When **ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG** =0, a write access to the register **ATOM[i]_CH[x]_CN0.CN0** triggers the start of incrementing the counter register **ATOM[i]_CH[x]_CN0.CN0** and pulse generation. If the time between the write access and the first edge generation of one-short PWM signal is defined as initial delay, the initial delay is determined by the value of (**ATOM[i]_CH[x]_CM0.CM0** - **ATOM[i]_CH[x]_CN0.CN0**) multiplied with the selected CMU clock period. Especially, if writing a value of greater than or equal to **ATOM[i]_CH[x]_CM0.CM0** -1 to **ATOM[i]_CH[x]_CN0.CN0** and configuring the initial clock source as cluster clock (the clock source at reset state), the initial delay is minimum, i.e. 1 cluster clock period *CLS[i]_CLK* . That means after the write access of **ATOM[i]_CH[x]_CN0.CN0** , the first edge of the output with the interrupt will be set in 1 *CLS[i]_CLK* period.

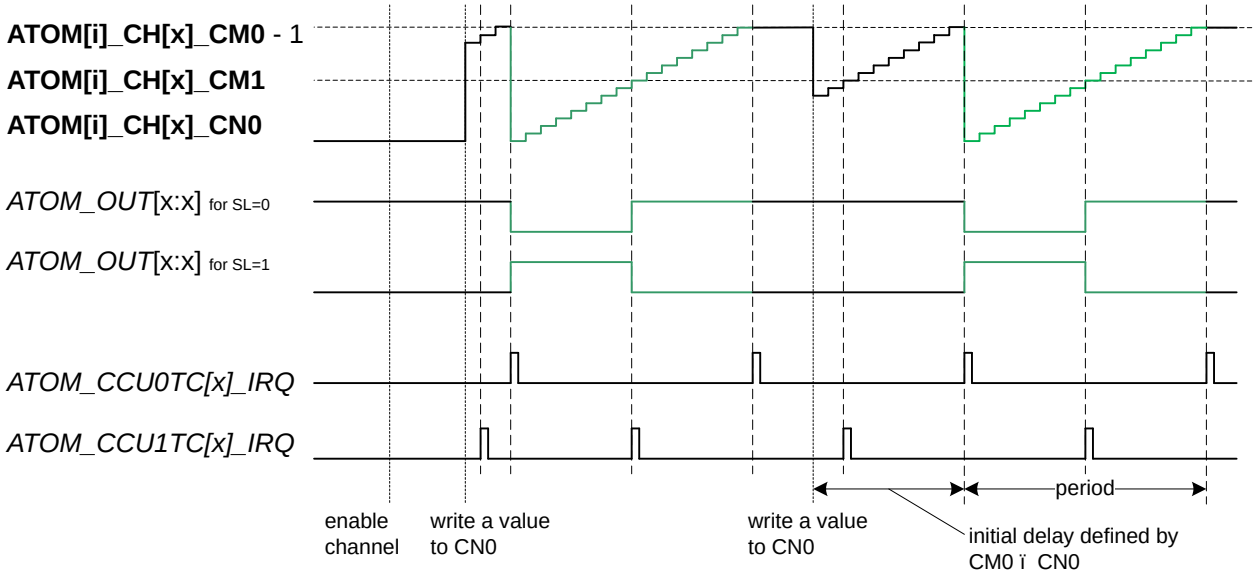
If the counter **ATOM[i]_CH[x]_CN0.CN0** is reset from **ATOM[i]_CH[x]_CM0.CM0** -1 back to zero, the first edge at *ATOM_OUT* is generated. And the operation phase of the pulse generation starts incrementing of **ATOM[i]_CH[x]_CN0.CN0** from 0 to **ATOM[i]_CH[x]_CM0.CM0** -1 again.

The second edge is generated, if **ATOM[i]_CH[x]_CN0.CN0** increments until it reaches **ATOM[i]_CH[x]_CM1.CM1** .

If the counter **ATOM[i]_CH[x]_CN0.CN0** reaches the value of **ATOM[i]_CH[x]_CM0.CM0** -1 at the second time, the counter will stop.

Figure *99 "PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 0"* depicts the pulse generation process mentioned above. The interrupt signals generation during PWM signal generation are similar as described in continuous counting up mode. If the value written to **ATOM[i]_CH[x]_CN0.CN0** for triggering or retriggering the counter is greater than or equal to **ATOM[i]_CH[x]_CM1.CM1** -1, an interrupt on *ATOM_CCU1TC[x]_IRQ* is generated after 1 selected CMU clock period. Similarly but not shown in the figure, if the value written to **ATOM[i]_CH[x]_CN0.CN0** for triggering or retriggering the counter is greater than or equal to **ATOM[i]_CH[x]_CM0.CM0** - 1, an interrupt on *ATOM_CCU0TC[x]_IRQ* is generated after 1 selected CMU clock period. Afterwards when **ATOM[i]_CH[x]_CN0.CN0** is reset from **ATOM[i]_CH[x]_CM0.CM0** -1 to 0, *ATOM_CCU0TC[x]_IRQ* of channel x is generated. When **ATOM[i]_CH[x]_CN0.CN0** is incrementing to reach **ATOM[i]_CH[x]_CM1.CM1** , *ATOM_CCU1TC[x]_IRQ* of channel x is generated.

Figure 99 PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 0



Retriggering an one-shot cycle (while **ATOM[i]_CH[x]_CN0.CN0** is already incrementing) by writing a value to **ATOM[i]_CH[x]_CN0.CN0** is possible but depends on the phase of **ATOM[i]_CH[x]_CN0.CN0** :

- phase 1: update of **ATOM[i]_CH[x]_CN0.CN0** before **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** for the first time (initial phase)
- phase 2: update of **ATOM[i]_CH[x]_CN0.CN0** after **ATOM[i]_CH[x]_CN0.CN0** has reached **ATOM[i]_CH[x]_CM0.CM0** for the first time until the current single PWM pulse is finished (operation phase)

In phase 1 (initial phase), writing a value CN0(new) to counter **ATOM[i]_CH[x]_CN0.CN0** (CN0(new)< **ATOM[i]_CH[x]_CM0.CM0**) leads to a shift of the first edge by the time **ATOM[i]_CH[x]_CM0.CM0** -CN0(new), i.e. the first edge is generated when **ATOM[i]_CH[x]_CN0.CN0** is reset from **ATOM[i]_CH[x]_CM0.CM0** -1 back to 0 for the first time.

In phase 2 (operation phase): writing to counter **ATOM[i]_CH[x]_CN0.CN0** is not possible and is protected by hardware.

A failed write access to **ATOM[i]_CH[x]_CN0.CN0** in phase 2 (operation phase) is indicated by the status register flag **ATOM[i]_CH[x]_STAT.OSM_RTF** .

After one PWM pulse is ended, other single pulses can be further triggered by a write access to register **ATOM[i]_CH[x]_CN0.CN0** .

It is recommended to enable the update mechanism by setting **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 0b10 for retriggering the second one-shot PWM signal generation with different initial delay and different PWM parameters. Similarly like continuous counting up mode with **ATOM[i]_CH[x]_CTRL.RST_CCU0** =0, the operation registers **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** / **ATOM[i]_CH[x]_CTRL.SL** / **ATOM[i]_CH[x]_CTRL.CLK_SRC** are always synchronously updated when **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** -1. Firstly, after the first one-shot PWM signal is generated, the new values for initial delay (phase 1) can be written into the shadow registers **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** / **ATOM[i]_CH[x]_CTRL_SR.SL_SR** / **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** before the retriggering is executed by writing of register **ATOM[i]_CH[x]_CN0.CN0** . The shadow registers should be loaded to the operation registers **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** / **ATOM[i]_CH[x]_CTRL.SL** / **ATOM[i]_CH[x]_CTRL.CLK_SRC** at the next falling edge of the selected CMU clock resolution signal, because **ATOM[i]_CH[x]_CN0.CN0** keeps as **ATOM[i]_CH[x]_CM0.CM0** -1. When **ATOM[i]_CH[x]_CN0.CN0** is written to be retriggered afterwards, the initial delay is defined according to the updated operation registers. Secondly, during the initial delay phase (phase 1), the new parameters for PWM signal can be written into the shadow registers again. The operational registers will be updated with the shadow registers when **ATOM[i]_CH[x]_CN0.CN0** is reset from **ATOM[i]_CH[x]_CM0.CM0** -1 at the end of phase 1 and phase 2 is started to generate the second one-shot PWM signal with the new updated operation registers.

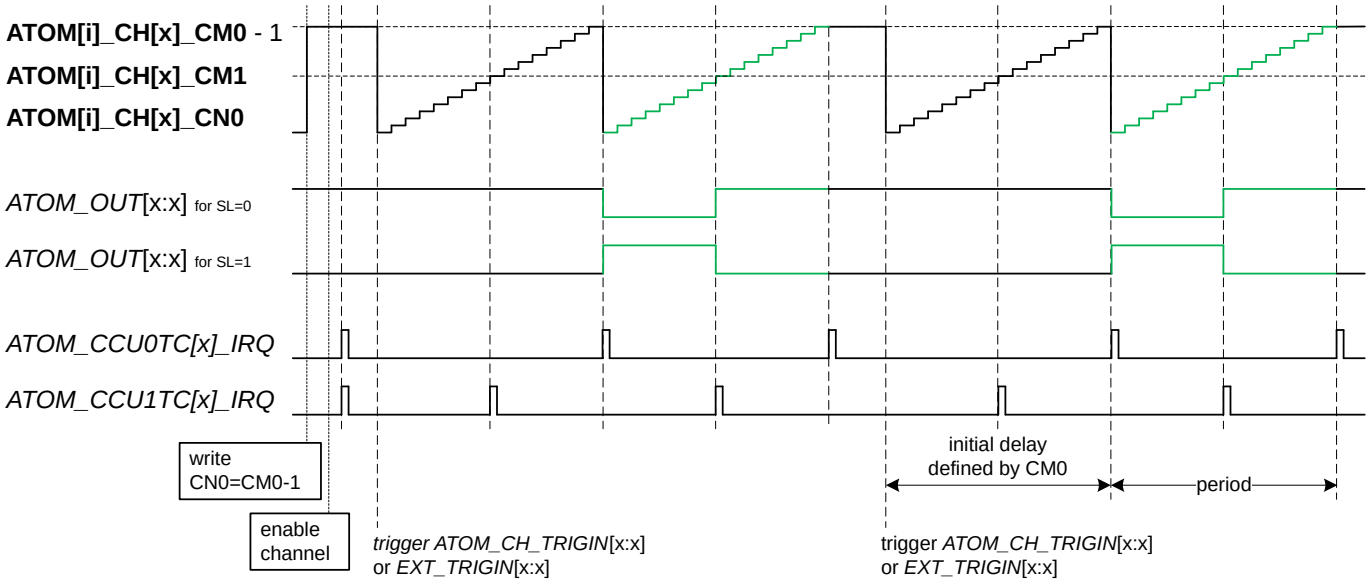
If a channel is configured to one-shot mode and configuration bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG** is set to 1, the trigger signal pulse *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN* [x:x] triggers start of pulse generation as shown in figure *100 "PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 1"* . In this case, it is not allowed to write **ATOM[i]_CH[x]_CN0.CN0** and so the register **ATOM[i]_CH[x]_STAT.OSM_RTF** is not relevant. Otherwise, a write access to **ATOM[i]_CH[x]_CN0.CN0** can make an unpredictable output behavior. Furthermore, as shown in figure *100 "PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 1"* , **ATOM[i]_CH[x]_CN0.CN0** should be initially configured as **ATOM[i]_CH[x]_CM0.CM0** -1 or greater value before enabling the channel. Otherwise, it is hard to define the counting behavior and the initial delay behavior. As shown in figure *100 "PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 1"* , when the channel is enabled, after the falling edge of the trigger pulse *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN_S* (the synchronized signal of *EXT_TRIGIN* [x:x], see chapter *15.2.2 "External Trigger Interface"*), **ATOM[i]_CH[x]_CN0.CN0** will be triggered to be reset that is synchronous with the next enable high pulse of the selected CMU clock source (i.e. synchronous to the next falling edge of the selected CMU enable signal). Then it counts from 0 to **ATOM[i]_CH[x]_CM0.CM0** -1. This process is defined as initial phase. If the internal trigger signal *ATOM_CH_TRIGIN* [x:x] is in use and it is synchronous to the selected CMU clock source or the external trigger signal *EXT_TRIGIN* [x:x] is in use and it must be synchronized to the selected CMU clock source before triggering, the initial delay can be defined as the time between the falling edge of the synchronized trigger signal to the time point of starting generating the PWM signal (green phase in the figure), i.e. **ATOM[i]_CH[x]_CM0.CM0** +1 selected CMU

clock periods. Then **ATOM[i]_CH[x]_CN0.CN0** will be reset again and count from 0 to **ATOM[i]_CH[x]_CM0.CM0** -1 at the second time to generate one signal PWM signal and then stop. This second counting is defined as operation phase.

After the first OSM signal is generated, the second trigger signal is allowed to be applied to start the counter and generate the next OSM signal after a certain initial delay. Figure *100 "PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 1"* shows such an application of generating 2 OSM signals without changing PWM parameters. In the case, after the second trigger signal is coming, the counter has the same behavior as described above for the first OSM signal generation. That means the initial delay is also **ATOM[i]_CH[x]_CM0.CM0** +1 selected CMU clock periods and the OSM signal is the same as the first one.

However, in some real applications of generating various single PWM signals, it is necessary to synchronously update the parameters in **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** / **ATOM[i]_CH[x]_CTRL.SL** / **ATOM[i]_CH[x]_CTRL.CLK_SRC** by setting **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 0b10. In this case, these registers are updated from their shadow registers when **ATOM[i]_CH[x]_CN0.CN0** is matching **ATOM[i]_CH[x]_CM0.CM0** -1. After the first OSM PWM signal, if **ATOM[i]_CH[x]_CM0.CM0** is updated with a new value CM0(new) before the second trigger signal comes, the initial delay of the second OSM signal depends on CM0(new) and the value CN0(old) of **ATOM[i]_CH[x]_CN0.CN0** when the counter stopped at the end of the first OSM signal. The value CN0(old) is actually the value of **ATOM[i]_CH[x]_CM0.CM0** -1 of the first OSM signal before the synchronous update, i.e. CN0(old)=CM0(old)-1. Like the first OSM signal generation, with the falling edge of the second trigger pulse signal, the counter will be retriggered synchronously to the next enable high pulse of the selected CMU clock source (i.e. synchronous to the next falling edge of the selected CMU enable signal). Assuming the internal trigger signal *ATOM_CH_TRIGIN* [x:x] is in use and it is synchronous to the selected CMU clock source or the external trigger signal *EXT_TRIGIN* [x:x] is in use and it will be synchronized to the selected CMU clock source before triggering, the initial delay can be defined as the time between the falling edge of the second synchronized trigger signal to the time point of starting generating the PWM signal (green phase in the figure). On the one hand, if CN0(old) is greater than or equal to updated CM0(new)-1, the counter will be reset and counts up to CM0(new)-1 and the initial delay is CM0(new)+1 selected CMU clock periods, i.e. **ATOM[i]_CH[x]_CM0.CM0** +1 selected CMU clock periods. The behavior of the counter is the same as described above for the initial delay of the first OSM signal. On the other hand, if CN0(old) is smaller than updated CM0(new)-1, the initial phase or the initial delay is longer. In the initial phase, the counter will count up from CN0(old) to CM0(new)-1 and reset, and then increment again to CM0(new)-1. That means the initial delay is no longer CM0(new)+1, but the larger value of 2*CM0(new)-CN0(old) CMU clock periods, i.e. 2*CM0(new)-CM0(old)+1 selected CMU clock periods.

Figure 100 PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 1



As shown in figure *100 "PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 1"* , the interrupt signals generation during PWM signal generation are similar as described in continuous counting up mode. As before enabling the channel the initial value written to **ATOM[i]_CH[x]_CN0.CN0** is greater than or equal to **ATOM[i]_CH[x]_CM0.CM0** - 1 and **ATOM[i]_CH[x]_CM1.CM1** - 1, both interrupt signals *ATOM_CCU0TC[x]_IRQ* and *ATOM_CCU1TC[x]_IRQ* are generated after 1 selected CMU clock period. Afterwards when **ATOM[i]_CH[x]_CN0.CN0** is reset from **ATOM[i]_CH[x]_CM0.CM0** -1 to 0, *ATOM_CCU0TC[x]_IRQ* of channel x is generated. When **ATOM[i]_CH[x]_CN0.CN0** is incrementing to reach **ATOM[i]_CH[x]_CM1.CM1** , *ATOM_CCU1TC[x]_IRQ* of channel x is generated.

In one-shot mode, it is not recommended to configure **ATOM[i]_CH[x]_CM0.CM0** <2. Otherwise, the behavior may be unexpected.

15.3.3.7 One-shot Counting Up-Down Mode

The ATOM channel can operate in one-shot counting up-down mode when the bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM** = 1 and the **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** != 0b00. One-shot mode means that a single pulse with the pulse level defined in bit **ATOM[i]_CH[x]_CTRL_SOMP.SL** is generated on the output signal. In this case, ARU interface communication is not supported, i.e. **ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** must be configured as 0.

Firstly, the channel has to be enabled by setting the corresponding **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[k]** .

In one-shot mode the counter **ATOM[i]_CH[x]_CN0.CN0** will not be incremented once the channel is enabled.

When **ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG** =0, a write access to the register **ATOM[i]_CH[x]_CN0.CN0** triggers the start of incrementing the counter register **ATOM[i]_CH[x]_CN0.CN0** and pulse generation.

If the counter **ATOM[i]_CH[x]_CN0.CN0** is greater than or equal to **ATOM[i]_CH[x]_CM1.CM1** , the output *ATOM_OUT* is set to **ATOM[i]_CH[x]_CTRL_SOMP.SL** value.

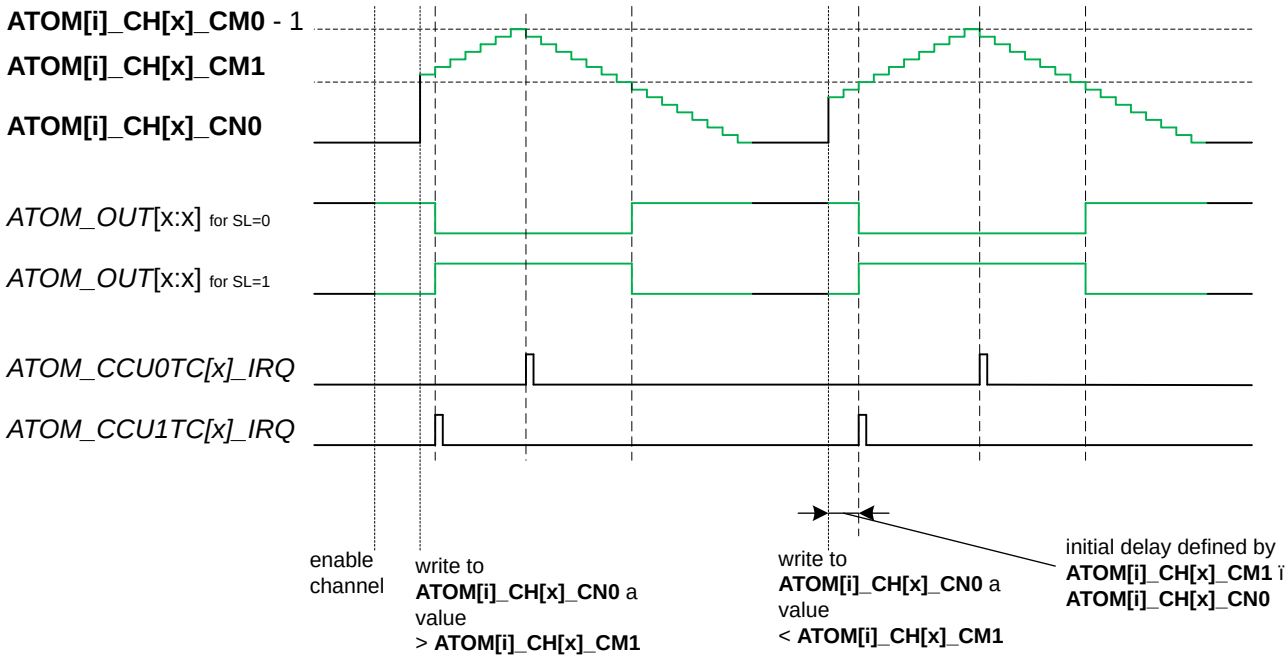
If the counter **ATOM[i]_CH[x]_CN0.CN0** is less than **ATOM[i]_CH[x]_CM1.CM1** , the output *ATOM_OUT* is set to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** value.

If the counter **ATOM[i]_CH[x]_CN0.CN0** has reached the value 0 (by counting down), it will stop.

The written value of **ATOM[i]_CH[x]_CN0.CN0** determines the start delay of the first edge. The delay time of the first edge is given by (**ATOM[i]_CH[x]_CM1.CM1** - **ATOM[i]_CH[x]_CN0.CN0**) multiplied with the period defined by current value of **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** .

Figure *101 "PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up-down mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 0"* depicts the pulse generation in SOMP one-shot counting up-down mode. The interrupt signals are generated similarly as described in one-shot counting up mode and continuous counting up-down mode. The write access of **ATOM[i]_CH[x]_CN0.CN0** can cause interrupts if the write value is greater than or equal to **ATOM[i]_CH[x]_CM1.CM1** - 1 or **ATOM[i]_CH[x]_CM0.CM0** -1. If the value written to **ATOM[i]_CH[x]_CN0.CN0** for triggering or retriggering the counter is greater than or equal to **ATOM[i]_CH[x]_CM1.CM1** -1, the output *ATOM_OUT* [x:x] will be set to **ATOM[i]_CH[x]_CTRL_SOMP.SL** after 1 selected CMU clock period together with an interrupt on *CCU1TC[x]_IRQ* . Similarly but not shown in the figure, if the value written to **ATOM[i]_CH[x]_CN0.CN0** for triggering or retriggering the counter is greater than or equal to **ATOM[i]_CH[x]_CM0.CM0** - 1, an interrupt on *ATOM_CCU0TC[x]_IRQ* will be generated after 1 selected CMU clock period. Afterwards when **ATOM[i]_CH[x]_CN0.CN0** is incrementing to reach **ATOM[i]_CH[x]_CM1.CM1** , an interrupt on *ATOM_CCU1TC[x]_IRQ* is generated. When **ATOM[i]_CH[x]_CN0.CN0** changes its counting direction from up to down, an interrupt on *ATOM_CCU0TC[x]_IRQ* is generated.

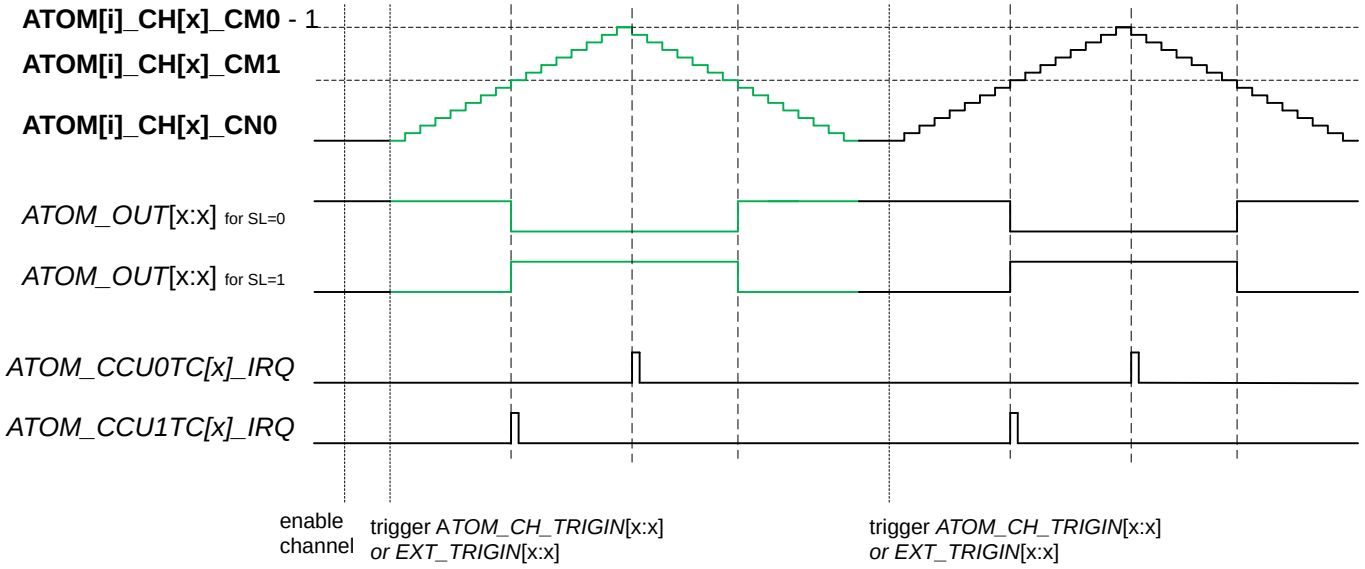
Figure 101 PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up-down mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 0



Further output of single pulses can be started by writing to register **ATOM[i]_CH[x]_CN0.CN0** .

If a channel is configured to one-shot counting up-down mode and configuration bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG** is set to 1, the falling edge of the trigger signal *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN_S* (the synchronized signal of *EXT_TRIGIN* [x:x]) triggers start of incrementing **ATOM[i]_CH[x]_CN0.CN0** and pulse generation. The output generation with interrupt signals is similar as mentioned above and shown in figure *102 "PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up-down mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 1"* . If the counter **ATOM[i]_CH[x]_CN0.CN0** is greater than or equal to **ATOM[i]_CH[x]_CM1.CM1** , the output *ATOM_OUT* is set to **ATOM[i]_CH[x]_CTRL_SOMP.SL** value. Otherwise, the output *ATOM_OUT* is set to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** value. When **ATOM[i]_CH[x]_CN0.CN0** is incrementing to reach **ATOM[i]_CH[x]_CM1.CM1** , an interrupt on *ATOM_CCU1TC[x]_IRQ* is generated. When **ATOM[i]_CH[x]_CN0.CN0** changes its counting direction from up to down, an interrupt on *ATOM_CCU0TC[x]_IRQ* is generated.

Figure 102 PWM Output with respect to configuration bit ATOM[i]_CH[x]_CTRL_SOMP.SL in one-shot counting up-down mode if ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG= 1



15.3.3.8 Pulse Count Modulation Mode

At the output *ATOM_OUT* a pulse count modulated signal can be generated instead of the simple PWM output signal in SOMP mode. This specific behavior is called pulse count modulation (PCM) mode. PCM mode is only available in SOMP mode when **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** =0.

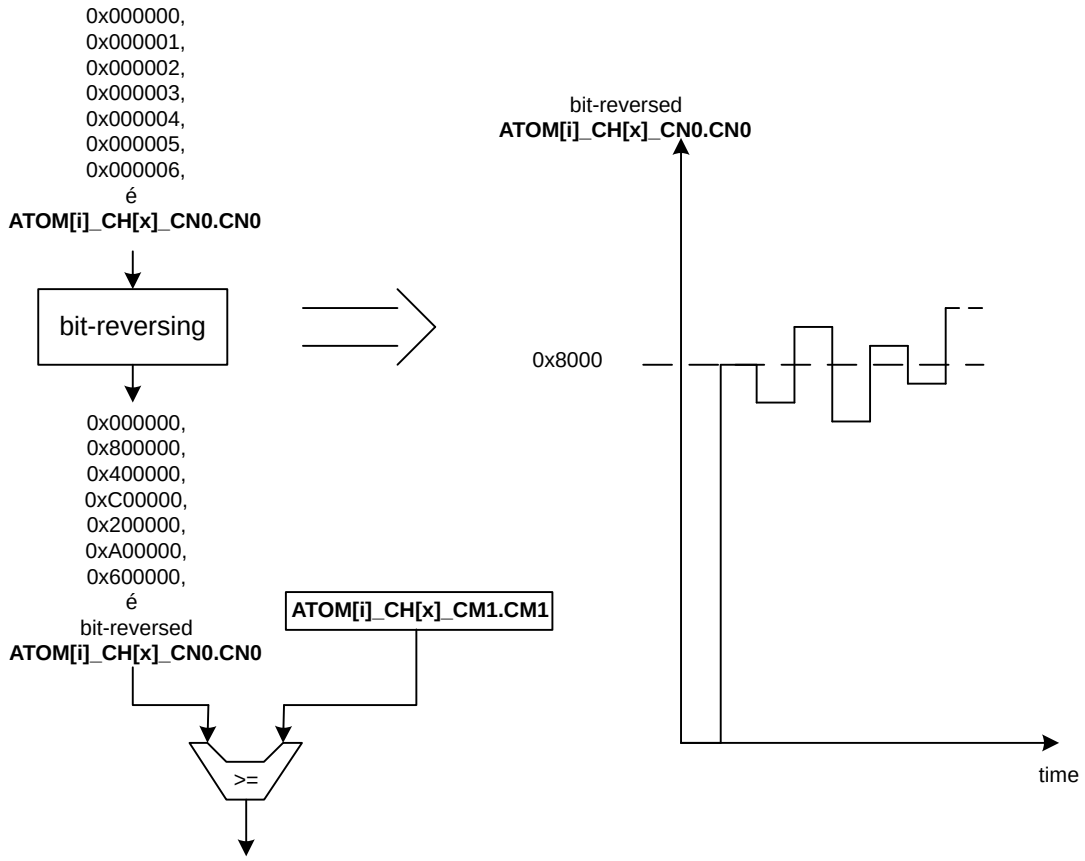
PCM mode is enabled by setting bit **ATOM[i]_CH[x]_CTRL_SOMP.BITREV** [6:6] to 1.

PCM mode is only available for odd numbered ATOM channels 1,3,5,7.

With the configuration bit **ATOM[i]_CH[x]_CTRL_SOMP.BITREV** =1 a bit-reversing of the counter output **ATOM[i]_CH[x]_CN0.CN0** is configured. In this case the bits LSB and MSB are swapped, the bits LSB+1 and MSB-1 are swapped, the bits LSB+2 and MSB-2 are swapped and so on.

The effect of bit-reversing of the **ATOM[i]_CH[x]_CN0.CN0** register value is shown in the following figure *103 "Bit reversing of counter ATOM[i]_CH[x]_CN0.CN0 output "* .

Figure 103 Bit reversing of counter ATOM[i]_CH[x]_CN0.CN0 output



In the PCM mode the counter register **ATOM[i]_CH[x]_CN0.CN0** is incremented by every clock tick depending on configured CMU clock resolution (*CCM[i]_CLK_RES*).

The output of counter register **ATOM[i]_CH[x]_CN0.CN0** is first bit-reversed and then compared with the configured register value **ATOM[i]_CH[x]_CM1.CM1** .

When the bit-reversed value of register **ATOM[i]_CH[x]_CN0.CN0** is greater than or equal to **ATOM[i]_CH[x]_CM1.CM1** , the output *ATOM_OUT* [x:x] is set to ! **ATOM[i]_CH[x]_CTRL.SL** and otherwise set to **ATOM[i]_CH[x]_CTRL.SL** .

In PCM mode the **ATOM[i]_CH[x]_CM0.CM0** register – in which the period is defined – normally has to be set to its maximum value 0xFFFFFF.

To reduce time period of updating pulse width value in **ATOM[i]_CH[x]_CM1.CM1** register, it is additionally possible to setup period value in **ATOM[i]_CH[x]_CM0.CM0** register to smaller values than maximum value as described before.
Possible values for **ATOM[i]_CH[x]_CM0.CM0** register are each even numbered values to the power of 2 e.g. 0x800000, 0x400000, 0x200000 ... .
In this case the pulse width has to be configured in the following manner.
Depending on how much the period in **ATOM[i]_CH[x]_CM0.CM0** register is decreased – means shifted right starting from 0x1000000 – the pulse width in **ATOM[i]_CH[x]_CM1.CM1** register has to be shifted to the left (= rotated: shift MSB back into LSB) with same value, e.g.: period **ATOM[i]_CH[x]_CM0.CM0** = 0x0010000 -> shifted 8 bits right from 0x1000000
--> so pulse width has to be shifted 8 bits left:
e.g. 50% duty cycle: pulse width = 0x0008000 -> shift 8 bits left -> **ATOM[i]_CH[x]_CM1.CM1** = 0x800000

Table 36 PCM examples

ATOM[i]_CH[x]_CM0.CM0 (period)	ATOM[i]_CH[x]_CM1.CM1 (pulse width)	shift	ATOM[i]_CH[x]_CM1.CM1 (configured value)
0xFFFFFF	0x800000	no shift	0x800000
0x800000	0x400000	shift 1 bit left	0x800000
0x400000	0x100000	shift 2 bits left	0x400000
0x200000	0x0FFFFF	shift 3 bits left	0x7FFFF8
0x100000	0x033333	shift 4 bits left	0x333330
0x080000	0x005555	shift 5 bits left	0x0AAAA0
0x000020	0x000008	shift 19 bits left	0x400000

ATOM[i]_CH[x]_CM0.CM0 (period)	ATOM[i]_CH[x]_CM1.CM1 (pulse width)	shift	ATOM[i]_CH[x]_CM1.CM1 (configured value)
0x000010	0x000005	shift 20 bits left	0x500000

In this mode the interrupt **ATOM[i]_CH[x]_IRQ_NOTIFY.CCU[1]TC** is set every time, when bit reverse value of **ATOM[i]_CH[x]_CN0.CN0** is greater than or equal to **ATOM[i]_CH[x]_CM1.CM1** which may be multiple times during one period. Therefore, from application point of view it is not useful to enable this interrupt.

15.3.3.9 Trigger generation

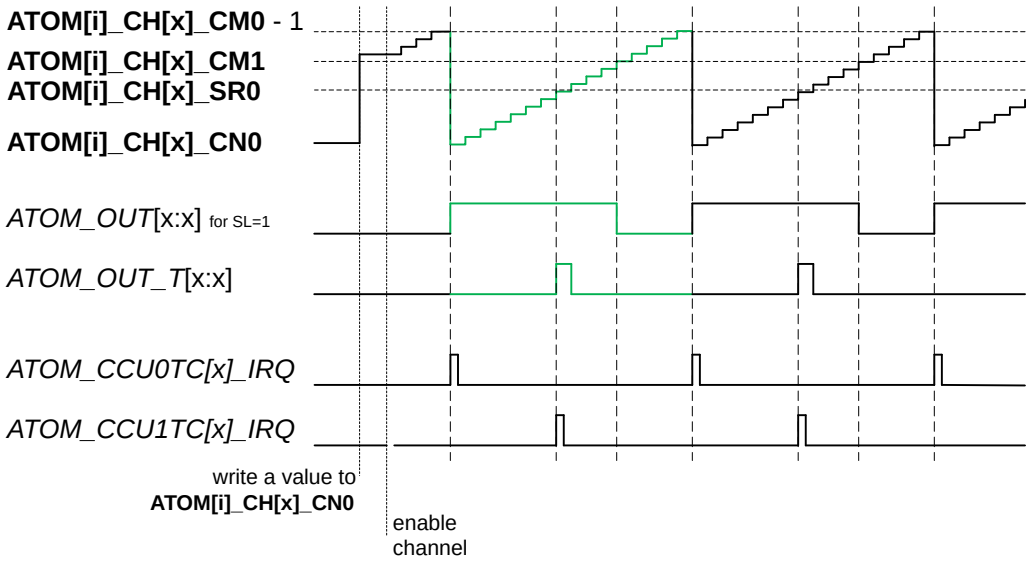
For applications with constant PWM period defined by **ATOM[i]_CH[x]_CM0.CM0** , it is not necessary to update the **ATOM[i]_CH[x]_CM0.-CM0** register regularly with **ATOM[i]_CH[x]_SR0.SR0** register. For these applications, the **ATOM[i]_CH[x]_SR0.SR0** register can be used to define an additional output signal and interrupt trigger.
If bit **ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG** is set to 1, the register **ATOM[i]_CH[x]_SR0.SR0** is no longer used as a shadow register for register **ATOM[i]_CH[x]_CM0.CM0** . Instead, **ATOM[i]_CH[x]_SR0.SR0** is compared against **ATOM[i]_CH[x]_CN0.CN0** . If both are equal, a pulse of signal level '1' is generated at the output *ATOM_OUT_T* [x:x] of instance i channel x. In this case, **ATOM[i]_CH[x]_CM1.CM1** will still be synchronously updated from its shadow register **ATOM[i]_CH[x]_SR1.SR1** if the update is enabled by setting **ATOM[i]_AGC_GLB_-CTRL.UPEN_CTRL[k]** =0b10.
The bit **ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG** can only be set if bit **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** of this channel is 0.

If **ATOM[i]_CH[x]_CTRL_SOMP.ARU_EN** =1 and both **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are updated via ARU, the new **ATOM[i]_CH[x]_SR0.SR0** value is used immediately after update. Update of **ATOM[i]_CH[x]_SR0.SR0** via ARU can be suppressed by the register bitfield configuration of **ATOM[i]_CH[x]_CTRL_SOMP.ADL** .

If bit **ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG** =1 is set, the interrupt on *ATOM_CCU1TC[x]_IRQ* is set at the same time with pulse generation on *ATOM_OUT_T* in case of a compare equal match of **ATOM[i]_CH[x]_SR0.SR0** = **ATOM[i]_CH[x]_CN0.CN0** . That means when the interrupt notify flag **ATOM[i]_CH[x]_IRQ_NOTIFY.CCU[1]TC** is no longer set on a compare match of **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_CN0.CN0** . Instead, the **ATOM[i]_CH[x]_IRQ_NOTIFY.CCU[1]TC** interrupt notify flag is set in case of a compare equal match of **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_CN0.CN0** . However, the interrupt generation of *ATOM_CCU0TC[x]_IRQ* is the same as the case of **ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG** =0. The output generation with interrupt signals are shown in figure *104 "Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.TRIG_PULSE=1 in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG=1"* .

With configuration bit **ATOM[i]_CH[x]_CTRL_SOMP.TRIG_PULSE** , it is configurable if the output *ATOM_OUT_T* of instance i and channel x is high as long as **ATOM[i]_CH[x]_CN0.CN0** = **ATOM[i]_CH[x]_SR0.SR0** (**ATOM[i]_CH[x]_CTRL_SOMP.TRIG_PULSE** =0) or if there will be only one pulse of length one cluster clock period when **ATOM[i]_CH[x]_CN0.CN0** becomes **ATOM[i]_CH[x]_SR0.SR0** (**ATOM[i]_C-H[x]_CTRL_SOMP.TRIG_PULSE** =1). Figure *104 "Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.TRIG_PULSE=1 in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG=1"* shows the case of **ATOM[i]_CH[x]_CTRL_SOMP.TRIG_PULSE** =1.

Figure 104 Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.TRIG_PULSE=1 in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG=1



15.3.3.10 High resolution PWM support (HRES mode)

Similar to TOM submodule, ATOM also provides high resolution PWM support mode (HRES mode). In HRES mode, ATOM offers the output signal *ATOM_OUT_HRES[x]* in addition to the PWM output signal *ATOM_OUT* [x:x] to control a circuit, e.g. a delay chain, which is needed to realize the high resolution PWM and which has to be implemented outside the GTM-IP. Please refer to chapter *3.8 "High Resolution PWM Support"* for additional information of the high resolution PWM support of the GTM-IP.

High resolution mode in ATOM submodule is only available in SOMP continuous counting up mode (**ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** =0, **ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG** =0). HRES mode is not supported in SOMP continuous counting up-down mode or when **ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG** =1.

The high resolution PWM support is not available in ATOM SOMP one-shot mode. If one-shot mode is enabled by setting of **ATOM[i]_CH[x]_CTRL_SOMP.OSM** to 1, the high resolution PWM support has to be disabled by setting of **ATOM[i]_CH[x]_CTRL2.HRES** to 0, because it is not supported.

The high resolution PWM support is also not available in PCM mode (**ATOM[i]_CH[x]_CTRL_SOMP.BITREV** =1).

This output signal *ATOM_OUT_HRES[x]* has a width of five bits while the decimal value of these five bits defines the number of micro-steps, by which the output signal *ATOM_OUT* [x:x] will be delayed in a suitable circuit (e.g. delay chain) outside the GTM-IP.

The high resolution support will be enabled by setting bit **ATOM[i]_CH[x]_CTRL2.HRES** = 1.

If HRES mode is switched off by setting the configuration bit **ATOM[i]_CH[x]_CTRL2.HRES** to zero, then the output signals *ATOM_OUT_HRES[x]* and *ATOM_OUT_T_HRES[x]* are clamped to zero.

The value for the PWM period, stored in the registers **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_SR0.SR0** , and pulse width, stored in the registers **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_SR1.SR1** , represent respectively the length of period and pulse width in number of cluster clock steps and micro-steps.

Hereby the number of cluster clock steps for the PWM output signal *ATOM_OUT* [x:x] are defined by the upper bits [23:5], whereas the lower 5 bits [4:0] determine the additional micro-steps, by which the output signal has to be delayed outside the GTM-IP. This is shown in figure *105 "Period and duty-cycle register in HRES mode"* .

Figure 105 Period and duty-cycle register in HRES mode

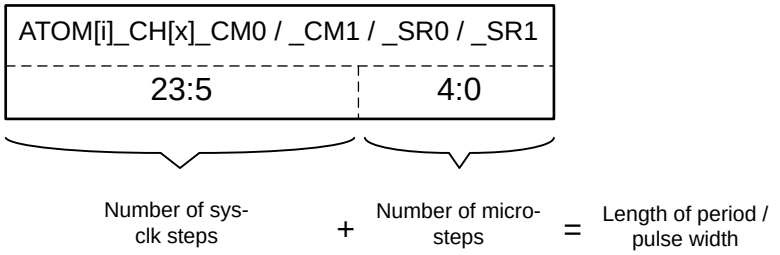
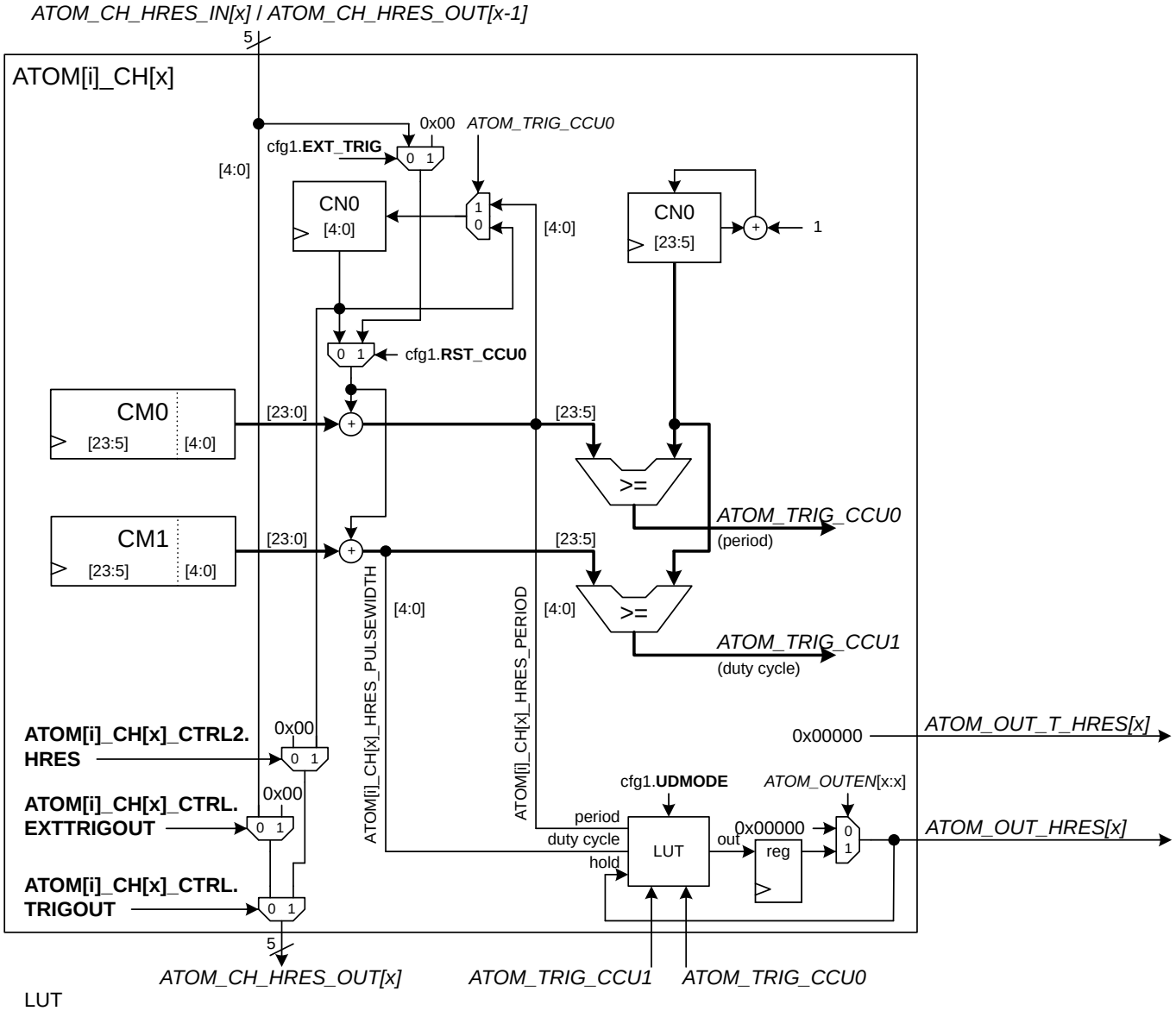


Figure *106 "High-level schematic view of ATOM_OUT_HRES[x] calculation"* shows the high-level schematic of the generation of the output signal *ATOM_OUT_HRES[x]* , which defines the number of micro-steps mentioned above. If the ATOM channel x has HRES support with the configuration of x<ATOM_HIGH_RES[i], a HRES trigger chain of *ATOM_CH_HRES_IN[x]* / *ATOM_CH_HRES_OUT[x]* is available together with the trigger chain of *ATOM_CH_TRIGIN* [x:x]/ *ATOM_CH_TRIGOUT* [x:x]. As shown in figure *79 "ATOM block diagram"* , the HRES trigger chain signals are routed through all ATOM channels and instances that have HRES support. ATOM0 channel 0 is driven by the output of ATOM0 channel 7 with a register in the trigger chain, which means a delay of one cluster clock period. The pipeline registers in the trigger chain of *ATOM_CH_TRIGIN* [x:x] / *ATOM_CH_TRIGOUT* [x:x] as mentioned in chapter *15.2.3 "Internal Trigger Interface"* are also inserted in the trigger chain of *ATOM_CH_HRES_IN[x]* / *ATOM_CH_HRES_OUT[x]* .

The channels x without HRES support (x>=ATOM_HRES_RES[i]) still have the HRES trigger chain of *ATOM_CH_HRES_IN[x]* / *ATOM_CH_HRES_OUT[x]* across these channels. The *ATOM_CH_HRES_IN[x]* coming from *ATOM_CH_HRES_OUT[x-1]* is directly connected to *ATOM_CH_HRES_OUT[x]* that is connected to *ATOM_CH_HRES_IN[x+1]* . There is no specific HRES functionalities in these channels without HRES support. Both HRES output signals *ATOM_OUT_HRES[x]* and *ATOM_OUT_T_HRES[x]* of these channels (x>=ATOM_HIGH_RES[i]) are clamped to zero.

When **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM0.CM0** -1, *ATOM_TRIG_CCU0* =1 will be set and then *ATOM_OUT* [x:x] will be set to **ATOM[i]_CH[x]_CTRL.SL** in the next CMU period. Similarly, when **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM1.CM1** -1, *ATOM_TRIG_CCU1* =1 will be set and then *ATOM_OUT* [x:x] will be set to ! **ATOM[i]_CH[x]_CTRL.SL** . When **ATOM[i]_CH[x]_CN0.CN0** < **ATOM[i]_CH[x]_CM0.CM0** -1, *ATOM_TRIG_CCU0* will be reset as 0. Similarly, when **ATOM[i]_CH[x]_CN0.CN0** < **ATOM[i]_CH[x]_CM1.CM1** -1, *ATOM_TRIG_CCU1* will be reset to 0. According to the unit LUT in Figure *106 "High-level schematic view of ATOM_OUT_HRES[x] calculation"* , the corresponding output signal *ATOM_OUT_HRES[x]* is generated together with each edge of PWM output signal *ATOM_OUT* [x:x].

Figure 106 High-level schematic view of ATOM_OUT_HRES[x] calculation



LUT

UDMODE	ATOM_TRIG_CCU0	ATOM_TRIG_CCU1	out
0	0	0	hold
0	0	1	ATOM[i]_CH[x]_HRES_PULSEWIDTH
0	1	-	ATOM[i]_CH[x]_HRES_PERIOD

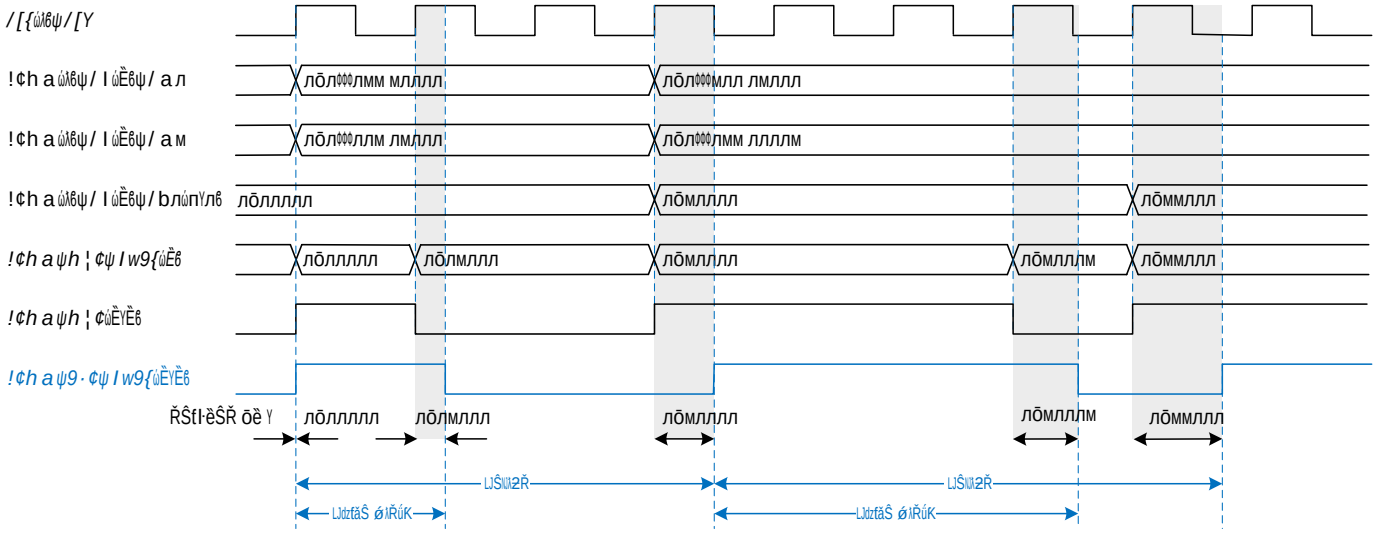
cfg1: **ATOM[i]_CH[x]_CTRL**

As shown in figure *106 "High-level schematic view of ATOM_OUT_HRES[x] calculation"* , when it is configured as continuous counting up mode (**ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** =0) and **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0, the output signal *ATOM_OUT_HRES[x]* has to be calculated based on the internal signal *ATOM[i]_CH[x]_HRES_PERIOD* and *ATOM[i]_CH[x]_HRES_PULSEWIDTH* in the following manner (here, t means the index of a PWM period):

- period: *ATOM[i]_CH[x]_HRES_PERIOD* = **ATOM[i]_CH[x]_CN0.CN0** [4:0] + **ATOM[i]_CH[x]_CM0.CM0** [4:0]
- pulse width: *ATOM[i]_CH[x]_HRES_PULSEWIDTH* = **ATOM[i]_CH[x]_CN0.CN0** [4:0] + **ATOM[i]_CH[x]_CM1.CM1** [4:0]
- with : (**ATOM[i]_CH[x]_CN0.CN0** [4:0])$_{t+1}$ = (**ATOM[i]_CH[x]_CN0.CN0** [4:0])$_{t}$ + (**ATOM[i]_CH[x]_CM0.CM0** [4:0])$_{t}$

The signal diagram in figure *107 "Signals in HRES mode when ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0 and ATOM[i]_CH[x]_CTRL_SOMP.SL=1"* shows an example for the output generation in HRES mode when **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0 and **ATOM[i]_CH[x]_CTRL_SOMP.SL** =1. The PWM output signal *ATOM_OUT* [x:x] with cluster clock steps is generated based on the upper bits [23:5] of **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** according to the same way of continuous counting up mode. Please refer to figure *95 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0"* for more details. The output signal *ATOM_OUT_HRES[x]* can be calculated based on the lower bits [4:0] of **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** according to figure *106 "High-level schematic view of ATOM_OUT_HRES[x] calculation"* and the formulas above. As shown in figure *107 "Signals in HRES mode when ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0 and ATOM[i]_CH[x]_CTRL_SOMP.SL=1"* , the value of *ATOM_OUT_HRES[x]* represents the number of micro-steps that should be taken by the external circuit to delay each edge of *ATOM_OUT* [x:x]. In this way, the delayed PWM signal *ATOM_EXT_HRES* [x:x] achieves higher resolution.

Figure 107 Signals in HRES mode when ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0 and ATOM[i]_CH[x]_CTRL_SOMP.SL=1



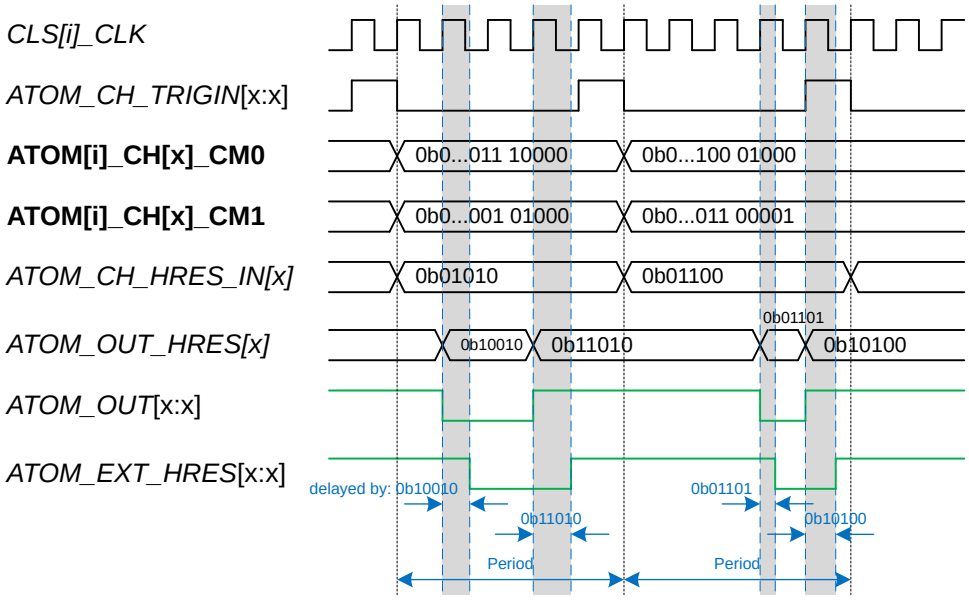
The HRES mode is also available in continuous counting up mode when **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1.

In this case (**ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** = 1), if **ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG** =0, the internal trigger signal *ATOM_CH_HRES_IN[x]* from the preceding channel is used for calculation instead of **ATOM[i]_CH[x]_CN0.CN0** [4:0] :

- period: *ATOM[i]_CH[x]_HRES_PERIOD* = *ATOM_CH_HRES_IN[x]* + **ATOM[i]_CH[x]_CM0.CM0** [4:0]
- pulse width: *ATOM[i]_CH[x]_HRES_PULSEWIDTH* = *ATOM_CH_HRES_IN[x]* + **ATOM[i]_CH[x]_CM1.CM1** [4:0]

The signal diagram in figure *108 "Signals in HRES mode when ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1, ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG=0 and ATOM[i]_CH[x]_CTRL_SOMP.SL=1"* shows an example for the output generation in HRES mode when **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1, **ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG** =0 and **ATOM[i]_CH[x]_CTRL_SOMP.SL** =1. The PWM output signal *ATOM_OUT* [x:x] with cluster clock steps is generated based on the upper bits [23:5] of **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** according to the same way of continuous counting up mode. Please refer to figure *96 "PWM Output behavior with respect to the ATOM[i]_CH[x]_CTRL_SOMP.SL bit in continuous counting up mode if ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1"* for more details. The output signal *ATOM_OUT_HRES[x]* representing the number of micro-steps can be calculated based on the lower bits [4:0] of **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and *ATOM_CH_HRES_IN[x]* according to figure *106 "High-level schematic view of ATOM_OUT_HRES[x] calculation"* and the formulas above. Then the external circuit can take the value of *ATOM_OUT_HRES[x]* to generate the delayed signal *ATOM_EXT_HRES* [x:x].

Figure 108 Signals in HRES mode when ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1, ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG=0 and ATOM[i]_CH[x]_CTRL_SOMP.SL=1

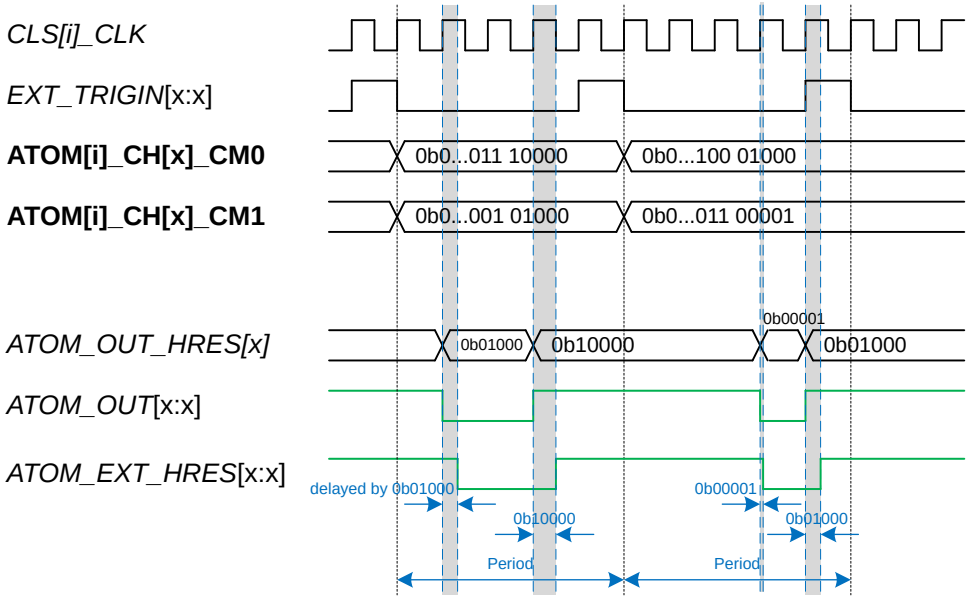


If **ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG** =1, the HRES output signal is only depending on **ATOM[i]_CH[x]_CM0.CM0** [4:0] and **ATOM[i]_CH[x]_CM1.CM1** [4:0]:

- period: *ATOM[i]_CH[x]_HRES_PERIOD* = 0b00 + **ATOM[i]_CH[x]_CM0.CM0** [4:0]
- pulse width: *ATOM[i]_CH[x]_HRES_PULSEWIDTH* = 0b00 + **ATOM[i]_CH[x]_CM1.CM1** [4:0]

The signal diagram in figure *109 "Signals in HRES mode when ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1, ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG=1 and ATOM[i]_CH[x]_CTRL_SOMP.SL=1"* shows an example for the output generation in HRES mode when **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1, **ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG** =1 and **ATOM[i]_CH[x]_CTRL_SOMP.SL** =1. Similarly, *ATOM_OUT* [x:x] is normal continuous counting up mode PWM signal with cluster clock steps, while *ATOM_OUT_HRES[x]* represents the number of micro-steps for generating *ATOM_EXT_HRES* [x:x].

Figure 109 Signals in HRES mode when ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=1, ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG=1 and ATOM[i]_CH[x]_CTRL_SOMP.SL=1



The following restrictions and characteristics must be considered, if the HRES mode is enabled:

- The selected clock resolution source **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** of *CCM[i]_CLK_RES* [x:x] has to be configured to the cluster clock *CLS[i]_CLK* .
- The maximum possible period is reduced to 2^{19} cluster clock cycles.
- Only one edge on *ATOM_OUT* [x:x] with one value of *ATOM_OUT_HRES[x]* per cluster clock cycle is possible.

In order to fulfill the constraint above, there are more constraints when ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0=0:

- For normal PWM signal, the constraint **ATOM[i]_CH[x]_CM0.CM0** >= 64 and 32<= **ATOM[i]_CH[x]_CM1.CM1** <= **ATOM[i]_CH[x]_CM0.CM0** − 32 must be fulfilled.
- For 0% duty cycle PWM signal (keep ! **ATOM[i]_CH[x]_CTRL_SOMP.SL**), the constraint **ATOM[i]_CH[x]_CM0.CM0** >= 64 and **ATOM[i]_CH[x]_CM1.CM1** = 0 must be fulfilled.
- For 100% duty cycle PWM signal (keep **ATOM[i]_CH[x]_CTRL_SOMP.SL**), the constraint **ATOM[i]_CH[x]_CM0.CM0** = **ATOM[i]_CH[x]_CM1.CM1** >= 64 must be fulfilled.

However, when **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1, there are other constraints and some of them are different from the case when HRES option is disabled:

- When configuring **ATOM[i]_CH[x]_CM0.CM0** [23:5]=0 or **ATOM[i]_CH[x]_CM1.CM1** [23:5]=0, the lower hres bits **ATOM[i]_CH[x]_CM0.CM0** [4:0]=0 or **ATOM[i]_CH[x]_CM1.CM1** [4:0] must also be 0.
- It is allowed to configure the registers as **ATOM[i]_CH[x]_CM1.CM1** <= **ATOM[i]_CH[x]_CM0.CM0** − 32 or **ATOM[i]_CH[x]_CM1.CM1** = **ATOM[i]_CH[x]_CM0.CM0** .
- The configuration **ATOM[i]_CH[x]_CM1.CM1** > **ATOM[i]_CH[x]_CM0.CM0** is not allowed.

Furthermore, if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1 and **ATOM[i]_CH[x]_CTRL_SOMP.EXT_TRIG** =0, the channel is triggered by the preceding channel with HRES output trigger chain. Besides the constraints above, there are more constraints between the triggered channel and the triggering channel. Here, the registers of the triggered channel are noted with a subscript "triggered" while the triggering of the preceding channel is marked with "master".

- The configuration must fulfill (**ATOM[i]_CH[x]_CM0.CM0**)$_{triggered}$ <= (**ATOM[i]_CH[x]_CM0.CM0**)$_{master}$ or (**ATOM[i]_CH[x]_CM0.CM0** [23:5])$_{triggered}$ > (**ATOM[i]_CH[x]_CM0.CM0** [23:5])$_{master}$.
- If (**ATOM[i]_CH[x]_CM0.CM0** [23:5])$_{triggered}$ = (**ATOM[i]_CH[x]_CM0.CM0** [23:5])$_{master}$, (**ATOM[i]_CH[x]_CM1.CM1**)$_{triggered}$ must be greater or equal to 32.

When HRES mode is enabled, *ATOM_CH_HRES_IN[x]* / *ATOM_CH_HRES_OUT[x]* are always generated synchronously together with *ATOM_CH_TRIGIN* [x:x]/ *ATOM_CH_TRIGOUT* [x:x] and may be delayed by same pipeline registers according to device configuration parameter configuration as defined in *15.2.3 "Internal Trigger Interface"* . If one channel triggers several following several channels, the 4 trigger signals *ATOM_CH_TRIGIN* [x:x]/ *ATOM_CH_TRIGOUT* [x:x] with *ATOM_CH_HRES_IN[x]* / *ATOM_CH_HRES_OUT[x]* build up a trigger chain from the triggering channel to the triggered channels. In this case, if there is no pipeline register in the trigger chain, these triggered channels can be configured with same operation registers and synchronously triggered to generate consistent PWM signals. However, if there are pipeline registers that may introduce delay of number of cluster clock periods, it is hard to simultaneously trigger several channels to generate consistent PWM signals. The reason for that is in HRES mode the operation clock frequency is the cluster clock and each pipeline register may cause a delay or shift of one cluster clock on the trigger signal pulse and this delay must be taken into account.

15.3.4 ATOM Signal Output Mode Serial (SOMS)

In ATOM Signal Output Mode Serial (SOMS) the ATOM channel acts as a serial output shift register where the content of the **ATOM[i]_CH[x]_CM1.CM1** register in the CCU1 unit is shifted out whenever the unit is triggered by the clock source, which is selected by **ATOM[i]_CH[x]_CTRL_SOMS.CLK_SRC** . The shift direction is configurable with the bit **ATOM[i]_CH[x]_CTRL_SOMS.ACB0** when ARU is disabled and the bit **ATOM[i]_CH[x]_STAT.ACBI** [0:0] when ARU is enabled.

The data inside the **ATOM[i]_CH[x]_CM1.CM1** register has to be aligned according to the selected shift direction in the **ATOM[i]_CH[x]_CTRL_SOMS.ACB0** / **ATOM[i]_CH[x]_STAT.ACBI** [0:0] bit. This means that when a right shift is selected, the data word has to be aligned to bit 0 of the **ATOM[i]_CH[x]_CM1.CM1** register and when a left shift is selected, the data has to be aligned to bit 23 of the **ATOM[i]_CH[x]_CM1.CM1** register.

Figure 110 SOMS Mode output generation

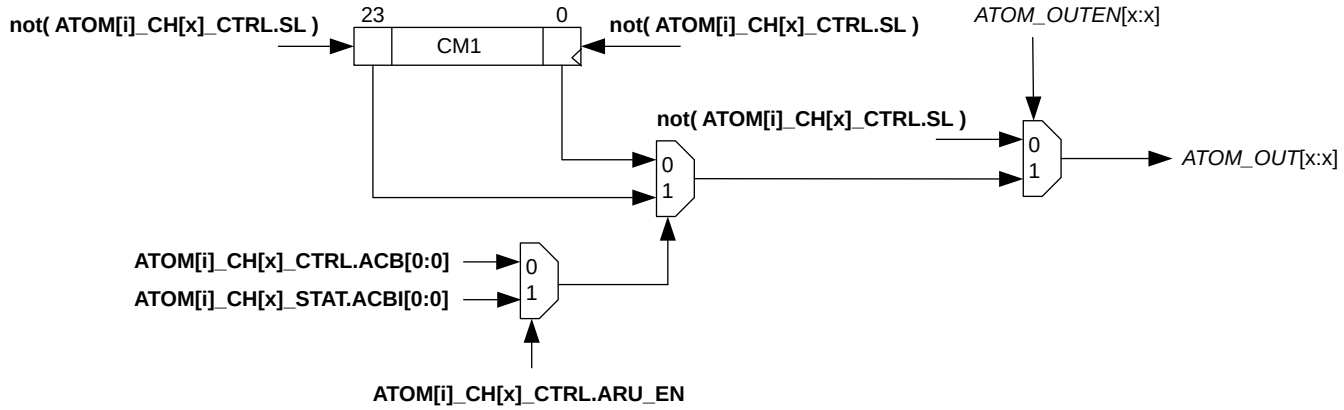


Figure *110 "SOMS Mode output generation"* shows the output generation in case of SOMS mode is selected.

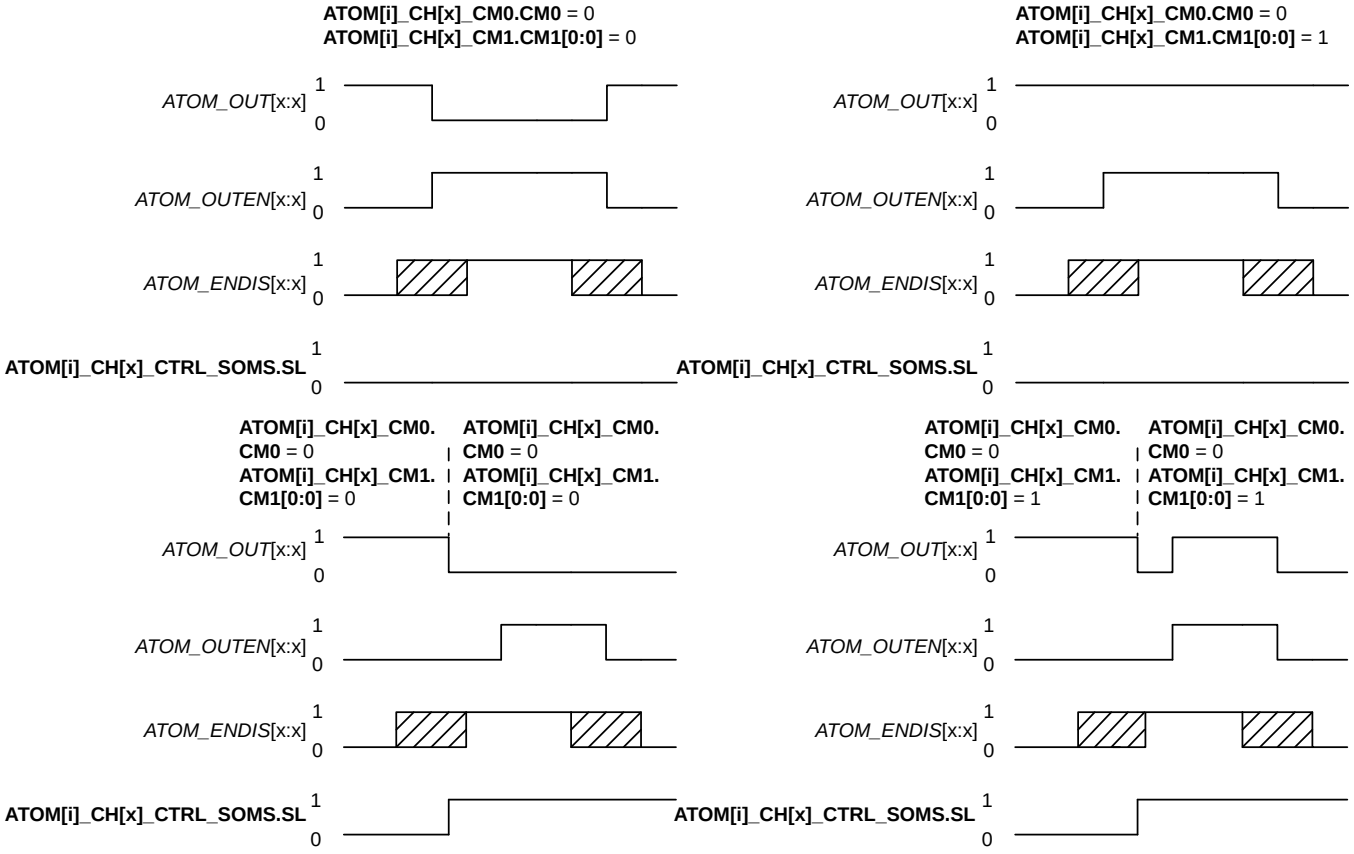
In SOMS mode CCU0 runs in counter/compare mode and counts the number of bits shifted out so far. The total number of bits that should be shifted is defined as **ATOM[i]_CH[x]_CM0.CM0** . The total number of bits that are visible at *ATOM_OUT* is **ATOM[i]_CH[x]_CM0.CM0** +1.

When the output is disabled the *ATOM_OUT* is set to the inverse **ATOM[i]_CH[x]_CTRL_SOMS.SL** bit definition.

When the content of the **ATOM[i]_CH[x]_CM1.CM1** register is shifted out, the inverse signal level is shifted into the **ATOM[i]_CH[x]_CM1.CM1** register.

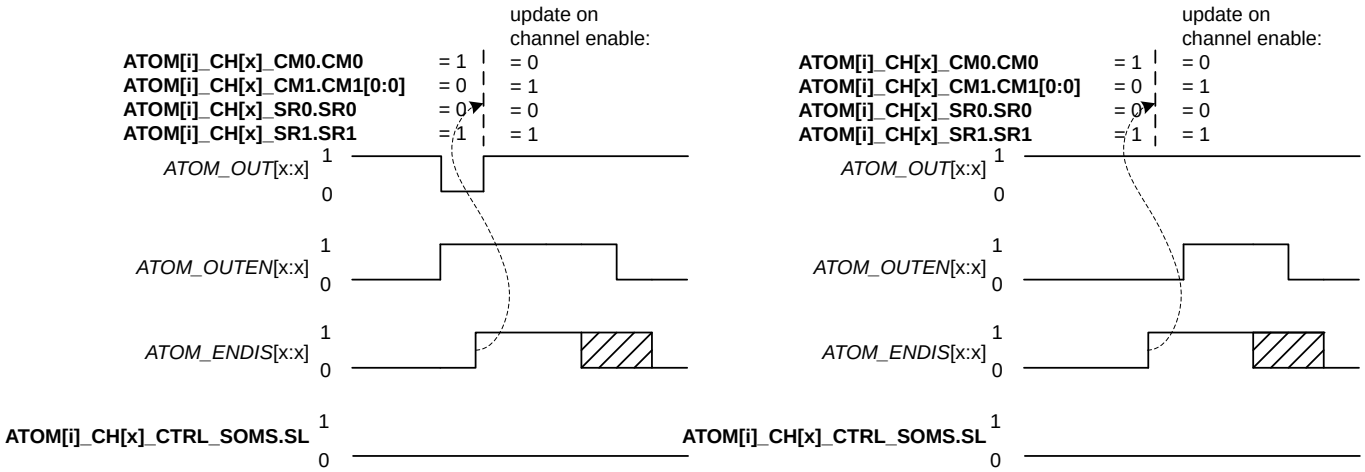
When the output is enabled while **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** is disabled, the ATOM_OUT signal level is defined by bit **ATOM[i]_CH[x]_CM1.CM1** [23:0], dependent on the shift direction defined by **ATOM[i]_CH[x]_CTRL_SOMS.ACB0** or **ATOM[i]_CH[x]_STAT.ACBI** [0:0] register setting. Figure *111 "SOMS Output signal level at startup, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] disabled"* should clarify the ATOM channel startup behavior in this case for right shift. For left shift the **ATOM[i]_CH[x]_CM1.CM1** [0:0] in *111 "SOMS Output signal level at startup, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] disabled"* has to be replaced by **ATOM[i]_CH[x]_CM1.CM1** [23:23].

Figure 111 SOMS Output signal level at startup, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] disabled



If **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** is set and the channel is enabled, the output level is defined by bit **ATOM[i]_CH[x]_CM1.CM1** [0:0] or **ATOM[i]_CH[x]_CM1.CM1** [23:23] depending on the shift direction. Figure *112 "SOMS Output signal level at startup, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] enabled"* shows the output behavior in that case.

Figure 112 SOMS Output signal level at startup, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] enabled



When the serial data to be shifted is provided via ARU the number of bits that should be shifted has to be defined in the lower 24-bits of the ARU word ([23:0]) and the data that is to be shifted has to be defined in the ARU bits [47:24] aligned according to the shift direction. This shift direction has to be defined in the ARU word bit 48 (**ATOM[i]_CH[x]_CTRL_SOMS.ACB0** bit).

After the channel was enabled by setting of **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[x]** =1 and **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =1 and ARU input stream is enabled by setting of **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** =1, the first ARU read request is set up with the next active clock source which is selected by **ATOM[i]_CH[x]_CTRL_SOMS.CLK_SRC** .

If bit **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** of a channel x is set, after update of **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the content of the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register, a new ARU read request is set up.

If bit **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** of a channel x is not set, no (further) ARU read request is set up (because the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register are never used for update) and the ATOM may stop shifting after **ATOM[i]_CH[x]_CN0.CN0** has reached **ATOM[i]_CH[x]_CM0.CM0** .

In this case also no automatic restart of shifting is possible.

If a channel is enabled with the settings SOMS mode and **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** = 1, the first received values from ARU are stored in register **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** . If **ATOM[i]_CH[x]_CN0.CN0** and **ATOM[i]_CH[x]_CM0.CM0** are 0 (i.e. **ATOM[i]_CH[x]_CN0.CN0** is not counting) and the update of channel x is enabled (**ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =1), an immediate update of the register **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** is also done. This update of **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** triggers the start of shifting.

It is recommended to configure the ATOM channel in one-shot mode when the **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** bit is not set, since the ATOM channel would reload new values from the shadow registers when **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** .

15.3.4.1 SOMS mode with ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN = 1 and ATOM[i]_CH[x]_CTRL_SOMS.OSM = 0, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] = 1:

In case of bit **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** is set and bit **ATOM[i]_CH[x]_CTRL_SOMS.OSM** is not set, the channel is running in the SOMS continuous mode. Then, if the content of the **ATOM[i]_CH[x]_CM0.CM0** register equals the counter **ATOM[i]_CH[x]_CN0.CN0** , the **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** registers are reloaded with the **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** content and new values are requested from the ARU. If the update of the shadow registers does not happen before **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** the old values of **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are used to reload the operation registers.

In contrast to controlling the channel via AEI, the shift direction defined by ARU word bit 48 takes effect only after the update of the operation registers **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** from the shadow registers **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** .

15.3.4.2 SOMS mode with ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN = 1 and ATOM[i]_CH[x]_CTRL_SOMS.OSM = 1, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] = 1:

In case bit **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** is set and bit **ATOM[i]_CH[x]_CTRL_SOMS.OSM** is set, the channel runs in the SOMS one-shot mode. Then, if the content of the **ATOM[i]_CH[x]_CM0.CM0** register equals the counter **ATOM[i]_CH[x]_CN0.CN0** and if new values are available in **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** (bit **ATOM[i]_CH[x]_STAT.DV** set), the **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** registers are reloaded with the **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** content and new values are requested from the ARU. If no new values are available in **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** , the register **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** will not be updated. The counter **ATOM[i]_CH[x]_CN0.CN0** stops and the ATOM channel continues to request new data from ARU. A subsequent reception of new ARU data in **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** will immediately force the update of the register **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** and restart the counter **ATOM[i]_CH[x]_CN0.CN0** .

15.3.4.3 SOMS mode with ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN = 0 and ATOM[i]_CH[x]_CTRL_SOMS.OSM = 0, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] = 1:

In case bit **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** is not set and bit **ATOM[i]_CH[x]_CTRL_SOMS.OSM** is not set, the ATOM channel updates its **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the content of the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register and restarts shifting immediately. The first bit of new **ATOM[i]_CH[x]_CM1.CM1** register value will be applied at the output without any gap to the last bit of the previous **ATOM[i]_CH[x]_CM1.CM1** register value.

15.3.4.4 SOMS mode with ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN = 0 and ATOM[i]_CH[x]_CTRL_SOMS.OSM = 1, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] = 1:

In case bit **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** is not set and bit **ATOM[i]_CH[x]_CTRL_SOMS.OSM** is set, the ATOM channel stops shifting when **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** and no update of **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** is performed.

Then, the shifting of the channel can be restarted again by writing a zero to the **ATOM[i]_CH[x]_CN0.CN0** register again.

The writing of a zero to **ATOM[i]_CH[x]_CN0.CN0** also causes an immediate update of **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the content of **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register.

Restarting the shifting by writing a zero to **ATOM[i]_CH[x]_CN0.CN0** can also be done while a shift cycle is already active. Then an immediate update of **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the content of **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register is done followed by a new shift operation.

15.3.4.5 SOMS mode with ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN = 0 and ATOM[i]_CH[x]_CTRL_SOMS.OSM = 0, ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x] = 0:

In case of bit **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** is not set and bit **ATOM[i]_CH[x]_CTRL_SOMS.OSM** is not set and the update enable bit **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** is not set, no update of **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register is executed. The shifting is done continuously without reloading from shadow register **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** and without any gap between the shift cycles.

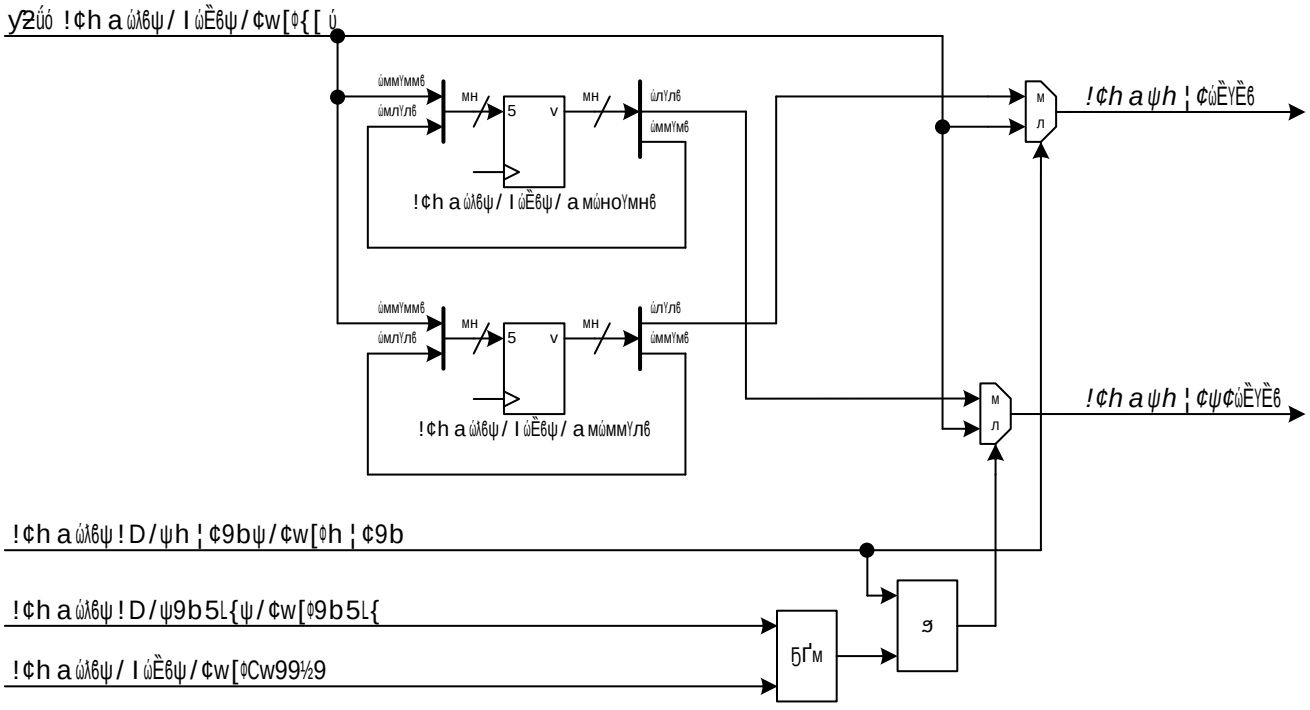
15.3.4.6 SOMS mode with double output

If the mode bit DSO is additionally set in SOMS mode (in register **ATOM[i]_CH[x]_CTRL_SOMS**), two 12-bit data streams can be shifted out on the outputs *ATOM_OUT* and *ATOM_OUT_T* of instance i and channel x in parallel.
This is achieved by splitting the register **ATOM[i]_CH[x]_CM1.CM1** into two parts. The lower 12 bits are used as a shift register of output *ATOM_OUT* (i.e. bit 0 is assigned to output *ATOM_OUT*), the upper 12 bits are used as a shift register of output *ATOM_OUT_T* (i.e. bit 12 is assigned to output *ATOM_OUT_T*).
On bit **ATOM[i]_CH[x]_CM1.CM1** [23:11] the value ! **ATOM[i]_CH[x]_CTRL_SOMS.SL** is shifted in.

In this mode only right shift is possible. Bit **ATOM[i]_CH[x]_CTRL_SOMS.ACB0** is ignored.

This behavior is depicted in the following figure:

Figure 113 Double Output Shift Mode



15.3.4.7 Interrupts in SOMS mode

In ATOM Signal Output Mode Serial only the interrupt **ATOM[i]_CH[x]_IRQ_NOTIFY.CCU[0]TC** in case of **ATOM[i]_CH[x]_CN0.CN0** >= **ATOM[i]_CH[x]_CM0.CM0** is generated. The interrupt **ATOM[i]_CH[x]_IRQ_NOTIFY.CCU[1]TC** has no meaning and is not generated.

15.3.5 ATOM Signal Output Mode Buffered Compare(SOMB)

15.3.5.1 Overview

In ATOM Signal Output Mode Buffered Compare (SOMB) the output action is performed according to the comparison result of the input values located in **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** registers and the two (three) time base values *CCM[i]_TBU_TS0* or *CCM[i]_TBU_TS1* (or *CCM[i]_TBU_TS2*) provided by the TBU. For a description of the time base generation please refer to the

TBU specification in chapter *12 "Time Base Unit (TBU)"* . It is configurable, which of the two (three) time bases is to be compared with one or both values in **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** .

The compare strategy of the two compare units CCU0 and CCU1 is controlled by the value of bit field of **ATOM[i]_CH[x]_STAT.ACBI** . This bit field is only readable over configuration interface. If ARU is disabled, the bit field **ATOM[i]_CH[x]_STAT.ACBI** can only be updated with the value of bit field **ATOM[i]_CH[x]_CTRL_SOMB.ACB** . If ARU is enabled, the **ATOM[i]_CH[x]_STAT.ACBI** bit field can be updated with the value of shadow register **ACB_SR** which contains a value received via ARU.

The table below lists all valid control configurations for bit field **ATOM[i]_CH[x]_STAT.ACBI** .

Table 37 ATOM SOMB compare strategies

ATOM[i]_CH[x]_STAT.ACBI [4:4]	ATOM[i]_CH[x]_STAT.ACBI [3:3]	ATOM[i]_CH[x]_STAT.ACBI [2:2]	CCU0/CCU1 comparison control
0	0	0	Prohibited.
0	0	1	Prohibited.
0	1	0	Compare in CCU0 only, use time base *CCM[i]_TBU_TS0* . Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMB.SL** and **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** / **ATOM[i]_CH[x]_STAT.ACBI** [1:0] bits (details see table *38 "ATOM SOMB output control by ATOM[i]_CH[x]_STAT.ACBI[1:0] and ATOM[i]_CH[x]_CTRL_SOMB.SL"*).
0	1	1	Compare in CCU1 only, use time base *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level at the compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMB.SL** and **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** / **ATOM[i]_CH[x]_STAT.ACBI** [1:0] bits (details see table *38 "ATOM SOMB output control by ATOM[i]_CH[x]_STAT.ACBI[1:0] and ATOM[i]_CH[x]_CTRL_SOMB.SL"*).
1	0	0	Serve Last: Compare in CCU0 and then in CCU1 using *CCM[i]_TBU_TS0* . Output signal level at the CCU0 compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMB.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0] (details see table *38 "ATOM SOMB output control by ATOM[i]_CH[x]_STAT.ACBI[1:0] and ATOM[i]_CH[x]_CTRL_SOMB.SL"*). At the CCU1 compare match event the output level is toggled.
1	0	1	Serve Last: Compare in CCU0 and then in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level at the CCU0 compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMB.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0] (details see table *38 "ATOM SOMB output control by ATOM[i]_CH[x]_STAT.ACBI[1:0] and ATOM[i]_CH[x]_CTRL_SOMB.SL"*). At the CCU1 compare match event the output level is toggled.

ATOM[i]_CH[x]_STAT.ACBI [4:4]	ATOM[i]_CH[x]_STAT.ACBI [3:3]	ATOM[i]_CH[x]_STAT.ACBI [2:2]	CCU0/CCU1 comparison control
1	1	0	Serve Last: Compare in CCU0 using *CCM[i]_TBU_TS0* and then in CCU1 using *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . Output signal level at the CCU1 compare match event is defined by combination of **ATOM[i]_CH[x]_CTRL_SOMB.SL** and **ATOM[i]_CH[x]_STAT.ACBI** [1:0] (details see table *38 "ATOM SOMB output control by ATOM[i]_CH[x]_STAT.ACBI[1:0] and ATOM[i]_CH[x]_CTRL_SOMB.SL"*).
1	1	1	Cancels pending comparison independent on **ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN**

Based on the configured compare strategy **ATOM[i]_CH[x]_STAT.ACBI** [4:2], CCU0 unit always performs cyclic event compare between **ATOM[i]_CH[x]_CM0.CM0** and the selected TBU time base *CCM[i]_TBU_TS0* / *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* , while CCU1 unit always performs cyclic event compare between **ATOM[i]_CH[x]_CM1.CM1** and the selected TBU time base *CCM[i]_TBU_TS0* / *CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2* . Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more information about cyclic event compare strategy. When the selected time base is "cyclically greater than or equal to" the compare value of **ATOM[i]_CH[x]_CM0.CM0** , CCU0 compare match event is considered to have occurred. When the selected time base is "cyclically greater than or equal to" the compare value of **ATOM[i]_CH[x]_CM1.CM1** , CCU1 compare match event is considered to have occurred. Whenever CCU0 or CCU1 compare match event has occurred, an edge is generated on output *ATOM_OUT* [x:x], depending on the predefined signal level in **ATOM[i]_CH[x]_CTRL_SOMB.SL** bit in combination with two control bits **ATOM[i]_CH[x]_STAT.ACBI** [1:0].

In SOMB mode, if ARU access is enabled, the new compare values received via ARU are always stored in the shadow register **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** and the control bits received via ARU are stored in the internal register **ACB_SR** .

If the scheduled compare match event in CCU0 and/or CCU1 have occurred and the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register contain new valid values, the registers **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** will be updated automatically with the content of the corresponding registers **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** , the **ATOM[i]_CH[x]_STAT.ACBI** bit field will be updated with the content of internal **ACB_SR** register and the **ATOM[i]_CH[x]_STAT.DV** bit will be set. If the registers **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** and **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** contain no valid value, the compare units will wait in the idle state.

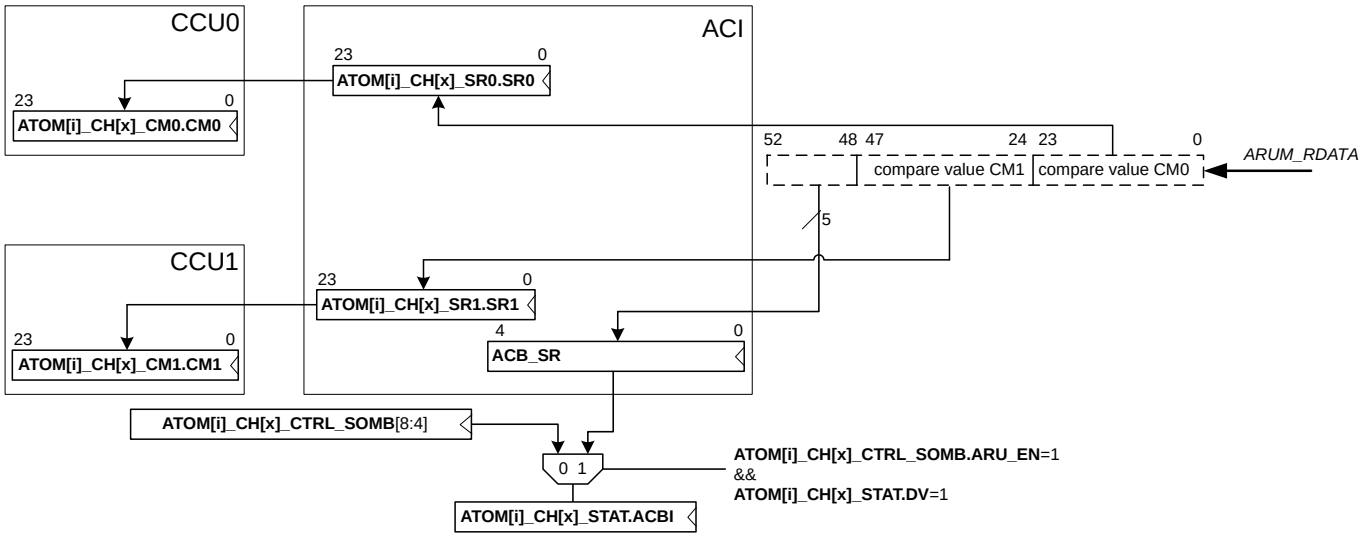
On a compare match event in one of the compare units CCU0/CCU1 units the output *ATOM_OUT* [x:x] is set according to combination of the bits **ATOM[i]_CH[x]_STAT.ACBI** [1:0] and **ATOM[i]_CH[x]_CTRL_SOMB.SL** according to the following table.

Table 38 ATOM SOMB output control by ATOM[i]_CH[x]_STAT.ACBI[1:0] and ATOM[i]_CH[x]_CTRL_SOMB.SL

SL	ATOM[i]_CH[x]_STAT.ACBI [1:1]	ATOM[i]_CH[x]_STAT.ACBI [0:0]	Output Behavior
0	0	0	No signal level change at output.
0	0	1	Set output signal level to 1.
0	1	0	Set output signal level to 0.
0	1	1	Toggle output signal level.
1	0	0	No signal level change at output.
1	0	1	Set output signal level to 0.
1	1	0	Set output signal level to 1.
1	1	1	Toggle output signal level.

In contrast to the SOMC mode, no time stamp value of TBU is captured in **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register.

Figure 114 ARU interface behavior in SOMB mode



The flag **ATOM[i]_CH[x]_STAT.DV** of register **ATOM[i]_CH[x]_STAT** indicates that at least one of the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register contains valid data and a compare event may be pending (if the channel is enabled).
The **ATOM[i]_CH[x]_STAT.DV** flag is reset if none of the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** registers contain valid data.

15.3.5.2 SOMB controlled over configuration interface

If bit **ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN** is not set, the ATOM channel can only be controlled over the configuration interface.

Writing to one of the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** registers automatically sets **ATOM[i]_CH[x]_STAT.DV** = 1 to validate the new compare value. A comparison depending on value **ATOM[i]_CH[x]_STAT.ACBI** is started immediately.

Note:
It is strongly recommended to update the bit fields **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** and **ATOM[i]_CH[x]_CTRL_SOMB.ACB42** before the registers **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** , as only the bits **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** and **ATOM[i]_CH[x]_CTRL_SOMB.ACB42** can be written and these bit fields serve as shadow registers for the working register **ATOM[i]_CH[x]_STAT.ACBI** .

Note:
It is mandatory to always write both bit fields **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** regardless of the compare strategy.

The compare strategy is controlled by the value stored in bit field **ATOM[i]_CH[x]_STAT.ACBI** . If ARU is disabled, this bit field can only be updated with the value of bit fields **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** and **ATOM[i]_CH[x]_CTRL_SOMB.ACB42** .

The update of bit field **ATOM[i]_CH[x]_STAT.ACBI** can be triggered by a forced update or the normal update mechanism with **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =0b10.

If ARU access is disabled (**ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN** =0), a force update updates the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the content of **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register and the **ATOM[i]_CH[x]_STAT.ACBI** bit field with the content of bit field **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** and **ATOM[i]_CH[x]_CTRL_SOMB.ACB42** .

Writing to one of the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** registers and triggering a forced update event can update **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the value of **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register and update **ATOM[i]_CH[x]_STAT.ACBI** bit field with the content of bits **ATOM[i]_CH[x]_CTRL_SOMB.ACB10** and **ATOM[i]_CH[x]_CTRL_SOMB.ACB42** . Then a new comparison will be started.

When a previous comparison is finished (**ATOM[i]_CH[x]_STAT.DV** bit is reset), writing to one of the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** registers will update **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_STAT.ACBI** while **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL** [x]= 0b10 and starts comparison with these updated new values.

15.3.5.3 SOMB under ARU control

If both compare units CCU0/CCU1 are finished with previous job (depending on compare strategy) and **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** contain no new valid value, then they will wait until new data is received via ARU and stored in **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register.

If both compare units are finished with previous job (depending on compare strategy) and the new data is available in the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register, the update of the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the value of the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register and the **ATOM[i]_CH[x]_STAT.ACBI** bit field with the value of internal **ACB_SR** register takes place and a new compare job is started immediately.

After an update of the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register, a new ARU read request is set.

New compare values received via the ARU are stored in shadow register **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** . The control bits received via ARU are stored in the internal register **ACB_SR** .

If ARU access is enabled (**ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN** =1), a force update event updates the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the content of **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register and the **ATOM[i]_CH[x]_STAT.ACBI** bit field with the content of internal **ACB_SR** register.

15.3.5.3.1 ARU Non-blocking mode

If bit **ATOM[i]_CH[x]_CTRL_SOMB.ABM** is not set, the ARU blocking mode is disabled. In this case, the ATOM channel continuously reads via ARU and stores new values in the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register and the ACB shadow register **ACB_SR** .

If **ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN** is not set, the bit **ATOM[i]_CH[x]_CTRL_SOMB.ABM** has no meaning.

In ARU non-blocking mode, if **ATOM[i]_CH[x]_CTRL_SOMB.FREEZE** =0 and the channel is disabled (*ATOM_ENDIS* =0), the comparison is stopped, the current compare values stored in **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** and also the next compare values stored in **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are no more valid and the ARU read request will be reset regardless of the current compare state. Therefore, no more ARU data can be further transferred to the ATOM channel. Afterwards, if the channel is enabled (*ATOM_ENDIS* =1) again, the comparison that was stopped before by disabling the channel cannot be continued. The channel must raise a new ARU read request again to receive new data in the shadow registers that will be immediately loaded into the operation registers. Since then, new compare can start.

In ARU non-blocking mode, if **ATOM[i]_CH[x]_CTRL_SOMB.FREEZE** =1 and the channel is disabled (*ATOM_ENDIS* =0), the comparison is stopped, but the current compare values stored in **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** and also the next compare values stored in **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are still valid but the ARU read request will be reset regardless of the current compare state. Therefore, no more ARU data can be further transferred to the ATOM channel. Afterwards, when the channel is enabled again, it can continue comparison with the old compare values (i.e. before the channel was disabled) and simultaneously request ARU data.

15.3.5.3.2 ARU Blocking mode

If bit **ATOM[i]_CH[x]_CTRL_SOMB.ABM** is set, the ARU blocking mode is enabled. In this case the ATOM channel stops requesting new **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** values via ARU after reception of a new **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** value and restarts requesting a new value via ARU after compare match on both compare units (depending on compare strategy) followed by the immediate update of the **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with content of **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register and an update of **ATOM[i]_CH[x]_STAT.ACBI** with the content of **ACB_SR** .

If **ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN** is not set, the bit **ATOM[i]_CH[x]_CTRL_SOMB.ABM** has no meaning.

In ARU blocking mode, if **ATOM[i]_CH[x]_CTRL_SOMB.FREEZE** =0 and the channel is disabled (*ATOM_ENDIS* =0), the comparison is stopped, the current compare values stored in **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** and also the next compare values stored in **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are no more valid and the ARU read request will be reset regardless of the current compare state. Therefore, no more ARU data can be further transferred to the ATOM channel during the channel is disabled. Afterwards, if the channel is enabled (*ATOM_ENDIS* =1) again, the comparison that was stopped before by disabling the channel cannot be continued. The channel must raise a new ARU read request again to receive new data in the shadow registers that will be immediately loaded into the operation registers. Since then, new compare can start.

In ARU blocking mode, if **ATOM[i]_CH[x]_CTRL_SOMB.FREEZE** =1 and the channel is disabled (*ATOM_ENDIS* =0), the comparison is stopped, but the current compare values stored in **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** and also the next compare values stored in **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are still valid but the ARU read request will be reset regardless of the current compare state. Therefore, no more ARU data can be further transferred to the ATOM channel during the channel is disabled. Afterwards, if the channel is enabled again, it can continue comparison with the old compare values stored in **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** (i.e. before the channel was disabled). Then, if the specified compare match event occurs, **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** will be updated with the old values in the shadow registers if they are not consumed before disabling channel and request new values for shadow registers.

15.3.5.3.3 Late Update via configuration interface

Although, the ATOM channel may be controlled by data received via the ARU, it is possible to request a late update of the compare register at any time through the configuration interface. This can be initiated by setting the **ATOM[i]_CH[x]_CTRL_SOMB.WR_REQ** bit.

If none of the two compare match events happened, the ATOM channel accepts the setting of **ATOM[i]_CH[x]_CTRL_SOMB.WR_REQ** bit. In this case, the ATOM will request no further data from ARU (if ARU access was enabled) and will disable the update of **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** register with the content of **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** register on a compare match event.

If in 'serve last' strategy, one compare match event occurs, the **ATOM[i]_CH[x]_CTRL_SOMB.WR_REQ** bit cannot be set to 1 and **ATOM[i]_CH[x]_STAT.WRF** flag will be set to 1 to indicate that.

The channel will in any case continue to compare against the values stored inside the compare registers (if bit **ATOM[i]_CH[x]_STAT.DV** was set). The compare values can now be updated by writing via the configuration interface to the shadow registers and force the ATOM channel to update the compare registers by writing to the force update register bits in the AGC register.

With a force update the **ATOM[i]_CH[x]_CTRL_SOMB.WR_REQ** bit is reset automatically and the ARU read request is set up again (if ARU access was enabled).

15.4 ATOM Software reset of Channels

Each channel x is assigned a unique bit x in the configuration register **ATOM[i]_AGC_GLB_CTRL** for software reset of the ATOM[i] channel x.

Writing the value 0b1 to the bitfield **ATOM[i]_AGC_GLB_CTRL.RST_CH[x]** via the configuration interface, will immediately reset the content of the following channel x registers to the initial hardware reset state.

- **ATOM[i]_CH[x]_CTRL**
- **ATOM[i]_CH[x]_CTRL_SR**
- **ATOM[i]_CH[x]_CTRL2**
- **ATOM[i]_CH[x]_CN0.CN0**
- **ATOM[i]_CH[x]_CM0.CM0**
- **ATOM[i]_CH[x]_CM1.CM1**
- **ATOM[i]_CH[x]_SR0.SR0**
- **ATOM[i]_CH[x]_SR1.SR1**
- **ATOM[i]_CH[x]_STAT**
- **ATOM[i]_CH[x]_RDADDR**
- **ATOM[i]_CH[x]_IRQ_NOTIFY**
- **ATOM[i]_CH[x]_IRQ_EN**
- **ATOM[i]_CH[x]_IRQ_MODE**
- internal signals inside the ATOM[i] channel [x]:

Atomic reset of multiple channels is possible by writing a 1 on each associated bit. For example, writing 0x0F to register **ATOM[0]_AGC_GLB_CTRL** will reset the ATOM0 channel x (x∈{0, 1, 2, 3}).

15.5 ATOM Registers Description

15.5.1 ATOM[i]_AGC_GLB_CTRL

Description	ATOM[i] AGC global control register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ATOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_AGC_GLB_CTRL
Address	$0x20000 * i + 0x1C40$
C-Name	GTM.CLS[i].ATOM.AGC.GLB_CTRL

Interface: MCS[i]

Name	ATOM[i]_AGC_GLB_CTRL
Address	$0x1C40$
C-Name	

HOST_TRIG	
Description	Trigger request signal (see AGC) to update the register ATOM[i]_AGC_ENDIS_STAT and ATOM[i]_AGC_OUTEN_STAT

HOST_TRIG	
Loop	–
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0 : no trigger request 1 : set trigger request
	Note: This flag is reset automatically after triggering the update

RST_CH[k]	
Description	Software reset of channel [k]
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[k+8 : k+8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0 : No action 1 : Reset channel
R-Coding	0 : No status
	Note: This bit is cleared automatically after write access via the configuration interface. The channel registers are set to their reset values and channel operation is stopped immediately. The output register of SOU unit is reset to inverse reset value of **ATOM[i]_CH[k]_CTRL.SL** bit. **Note:** The write access to **ATOM[i]_AGC_GLB_CTRL.RST_CH[c]** bit will take two internal bus clock cycles. The internal bus clock is the cluster clock.

UPEN_CTRL[k]	
Description	ATOM channel [k] enable update of register ATOM[i]_CH[k]_CM0, ATOM[i]_CH[k]_CM1, ATOM[i]_CH[k]_CTRL.SL and ATOM[i]_CH[k]_CTRL.CLK_SRC from ATOM[i]_CH[k]_SR0, ATOM[i]_CH[k]_SR1, ATOM[i]_CH[k]_CTRL_SR.SL_SR and ATOM[i]_CH[k]_CTRL_SR.CLK_SRC_SR.
Loop	$k = \{n : 0 <= n <= 7\}$

UPEN_CTRL[k]	
Bit Range	$[2 * k + 17 : 2 * k + 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : don't care, bits will not be changed 0b01 : disable update 0b10 : enable update 0b11 : don't care, bits will not be changed
R-Coding	0b00 : update disabled 0b01 : unused 0b10 : unused 0b11 : update enabled

15.5.2 ATOM[i]_AGC_ENDIS_CTRL

Description	ATOM[i] AGC enable/disable control register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_AGC_ENDIS_CTRL
Address	$0x20000 * i + 0x1C44$
C-Name	GTM.CLS[i].ATOM.AGC.ENDIS_CTRL

Interface: MCS[i]

Name	ATOM[i]_AGC_ENDIS_CTRL
Address	$0x1C44$
C-Name	

ENDIS_CTRL[k]	
Description	ATOM [i] channel [x] enable/disable control register.
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * k + 1 : 2 * k]$
Access Type	RW
Volatile	False

ENDIS_CTRL[k]	
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : bit field **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[k]** of register **ATOM[i]** will not be changed on an update trigger 0b01 : disable channel on an update trigger 0b10 : enable channel on an update trigger 0b11 : don't change bits of this register
R-Coding	0b00 : bit field **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[k]** will not be changed on an update trigger 0b01 : disable channel on an update trigger 0b10 : enable channel on an update trigger

15.5.3 ATOM[i]_AGC_ENDIS_STAT

Description	ATOM[i] AGC enable/disable status register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_AGC_ENDIS_STAT
Address	$0x20000 * i + 0x1C48$
C-Name	GTM.CLS[i].ATOM.AGC.ENDIS_STAT

Interface: MCS[i]

Name	ATOM[i]_AGC_ENDIS_STAT
Address	$0x1C48$
C-Name	

ENDIS_STAT[k]	
Description	ATOM [i] channel [k] enable/disable status register
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * k + 1 : 2 * k]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0

ENDIS_STAT[k]	
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : don't care, bits **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[k]** will not be changed 0b01 : disable channel 0b10 : enable channel 0b11 : don't care, bits **ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[k]** will not be changed
R-Coding	0b00 : channel disabled 0b01 : unused 0b10 : unused 0b11 : channel enabled
	Note: If a channel of ATOM instance i is disabled(*ATOM_ENDIS* [x:x]=0), the counter **ATOM[i]_CH[x]_CN0.CN0** is stopped (SOMP, SOMS mode) and the comparisons in CCU0 and CCU1 are stopped (SOMP, SOMC, SOMB mode). When the channel is re-enabled afterwards, the counter **ATOM[i]_CH[x]_CN0.CN0** starts counting from its current value or/and the comparison is restarted. In SOMI mode, the operation and output generation is independent on **ATOM[i]_AGC_ENDIS_STAT** / **ATOM[i]_AGC_ENDIS_CTRL** and *ATOM_ENDIS* [x:x].

15.5.4 ATOM[i]_AGC_ACT_TB

Description	ATOM[i] AGC action time base register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_AGC_ACT_TB
Address	$0x20000 * i + 0x1C4C$
C-Name	GTM.CLS[i].ATOM.AGC.ACT_TB

Interface: MCS[i]

Name	ATOM[i]_AGC_ACT_TB
Address	$0x1C4C$
C-Name	

ACT_TB	
Description	Action time base. When the selected TBU time base CCM[i]_TBU_TS0/CCM[i]_TBU_TS1/CCM[i]_TBU_TS2 is "-cyclically greater than or equal to" the action time base ATOM[i]_AGC_ACT_TB.ACT_TB, the compare event specified in ATOM[i]_AGC_ACT_TB.ACT_TB is considered to have occurred in the past and so the trigger ATOM_CTRL_TRIG is immediately generated in AGC unit if ATOM[i]_AGC_ACT_TB.TB_TRIG is 1. Please refer to chapter GTM Architecture for more information about cyclic event compare strategy.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

ACT_TB	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

TB_TRIG	
Description	Set trigger request
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : no trigger request 1 : set trigger request
	Note: This flag is reset automatically if the selected time base unit (*CCM[i]_TBU_TS0* or *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* if present) has reached the value **ATOM[i]_AGC_ACT_TB.ACT_TB** and the update of the register were triggered.

TBU_SEL	
Description	Selection of time base used for comparison
Loop	–
Bit Range	[26 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : *CCM[i]_TBU_TS0* selected 0b01 : *CCM[i]_TBU_TS1* selected 0b10 : *CCM[i]_TBU_TS2* selected 0b11 : same as 0b00

15.5.5 ATOM[i]_AGC_OUTEN_CTRL

Description	ATOM[i] AGC output enable control register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_AGC_OUTEN_CTRL
Address	$0x20000 * i + 0x1C50$
C-Name	GTM.CLS[i].ATOM.AGC.OUTEN_CTRL

Interface: MCS[i]

Name	ATOM[i]_AGC_OUTEN_CTRL
Address	$0x1C50$
C-Name	

OUTEN_CTRL[k]	
Description	Output enable control of ATOM [i] channel [k] output ATOM_OUT[x:x]
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * k + 1 : 2 * k]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : bit field **ATOM[i]_AGC_OUTEN_STAT.OUTEN_STAT[k]** will not be changed on an update trigger 0b01 : disable channel output on an update trigger 0b10 : enable channel output on an update trigger 0b11 : don't change bits of this register
R-Coding	0b00 : bit field **ATOM[i]_AGC_OUTEN_STAT.OUTEN_STAT[k]** will not be changed on an update trigger 0b01 : disable channel output on an update trigger 0b10 : enable channel output on an update trigger

15.5.6 ATOM[i]_AGC_OUTEN_STAT

Description	ATOM[i] AGC output enable status register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	ATOM[i]_CLK_ENABLE == 1

Interface: CPU

Name	ATOM[i]_AGC_OUTEN_STAT
Address	$0x20000 * i + 0x1C54$
C-Name	GTM.CLS[i].ATOM.AGC.OUTEN_STAT

Interface: MCS[i]

Name	ATOM[i]_AGC_OUTEN_STAT
Address	$0x1C54$
C-Name	

OUTEN_STAT[k]	
Description	Output enable status of ATOM [i] channel [x] output ATOM_OUT[k:k]
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * k + 1 : 2 * k]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : don't care, bits **ATOM[i]_AGC_OUTEN_STAT.OUTEN[k]** will not be changed 0b01 : disable output 0b10 : enable output 0b11 : don't care, bits **ATOM[i]_AGC_OUTEN_STAT.OUTEN[k]** will not be changed
R-Coding	0b00 : output disabled 0b01 : unused 0b10 : unused 0b11 : output enabled
	Note: If the output of ATOM i channel x is disabled (*ATOM_OUTEN* [x:x]=0), both ATOM channel outputs *ATOM_OUT* [x:x] and *ATOM_OUT_T* [x:x] of instance i channel x are the inverse value of bit **ATOM[i]_CH[x]_CTRL.SL** .

15.5.7 ATOM[i]_AGC_FUPD_CTRL

Description	ATOM[i] AGC force update control register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	ATOM[i]_CLK_ENABLE == 1

Interface: CPU

Name	ATOM[i]_AGC_FUPD_CTRL
Address	$0x20000 * i + 0x1C58$
C-Name	GTM.CLS[i].ATOM.AGC.FUPD_CTRL

Interface: MCS[i]

Name	ATOM[i]_AGC_FUPD_CTRL
Address	$0x1C58$
C-Name	

FUPD_CTRL[k]	
Description	Force update of ATOM channel [k] operation registers
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * k + 1 : 2 * k]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : don't care, bits **ATOM[i]_AGC_FUPD_CTRL.FUPD_CTRL[k]** will not be changed 0b01 : disable force update 0b10 : enable force update 0b11 : don't care, bits **ATOM[i]_AGC_FUPD_CTRL.FUPD_CTRL[k]** will not be changed
R-Coding	0b00 : force update disabled 0b01 : unused 0b10 : unused 0b11 : force update enabled
	If enabled, force update of register **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL.SL** and **ATOM[i]_CH[x]_CTRL.CLK_SRC** triggered by **ATOM[i]_AGC_GLB_CTRL.HOST_TRIG** , **ATOM[i]_AGC_ACT_TB.ACT_TB** compare match or internal trigger.

RSTCN0_CH[k]	
Description	Reset ATOM[i]_CH[k]_CN0 of channel [k] on force update event
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * k + 17 : 2 * k + 16]$
Access Type	RW
Volatile	False
Multithread	False

RSTCN0_CH[k]	
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : don't care, bits will not be changed 0b01 : do not reset **ATOM[i]_CH[x]_CN0.CN0** on forced update 0b10 : reset **ATOM[i]_CH[x]_CN0.CN0** on forced update 0b11 : don't care, bits will not be changed
R-Coding	0b00 : **ATOM[i]_CH[x]_CN0.CN0** is not reset on forced update 0b01 : unused 0b10 : unused 0b11 : **ATOM[i]_CH[x]_CN0.CN0** is reset on forced update
	If enabled, **ATOM[i]_CH[x]_CN0.CN0** will be reset with a force update event.

15.5.8 ATOM[i]_AGC_INT_TRIG

Description	ATOM[i] AGC internal trigger control register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ATOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_AGC_INT_TRIG
Address	$0x20000 * i + 0x1C5C$
C-Name	GTM.CLS[i].ATOM.AGC.INT_TRIG

Interface: MCS[i]

Name	ATOM[i]_AGC_INT_TRIG
Address	$0x1C5C$
C-Name	

INT_TRIG[k]	
Description	Select input signal ATOM_CH_TRIGOUT[k:k] as a trigger source
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * k + 1 : 2 * k]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	modify
Condition	-

INT_TRIG[k]	
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
W-Coding	0b00 : don't care, bits will not be changed 0b01 : do not use internal trigger from channel k (*ATOM_CH_TRIGOUT* [k:k]) 0b10 : use internal trigger from channel k (*ATOM_CH_TRIGOUT* [k:k]) 0b11 : don't care, bits will not be changed
R-Coding	0b00 : internal trigger from channel k (*ATOM_CH_TRIGOUT* [k:k]) not used 0b01 : unused 0b10 : unused 0b11 : internal trigger from channel k (*ATOM_CH_TRIGOUT* [k:k]) used

15.5.9 ATOM[i]_CH[x]_CTRL

Description	ATOM[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_CTRL
Address	$0x20000 * i + 0x80 * x + 0x1804$
C-Name	GTM.CLS[i].ATOM.CH[x].CTRL

Interface: MCS[i]

Name	ATOM[i]_CH[x]_CTRL
Address	$0x80 * x + 0x1804$
C-Name	

MODE	
Description	ATOM channel mode select.
Loop	-
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

MODE	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : SOMI mode 1 : SOMC mode 2 : SOMP mode 3 : SOMS mode

TB12_SEL	
Description	Select time base value CCM[i]_TBU_TS1 or CCM[i]_TBU_TS2.
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

ARU_EN	
Description	ARU Input stream enable.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

ACB	
Description	ATOM Mode control bits.
Loop	–
Bit Range	[8 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

ACB	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

CMP_CTRL	
Description	CCU[x] compare strategy select.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

EUPM	
Description	Extended update mode
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

SL	
Description	Initial signal level.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–

SL	
Condition	–
Initial value	!ATOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

CLK_SRC	
Description	CMU clock source
Loop	–
Bit Range	[15 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

WR_REQ	
Description	CPU Write request bit for late compare register update.
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

TRIG_PULSE	
Description	Trigger output pulse length of one cluster clock period
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

TRIG_PULSE	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

UDMODE	
Description	Up/down counter mode
Loop	–
Bit Range	[19 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

RST_CCU0	
Description	Reset source of CCU0
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

OSM_TRIG	
Description	Enable trigger of one-shot pulse by the selected trigger signal
Loop	–
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

OSM_TRIG	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

EXT_TRIG	
Description	Select EXT_TRIGIN[x:x] as trigger signal
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

EXTTRIGOUT	
Description	Select EXT_TRIGIN[x:x] as potential output signal ATOM_CH_TRIGOUT[x:x]
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

TRIGOUT	
Description	Trigger output selection (output signal ATOM_CH_TRIGOUT[x:x]) of module ATOM [i] channel [x].
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

TRIGOUT	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

SLA	
Description	'Serve last' ARU communication strategy
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

OSM	
Description	One-shot mode
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

ABM	
Description	ARU blocking mode
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

ABM	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

EXT_FUPD	
Description	Enable force update by external trigger signal
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

SOMB	
Description	SOMB mode
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable SOMB mode 1 : Enable SOMB mode

FREEZE	
Description	ATOM Freeze Mode enable
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False

FREEZE	
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

Note:
Please see following multiview register description for more details.

15.5.10 ATOM[i]_CH[x]_CTRL_SOMI

Description	ATOM[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
View Condition	ATOM[i]_CH[x]_CTRL.MODE == 0 && ATOM[i]_CH[x]_CTRL.SOMB == 0
Register Reference	*15.5.9 "ATOM[i]_CH[x]_CTRL"*

Interface: CPU
Interface: MCS[i]

MODE	
Description	ATOM channel mode select.
Loop	-
Bit Range	[1 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : SOMI mode 1 : SOMC mode 2 : SOMP mode 3 : SOMS mode
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMI.MODE** = 0b00 for ATOM Signal Output Mode Immediate (SOMI)

TB12_SEL	
Description	Select time base value CCM[i]_TBU_TS1 or CCM[i]_TBU_TS2, but not applicable in SOMI mode.
Loop	-

TB12_SEL	
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

ARU_EN	
Description	ARU Input stream enable.
Loop	-
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : ARU input stream disabled 1 : ARU input stream enabled

ACB0	
Description	ATOM output control
Loop	-
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

ACB0	
RW-Coding	0 : Set output to the inverse value of **ATOM[i]_CH[x]_CTRL_SOMI.SL** bit 1 : Set output to **ATOM[i]_CH[x]_CTRL_SOMI.SL**

ACB41	
Description	ATOM Mode control bits but not applicable in SOMI mode.
Loop	–
Bit Range	$[8 : 5]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b0000 : Not applicable, initial value 0b0001 : Prohibited 0b0010 : Prohibited 0b0011 : Prohibited 0b0100 : Prohibited 0b0101 : Prohibited 0b0110 : Prohibited 0b0111 : Prohibited 0b1000 : Prohibited 0b1001 : Prohibited 0b1010 : Prohibited 0b1011 : Prohibited 0b1100 : Prohibited 0b1101 : Prohibited 0b1110 : Prohibited 0b1111 : Prohibited

CMP_CTRL	
Description	CCU[x] compare strategy selection, but not applicable in SOMI mode.
Loop	–
Bit Range	$[9 : 9]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

CMP_CTRL	
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EUPM	
Description	Extended update mode, but not applicable in SOMI mode.
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

SL	
Description	Initial signal level.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	!ATOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Low signal level 1 : High signal level
	Note: Reset value depends on the hardware configuration chosen by silicon vendor. **Note:** If the output of ATOM i channel x is disabled (*ATOM_OUTEN* [x:x]=0), the output *ATOM_OUT* [x:x] of instance i channel x will be set to the inverse value of **ATOM[i]_CH[x]_CTRL_SOMI.SL** . If the output is enabled (*ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] of instance i channel x will be set according to the configuration of **ATOM[i]_CH[x]_CTRL_SOMI** . The output is not relevant to *ATOM_ENDIS* [x:x] or **ATOM[i]_CH[x]_CTRL_SOMI.FREEZE** . The output *ATOM_OUT_T* [x:x] is not supported in SOMI mode.

CLK_SRC	
Description	CMU clock source but not applicable in SOMI mode

CLK_SRC	
Loop	–
Bit Range	[15 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b0000 : *CCM[i]_CLK_RES* [0:0] resolution in use 0b0001 : Prohibited 0b0010 : Prohibited 0b0011 : Prohibited 0b0100 : Prohibited 0b0101 : Prohibited 0b0110 : Prohibited 0b0111 : Prohibited 0b1000 : Prohibited 0b1001 : Prohibited 0b1010 : Prohibited 0b1011 : Prohibited 0b1100 : Prohibited 0b1101 : Prohibited 0b1110 : Prohibited 0b1111 : Prohibited

WR_REQ	
Description	CPU Write request bit for late compare register update, but not applicable in SOMI mode.
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

TRIG_PULSE	
Description	Trigger output pulse length of one cluster clock period, , but not applicable in SOMI mode.

TRIG_PULSE	
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

UDMODE	
Description	Up/down counter mode, but not applicable in SOMI mode.
Loop	–
Bit Range	[19 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Not applicable, initial value 0b01 : Prohibited 0b10 : Prohibited 0b11 : Prohibited

RST_CCU0	
Description	Reset source of CCU0, but not applicable in SOMI mode.
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

RST_CCU0	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

OSM_TRIG	
Description	Enable trigger of one-shot pulse by trigger signal ATOM_CH_TRIGOUT[x-1:x-1] or EXT_TRIGIN[x:x], but not applicable in SOMI mode.
Loop	-
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited
	Note: This bit may only be set if bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM** =1 and bit **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0.

EXT_TRIG	
Description	Select EXT_TRIGIN[x:x] as trigger signal, but not applicable in SOMI mode.
Loop	-
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXTTRIGOUT	
Description	Select EXT_TRIGIN[x:x] as potential output signal ATOM_CH_TRIGOUT[x:x]
Loop	-

EXTTRIGOUT	
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : signal *ATOM_CH_TRIGIN* [x:x] is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMI.TRIGOUT** = 0) 1 : signal *EXT_TRIGIN* [x:x] after synchronization to the selected clock resolution is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMI.TRIGOUT** = 0)

TRIGOUT	
Description	Trigger output selection (output signal ATOM_CH_TRIGOUT[x:x]) of module ATOM channel x.
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN* [x:x] 1 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_TRIG_CCU0*

SLA	
Description	'Serve last' ARU communication strategy, but not applicable in SOMI mode.
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

SLA	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

OSM	
Description	One-shot mode, but not applicable in SOMI mode.
Loop	-
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

ABM	
Description	ARU blocking mode, but not applicable in SOMI mode.
Loop	-
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXT_FUPD	
Description	External forced update
Loop	-
Bit Range	[29 : 29]
Access Type	RW
Volatile	False

EXT_FUPD	
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : use *ATOM_FUPD* [x:x] signal from AGC to force update 1 : use *EXT_TRIGIN* [x:x] signal to force update

SOMB	
Description	SOMB mode
Loop	-
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable SOMB mode, initial value 1 : Enable SOMB mode, prohibited
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMI.SOMB** = 0 for SOMI mode.

FREEZE	
Description	ATOM freeze mode enable configuration but it is not valid and supported in SOMI mode.
Loop	-
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

15.5.11 ATOM[i]_CH[x]_CTRL_SOMC

Description	ATOM[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
View Condition	ATOM[i]_CH[x]_CTRL.MODE == 1 && ATOM[i]_CH[x]_CTRL.SOMB == 0
Register Reference	*15.5.9 "ATOM[i]_CH[x]_CTRL"*

Interface: CPU
Interface: MCS[i]

MODE	
Description	ATOM channel mode select
Loop	–
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : SOMI mode 1 : SOMC mode 2 : SOMP mode 3 : SOMS mode
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMC.MODE** = 0b01 for ATOM Signal Output Mode Compare (SOMC)

TB12_SEL	
Description	Select time base value CCM[i]_TBU_TS1 or CCM[i]_TBU_TS2.
Loop	–
Bit Range	$[2 : 2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : *CCM[i]_TBU_TS1* selected for comparison 1 : *CCM[i]_TBU_TS2* selected for comparison

TB12_SEL	
	Note: This bit is only applicable if three time bases are present in the GTM-IP. Otherwise, this bit is reserved.

ARU_EN	
Description	ARU input stream enable
Loop	-
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : ARU input stream disabled 1 : ARU nput stream enabled

ACB10	
Description	Signal level control
Loop	-
Bit Range	[5 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN == 0 0b00 : No signal level change at output (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001). 0b01 : Set output signal level to 1 when **ATOM[i]_CH[x]_CTRL_SOMC.SL** = 0 else output signal level to 0 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001). 0b10 : Set output signal level to 0 when **ATOM[i]_CH[x]_CTRL_SOMC.SL** = 0 else output signal level to 1 (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001). 0b11 : Toggle output signal level (exception in table *31 "ATOM Serve first definition ATOM[i]_CH[x]_CTRL_SOMC.ACB42 = 0b001 or ATOM[i]_CH[x]_CTRL_STAT.ACBI = 0b001"* mode **ATOM[i]_CH[x]_CTRL_SOMC.ACB42** =001).

ACB10	
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN != 0 0b00 : Not applicable, initial value. 0b01 : Prohibited. 0b10 : Prohibited. 0b11 : Prohibited.
	Note: These bits are only applicable if **ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN** = 0.

ACB42	
Description	Compare strategy
Loop	–
Bit Range	$[8:6]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN == 0 0b000 : Compare in CCU0 and CCU1 in parallel, disable the CCUx on a compare match on either of compare units. Use *CCM[i]_TBU_TS0* in CCU0 and *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* in CCU1. 0b001 : Compare in CCU0 and CCU1 in parallel, disable the CCU0/CCU1 on a compare match on either compare units. Use *CCM[i]_TBU_TS0* in CCU0 and *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* in CCU1. 0b010 : Compare in CCU0 only against *CCM[i]_TBU_TS0* . 0b011 : Compare in CCU1 only against *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . 0b100 : Compare first in CCU0 and then in CCU1. Use *CCM[i]_TBU_TS0* . 0b101 : Compare first in CCU0 and then in CCU1. Use *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . 0b110 : Compare first in CCU0 and then in CCU1. Use *CCM[i]_TBU_TS0* in CCU0 and *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* in CCU1. 0b111 : Cancel pending compare events.
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN != 0 0b000 : Not applicable, initial value. 0b001 : Prohibited. 0b010 : Prohibited. 0b011 : Prohibited. 0b100 : Prohibited. 0b101 : Prohibited. 0b110 : Prohibited. 0b111 : Cancel pending compare events.
	Note: Independent of **ATOM[i]_CH[x]_CTRL_SOMC.ARU_EN** , a writing of 0b111 cancels any pending CCU0 or CCU1 compare.

CMP_CTRL	
Description	CCU[x] compare strategy select.
Loop	–

CMP_CTRL	
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Cyclic event compare of the selected time base (*CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2*) against **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** is performed when the selected TBU time base (*CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2*) is up-counting. Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more information about cyclic event compare strategy. 1 : Cyclic event compare of the selected time base (*CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2*) against **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** is performed when the selected TBU time base (*CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2*) is down-counting. Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more information about cyclic event compare strategy.
	Note: If the selected TBU time base is *CCM[i]_TBU_TS0* that can only be up-counting, cyclic event compare of the selected time base *CCM[i]_TBU_TS0* against **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** is performed independent on **ATOM[i]_CH[x]_CTRL_SOMC.CMP_CTRL** .

EUPM	
Description	Extended update mode
Loop	-
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : No extended update of **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** via configuration interface or ARU 1 : Extended update mode in case of compare strategy 'serve last': update of **ATOM[i]_CH[x]_CM1.CM1** after CCU0 compare match possible via ARU or configuration interface.
	Note: If **ATOM[i]_CH[x]_CTRL_SOMC.EUPM** =1 a write access to **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** never causes an AEI write status 0b10.

SL	
Description	Initial signal level.
Loop	-

SL	
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	!ATOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Low signal level 1 : High signal level
	Note: Reset value depends on the hardware configuration chosen by silicon vendor. **Note:** If the output is disabled (*ATOM_OUTEN* =0), the outputs *ATOM_OUT* [x:x] and *ATOM_OUT_T* [x:x] of instance i and channel x are set to ! **ATOM[i]_CH[x]_CTRL_SOMC.SL** . It is independent on **ATOM[i]_CH[x]_CTRL_SOMC.FREEZE** . **Note:** If **ATOM[i]_CH[x]_CTRL_SOMC.FREEZE** =0, when the channel is disabled (*ATOM_ENDIS* [x:x]=0) but the output is still enabled (*ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] will be set to the value of **ATOM[i]_CH[x]_CTRL_SOMC.SL** . Thus, if the channel is re-enabled afterwards (*ATOM_OUTEN* [x:x]=1 && *ATOM_ENDIS* [x:x]=1), the output *ATOM_OUT* [x:x] is initially **ATOM[i]_CH[x]_CTRL_SOMC.SL** . *ATOM_OUT_T* [x:x] is not supported in SOMC mode. **Note:** If **ATOM[i]_CH[x]_CTRL_SOMC.FREEZE** =1, when the channel is disabled (*ATOM_ENDIS* [x:x]=0) but the output is still enabled (*ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] will be not changed. Thus, if the channel is re-enabled afterwards (*ATOM_OUTEN* [x:x]=1 && *ATOM_ENDIS* [x:x]=1), the output *ATOM_OUT* [x:x] is initially the same state as it is before the channel is disabled. *ATOM_OUT_T* [x:x] is not supported in SOMC mode.

CLK_SRC	
Description	CMU clock source but not applicable in SOMC mode
Loop	–
Bit Range	[15 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

CLK_SRC	
RW-Coding	0b0000 : Not applicable, initial value 0b0001 : Prohibited 0b0010 : Prohibited 0b0011 : Prohibited 0b0100 : Prohibited 0b0101 : Prohibited 0b0110 : Prohibited 0b0111 : Prohibited 0b1000 : Prohibited 0b1001 : Prohibited 0b1010 : Prohibited 0b1011 : Prohibited 0b1100 : Prohibited 0b1101 : Prohibited 0b1110 : Prohibited 0b1111 : Prohibited

WR_REQ	
Description	CPU Write request bit for late compare register update.
Loop	-
Bit Range	[16 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : No late update requested by writing via configuration interface 1 : Late update requested by writing via configuration interface
	Note: If this bit is set to 1, subsequent ARU read requests by the channel can be disabled and a update of the shadow registers with new compare values can be performed by write access via the configuration interface, while the compare units operate on old compare values received by former ARU accesses. Then the compare registers **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** can be updated with the new values stored in the shadow registers by triggering a force update event. **Note:** With a compare match event or 2 compare match events at "serve last" strategy, the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit will be reset by hardware. **Note:** At the point of the force update only the shadow registers **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are transferred into the **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** registers. The output action is still defined by the **ATOM[i]_CH[x]_STAT.ACBI** bit field described by the ARU together with the old compare values for **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** .

TRIG_PULSE	
Description	Trigger output pulse length of one cluster clock period, , but not applicable in SOMC mode.
Loop	-

TRIG_PULSE	
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

UDMODE	
Description	Up/down counter mode, but not applicable in SOMC mode.
Loop	-
Bit Range	[19 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Not applicable, initial value 0b01 : Prohibited 0b10 : Prohibited 0b11 : Prohibited

RST_CCU0	
Description	Reset source of CCU0, but not applicable in SOMC mode.
Loop	-
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

RST_CCU0	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

OSM_TRIG	
Description	Enable trigger of one-shot pulse by trigger signal ATOM_CH_TRIGOUT[x-1:x-1] or EXT_TRIGIN[x:x], but not applicable in SOMC mode.
Loop	-
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXT_TRIG	
Description	Select EXT_TRIGIN[x:x] as trigger signal, but not applicable in SOMC mode.
Loop	-
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXTTRIGOUT	
Description	Select EXT_TRIGIN[x:x] as potential output signal ATOM_CH_TRIGOUT[x:x]
Loop	-
Bit Range	[23 : 23]
Access Type	RW

EXTTRIGOUT	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : signal *ATOM_CH_TRIGIN* [x:x] is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMC.TRIGOUT** = 0) 1 : signal *EXT_TRIGIN* [x:x] after synchronization to the selected clock resolution is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMC.TRIGOUT** = 0)

TRIGOUT	
Description	Trigger output selection (output signal ATOM_CH_TRIGOUT[x:x]) of module ATOM channel x.
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN* [x:x]. 1 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_TRIG_CCU0*

SLA	
Description	'Serve last' ARU communication strategy
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

SLA	
RW-Coding	bitrange(ATOM[i]_CH[x]_STAT.ACBI, 4, 2) == 4 || bitrange(ATOM[i]_CH[x]_STAT.ACBI, 4, 2) == 5 || bitrange(ATOM[i]_CH[x]_STAT.ACBI, 4, 2) == 6 0 : Captured **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** time stamps after CCU0 match event are not provided to ARU 1 : Captured **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** time stamps after CCU0 match event are provided to ARU
RW-Coding	bitrange(ATOM[i]_CH[x]_STAT.ACBI, 4, 2) != 4 && bitrange(ATOM[i]_CH[x]_STAT.ACBI, 4, 2) != 5 && bitrange(ATOM[i]_CH[x]_STAT.ACBI, 4, 2) != 6 0 : Not applicable, initial value 1 : Prohibited
	Note: Setting this bit has effect only, when **ATOM[i]_CH[x]_STAT.ACBI** [4:2] is configured for 'serve last' compare strategy ("100", "101", or "110"). **Note:** When this bit is not set, the captured time stamps in the shadow registers **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** are only provided after the CCU1 match occurred. The **ATOM[i]_CH[x]_STAT.ACBO** [4:3] bits always return "10" after the CCU1 match event occurred. **Note:** By setting this bit, the ATOM channel also provides the captured time stamps after the CCU0 match event to the ARU. The **ATOM[i]_CH[x]_STAT.ACBO** [4:3] bits are set to 0b01 in that case. After the CCU1 match event, the time stamps are captured again in the **ATOM[i]_CH[x]_SR0.SR0** / **ATOM[i]_CH[x]_SR1.SR1** registers and provided to the ARU. The **ATOM[i]_CH[x]_STAT.ACBO** [4:3] bits are set to 0b10. When the data in the shadow registers after the CCU0 match are not consumed by an ARU destination and the CCU1 match occurs, the data in the shadow registers will be overwritten by the new captured time stamps. In the "serve last" strategy, the ATOM channel does not request new data from the ARU before the CCU1 match event when the CCU0 match values are read from an ARU destination.

OSM	
Description	One-shot mode, but not applicable in SOMC mode.
Loop	-
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

ABM	
Description	ARU blocking mode
Loop	-
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False

ABM	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : ARU blocking mode disabled: ATOM reads continuously from ARU and updates **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_CTRL.ACB** bits, independent on pending compare match event. 1 : ARU blocking mode enabled: after update of **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_CTRL.ACB** bit, no new data is read via ARU until compare match event occurs and **ATOM[i]_CH[x]_SR0.SR0** and/or **ATOM[i]_CH[x]_SR1.SR1** are read via configuration interface or ARU.

EXT_FUPD	
Description	External forced update, but not applicable in SOMC mode.
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

SOMB	
Description	SOMB mode
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable SOMB mode, initial value 1 : Enable SOMB mode, prohibited

SOMB	
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMC.SOMB** = 0 in SOMC mode

FREEZE	
Description	ATOM freeze mode enable
Loop	–
Bit Range	$[31 : 31]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : a channel disable/enable may change internal register and output register 1 : a channel enable/disable does not change an internal or output register but stops comparison

15.5.12 ATOM[i]_CH[x]_CTRL_SOMP

Description	ATOM[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
View Condition	ATOM[i]_CH[x]_CTRL.MODE == 2 && ATOM[i]_CH[x]_CTRL.SOMB == 0
Register Reference	*15.5.9 "ATOM[i]_CH[x]_CTRL"*

Interface: CPU
Interface: MCS[i]

MODE	
Description	ATOM channel mode select
Loop	–
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

MODE	
RW-Coding	0 : SOMI mode 1 : SOMC mode 2 : SOMP mode 3 : SOMS mode
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMP.MODE** = 0b10 for ATOM Signal Output Mode PWM (SOMP)

TB12_SEL	
Description	Select time base value CCM[i]_TBU_TS1 or CCM[i]_TBU_TS2, but not applicable in SOMP mode.
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

ARU_EN	
Description	ARU input stream enable
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : ARU input stream disabled 1 : ARU input stream enabled

ADL	
Description	ARU data select
Loop	–
Bit Range	[5 : 4]

ADL	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Load both ARU words into shadow registers. 0b01 : Load ARU low word ([23:0]) into shadow register **ATOM[i]_CH[x]_SR0.SR0** . 0b10 : Load ARU high word ([47:24]) into shadow register **ATOM[i]_CH[x]_SR1.SR1** . 0b11 : Reserved.

BITREV	
Description	PCM mode enable
Loop	-
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	((x == 1) || (x == 3) || (x == 5) || (x == 7)) && (ATOM[i]_CH[x]_CTRL_SOMP.UDMODE == 0) 0 : PCM mode disabled (no bit-reversing of output of counter register **ATOM[i]_CH[x]_CN0.CN0**) 1 : PCM mode enabled (bit-reversing of output of counter register **ATOM[i]_CH[x]_CN0.CN0**)
RW-Coding	((x != 1) && (x != 3) && (x != 5) && (x != 7)) || (ATOM[i]_CH[x]_CTRL_SOMP.UDMODE != 0) 0 : Not applicable, initial value 1 : Prohibited
	Note: PCM mode is only available for odd numbered ATOM channels (channel 1,3,5,7). **Note:** PCM mode is only available in SOMP mode when **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** =0.

SR0_TRIG	
Description	SR0 used for ATOM_OUT_T of instance i and channel x
Loop	-
Bit Range	[7 : 7]
Access Type	RW
Volatile	False

SR0_TRIG	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0 == 0 0 : **ATOM[i]_CH[x]_SR0.SR0** is used as a shadow register for register **ATOM[i]_CH[x]_CM0.CM0** . 1 : **ATOM[i]_CH[x]_SR0.SR0** is not used as a shadow register for register **ATOM[i]_CH[x]_CM0.CM0** . **ATOM[i]_CH[x]_SR0.SR0** is compared with **ATOM[i]_CH[x]_CN0.CN0** and if both are equal, a trigger pulse is generated at output *ATOM_OUT_T* of instance i and channel x.
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0 != 0 0 : Not applicable, initial value 1 : Prohibited
	Note: This bit may only be set if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** of this channel is 0.

ACB4	
Description	ATOM output control, but not applicable in SOMP mode.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

CMP_CTRL	
Description	CCU[x] compare strategy selection, but not applicable in SOMP mode.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

CMP_CTRL	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EUPM	
Description	Extended update mode, but not applicable in SOMP mode.
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

SL	
Description	Initial signal level.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	!ATOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Low signal level 1 : High signal level

SL	
	Note: Reset value depends on the hardware configuration chosen by silicon vendor. **Note:** If the output is disabled(*ATOM_OUTEN* [x:x]=0), the outputs *ATOM_OUT* [x:x] and *ATOM_OUT_T* [x:x] of instance i and channel x are set to inverse **ATOM[i]_CH[x]_CTRL_SOMP.SL** . **Note:** If **ATOM[i]_CH[x]_CTRL_SOMP.FREEZE** =0, when the channel is disabled (*ATOM_ENDIS* [x:x]=0) and the output is enabled(*ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] will be set to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** and in continuous counting up-down mode with **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1 the other output *ATOM_OUT_T* [x:x] will be also set to ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** . Thus, when the channel is re-enabled (*ATOM_ENDIS* [x:x]=1 && *ATOM_OUTEN* [x:x]=1) afterwards, the output *ATOM_OUT* [x:x] is initially ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** and in continuous counting up-down mode with **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1 the other output *ATOM_OUT_T* [x:x] is also initially ! **ATOM[i]_CH[x]_CTRL_SOMP.SL** . In other SOMP modes, *ATOM_OUT_T* [x:x] is not supported. **Note:** If **ATOM[i]_CH[x]_CTRL_SOMP.FREEZE** =1, when the channel is disabled (*ATOM_ENDIS* [x:x]=0) and the output is enabled(*ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] will be not changed and in continuous counting up-down mode with **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1 the other output *ATOM_OUT_T* [x:x] will also be not changed. Thus, when the channel is re-enabled (*ATOM_ENDIS* [x:x]=1 && *ATOM_OUTEN* [x:x]=1) afterwards, the output *ATOM_OUT* [x:x] is initially the same state as it is before the channel is disabled and in continuous counting up-down mode with **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1 the other output *ATOM_OUT_T* [x:x] also initially remains the same. In other SOMP modes, *ATOM_OUT_T* [x:x] is not supported.

CLK_SRC	
Description	CMU clock source for SOMP mode
Loop	-
Bit Range	$[15:12]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b0000 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *CCM[i]_CLK_RES* [0:0] resolution is selected to be used 0b0001 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *CCM[i]_CLK_RES* [1:1] resolution is selected to be used 0b0010 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *CCM[i]_CLK_RES* [2:2] resolution is selected to be used 0b0011 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *CCM[i]_CLK_RES* [3:3] resolution is selected to be used 0b0100 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *CCM[i]_CLK_RES* [4:4] resolution is selected to be used 0b0101 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *CCM[i]_CLK_RES* [5:5] resolution is selected to be used 0b0110 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *CCM[i]_CLK_RES* [6:6] resolution is selected to be used 0b0111 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *CCM[i]_CLK_RES* [7:7] resolution is selected to be used ▽

CLK_SRC	
	△ 0b1000 : Prohibited 0b1001 : Prohibited 0b1010 : Prohibited 0b1011 : Prohibited 0b1100 : Functional operation stopped, clock resolution tied to zero. 0b1101 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *ATOM_CH_TRIGOUT* [x-1:x-1] is selected to be used 0b1110 : after its first update from the shadow register **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** , *EXT_TRIGIN* is selected to be used 0b1111 : Prohibited
	Note: The register **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** can be synchronously updated with the value of **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** together with the update of register **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** . **Note:** After (channel) reset the channel operation will run with *CLS[i]_CLK* . This fact is not reflected by reset value of **ATOM[i]_CH[x]_CTRL.CLK_SRC** bit field (which is set to 0b0000 after channel reset). Only directly writing **ATOM[i]_CH[x]_CTRL.CLK_SRC** does not change the CMU clock source selection. The value read from **ATOM[i]_CH[x]_CTRL.CLK_SRC** bit field after channel reset does not reflect the current clock source selection status. **Note:** After (channel) reset the channel operation will run with *CLS[i]_CLK* . In order to change the clock source resolution, it is necessary to perform a synchronous update (via **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR**) before enabling the channel. This can be done by performing the following configuration sequence before enabling the channel: **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** =<selected CCM clock source> **ATOM[i]_CH[x]_SR0** =0x0 **ATOM[i]_CH[x]_CM0** =0x0 **ATOM[i]_CH[x]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 0x10 With this configuration sequence, **ATOM[i]_CH[x]_CTRL.CLK_SRC** will be immediately updated with the clock source value stored in the shadow register. Afterwards once the channel is enabled, the channel will operate on the selected clock source. Since then, the value read from the register **ATOM[i]_CH[x]_CTRL.CLK_SRC** reflects the actual selected clock source of operation. **Note:** After the register has been updated once from its shadow register after reset state, it is possible to change the clock source also by directly writing **ATOM[i]_CH[x]_CTRL.CLK_SRC** . That means if the running channel is disabled but not reset afterwards, it is possible to directly configure **ATOM[i]_CH[x]_CTRL.CLK_SRC** with a new clock source selection. **Note:** The clock of the channel is stopped, if "0b1100" value is configured. Restarting a channel by writing a value for a valid clock selection again is possible.

WR_REQ	
Description	CPU Write request bit for late compare register update, but not applicable in SOMP mode.
Loop	-
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

WR_REQ	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

TRIG_PULSE	
Description	Trigger output pulse length of one cluster clock period
Loop	-
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG == 1 0 : output on *ATOM_OUT_T* [x:x] is '1' as long as **ATOM[i]_CH[x]_CN0.CN0** = **ATOM[i]_CH[x]_SR0.SR0** 1 : output on *ATOM_OUT_T* [x:x] is '1' for only one cluster clock period if **ATOM[i]_CH[x]_CN0.CN0** = **ATOM[i]_CH[x]_SR0.SR0**
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG == 0 0 : Not applicable, initial value 1 : Prohibited
	Attention: This bit may only be changed if bit **ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG** of this channel is zero.

UDMODE	
Description	Up/down counter mode
Loop	-
Bit Range	[19 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

UDMODE	
RW-Coding	0b00 : up/down counter mode disabled: **ATOM[i]_CH[x]_CN0.CN0** counts always up 0b01 : up/down counter mode enabled: **ATOM[i]_CH[x]_CN0.CN0** counts up and down, **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** are updated if **ATOM[i]_CH[x]_CN0.CN0** is 0 (i.e. the counting direction changes from down to up). 0b10 : up/down counter mode enabled: **ATOM[i]_CH[x]_CN0.CN0** counts up and down, **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** are updated if **ATOM[i]_CH[x]_CN0.CN0** reaches **ATOM[i]_CH[x]_CM0.CM0** -1 or the channel receives the trigger signal and the counting direction changes from up to down. 0b11 : up/down counter mode enabled: **ATOM[i]_CH[x]_CN0.CN0** counts up and down, **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** are updated if **ATOM[i]_CH[x]_CN0.CN0** is 0 or reaches **ATOM[i]_CH[x]_CM0.CM0** -1 or the channel receives the trigger signal and the counting direction changes from up to down.
	Note: The update mechanism is enabled by **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** = 1. Especially, if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** = 0 and **ATOM[i]_CH[x]_CTRL_SOMP.UDMODE** = 0b10 or 0b11, the update may occur when **ATOM[i]_CH[x]_CN0.CN0** matches **ATOM[i]_CH[x]_CM1.CM1** -1. In this case, it is not recommended to update **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC** and **ATOM[i]_CH[x]_CTRL_SOMP.SL** with a different value. The behavior after such an update may be unexpected.

RST_CCU0	
Description	Reset source of CCU0
Loop	-
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMP.OSM == 0 0 : Reset counter register **ATOM[i]_CH[x]_CN0.CN0** to 0 on matching comparison with **ATOM[i]_CH[x]_CM0.CM0** 1 : Reset counter register **ATOM[i]_CH[x]_CN0.CN0** to 0 on trigger *ATOM_CH_TRIGOUT* [x-1:x-1] or *EXT_TRIGIN* .
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMP.OSM != 0 0 : Not applicable, initial value 1 : Prohibited
	Note: If **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1 and **ATOM[i]_AGC_GLB_CTRL.UPEN_CTRL[x]** =1 are set, *ATOM_CH_TRIGOUT* [x-1:x-1] or *EXT_TRIGIN* [x:x] triggers also the update of work register (**ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** , **ATOM[i]_CH[x]_CTRL_SOMP.SL** and **ATOM[i]_CH[x]_CTRL_SOMP.CLK_SRC**). **Note:** This bit may only be set if bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM** =0 (i.e. in continuous mode)

OSM_TRIG	
Description	Enable trigger of one-shot pulse by trigger signal ATOM_CH_TRIGOUT[x-1:x-1] or EXT_TRIGIN[x:x]
Loop	-

OSM_TRIG	
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	(ATOM[i]_CH[x]_CTRL_SOMP.OSM == 1) && (ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0 == 0) 0 : The trigger signal *ATOM_CH_TRIGOUT* [x-1:x-1] or *EXT_TRIGIN* cannot trigger start of single pulse generation 1 : The trigger signal *ATOM_CH_TRIGOUT* [x-1:x-1] or *EXT_TRIGIN* generation can trigger start of single pulse (only if bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM** = 1)
RW-Coding	(ATOM[i]_CH[x]_CTRL_SOMP.OSM != 1) || (ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0 != 0) 0 : Not applicable, initial value 1 : Prohibited
	Note: This bit may only be set if bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM** =1 and bit **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0.

EXT_TRIG	
Description	Select EXT_TRIGIN[x:x] as trigger signal
Loop	-
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1 or **ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG** =1, signal *ATOM_CH_TRIGIN* [x:x] is selected as trigger to reset **ATOM[i]_CH[x]_CN0.CN0** or to start single pulse generation. 1 : if **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =1 or **ATOM[i]_CH[x]_CTRL_SOMP.OSM_TRIG** =1, signal *EXT_TRIGIN* [x:x] is selected as trigger to reset **ATOM[i]_CH[x]_CN0.CN0** or to start single pulse generation.

EXTTRIGOUT	
Description	Select EXT_TRIGIN[x:x] as potential output signal ATOM_CH_TRIGOUT[x:x]
Loop	-
Bit Range	[23 : 23]

EXTTRIGOUT	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : signal *ATOM_CH_TRIGIN* [x:x] is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMP.TRIGOUT** = 0) 1 : signal *EXT_TRIGIN* [x:x] after synchronization to the selected clock resolution is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMC.TRIGOUT** = 0)

TRIGOUT	
Description	Trigger output selection (output signal ATOM_CH_TRIGOUT[x:x]) of module ATOM [i] channel [x].
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN* [x:x] 1 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_TRIG_CCU0*

SLA	
Description	'Serve last' ARU communication strategy, but not applicable in SOMP mode.
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

SLA	
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

OSM	
Description	One-shot mode
Loop	-
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Continuous PWM generation after channel enable 1 : A single pulse is generated

ABM	
Description	ARU blocking mode, but not applicable in SOMP mode.
Loop	-
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXT_FUPD	
Description	External forced update
Loop	-
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

EXT_FUPD	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : use *ATOM_FUPD* [x:x] signal from AGC to force update 1 : use *EXT_TRIGIN* [x:x] signal to force update

SOMB	
Description	SOMB mode
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable SOMB mode, initial value 1 : Enable SOMB mode, prohibited
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMP.SOMB** = 0 for SOMP mode

FREEZE	
Description	ATOM Freeze Mode enable
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : a channel disable/enable may change internal register and output register 1 : a channel enable/disable does not change an internal or output register but stops counter **ATOM[i]_CH[x]_CN0.CN0**

15.5.13 ATOM[i]_CH[x]_CTRL_SOMS

Description	ATOM[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
View Condition	ATOM[i]_CH[x]_CTRL.MODE == 3 && ATOM[i]_CH[x]_CTRL.SOMB == 0
Register Reference	*15.5.9 "ATOM[i]_CH[x]_CTRL"*

Interface: CPU
Interface: MCS[i]

MODE	
Description	ATOM channel mode select.
Loop	–
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : SOMI mode 1 : SOMC mode 2 : SOMP mode 3 : SOMS mode
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMS.MODE** = 0b11 for ATOM Signal Output Mode Serial (SOMS)

TB12_SEL	
Description	Select time base value CCM[i]_TBU_TS1 or CCM[i]_TBU_TS2, but not applicable in SOMS mode.
Loop	–
Bit Range	$[2 : 2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

TB12_SEL	
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

ARU_EN	
Description	ARU Input stream enable.
Loop	-
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : ARU Input stream disabled 1 : ARU Input stream enabled

ACB0	
Description	Shift direction
Loop	-
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN == 0 0 : Right shift of data is started from bit **ATOM[i]_CH[x]_CM1.CM1** [0:0]. 1 : Left shift of data is started from bit **ATOM[i]_CH[x]_CM1.CM1** [23:23].
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN == 1 0 : Not applicable, initial value. 1 : Prohibited.

ACB0	
	Note: The data that has to be shifted out to be aligned inside the **ATOM[i]_CH[x]_CM1.CM1** register according to the defined shift direction. **Note:** This bit is only applicable if **ATOM[i]_CH[x]_CTRL_SOMS.ARU_EN** = 0. **Note:** If the direction (**ATOM[i]_CH[x]_CTRL_SOMS.ACB0**) is changed, the output *ATOM_OUT* switches immediately to the other 'first' bit of **ATOM[i]_CH[x]_CM1.CM1** (bit 0 if **ATOM[i]_CH[x]_CTRL_SOMS.ACB0** =0, bit 23 if **ATOM[i]_CH[x]_CTRL_SOMS.ACB0** = 1).

ACB21	
Description	ATOM output control, but not applicable in SOMS mode.
Loop	–
Bit Range	[6 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Not applicable, initial value 0b01 : Prohibited 0b10 : Prohibited 0b11 : Prohibited

DSO	
Description	Double shift output
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : **ATOM[i]_CH[x]_CM1.CM1** is used as a 24-bit shift register. 1 : **ATOM[i]_CH[x]_CM1.CM1** is split into two 12-bit shift register.

DSO	
	Note: If **ATOM[i]_CH[x]_CTRL_SOMS.DSO** =1 , only shift right is possible.

ACB4	
Description	ATOM output control, but not applicable in SOMS mode.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

CMP_CTRL	
Description	CCU[x] compare strategy selection, but not applicable in SOMS mode.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EUPM	
Description	Extended update mode, but not applicable in SOMS mode.
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

EUPM	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

SL	
Description	Initial signal level.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	!ATOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Low signal level 1 : High signal level
	Note: Reset value depends on the hardware configuration chosen by silicon vendor. **Note:** The inverse value of **ATOM[i]_CH[x]_CTRL_SOMS.SL** is shifted into the **ATOM[i]_CH[x]_CM1.CM1** register. **Note:** If the output of ATOM i channel x is disabled (*ATOM_OUTEN* [x:x]=0), both ATOM channel outputs *ATOM_OUT* and *ATOM_OUT_T* of instance i channel x are the inverse value of bit **ATOM[i]_CH[x]_CTRL.SL** . **Note:** If the output of ATOM i channel x is enabled (*ATOM_OUTEN* [x:x]=1) but the channel is disabled (*ATOM_ENDIS* [x:x]=0), the output *ATOM_OUT* [x:x] will be set as **ATOM[i]_CH[x]_CM1.CM1** [0:0] when configured as right shift or **ATOM[i]_CH[x]_CM1.CM1** [23:23] when configured as left shift. Afterwards when the channel is re-enabled (*ATOM_OUTEN* [x:x]=1 && *ATOM_ENDIS* [x:x]=1), the output *ATOM_OUT* [x:x] is initially **ATOM[i]_CH[x]_CM1.CM1** [0:0] or **ATOM[i]_CH[x]_CM1.CM1** [23:23]. **Note:** The output *ATOM_OUT_T* [x:x] is supported in double shift mode but not supported in single shift mode. In double shift mode, if **ATOM[i]_CH[x]_CTRL_SOMS.FREEZE** =0, when the output of ATOM i channel x is enabled (*ATOM_OUTEN* [x:x]=1) but the channel is disabled (*ATOM_ENDIS* [x:x]=0), the output *ATOM_OUT_T* [x:x] is set to ! **ATOM[i]_CH[x]_CTRL.SL** . Afterwards when the channel is re-enabled (*ATOM_OUTEN* [x:x]=1 && *ATOM_ENDIS* [x:x]=1), the output *ATOM_OUT_T* [x:x] that is initially ! **ATOM[i]_CH[x]_CTRL.SL** . The output *ATOM_OUT_T* [x:x] is supported in double shift mode but not supported in single shift mode. In double shift mode, if **ATOM[i]_CH[x]_CTRL_SOMS.FREEZE** =1, when the output of ATOM i channel x is enabled (*ATOM_OUTEN* [x:x]=1) but the channel is disabled (*ATOM_ENDIS* [x:x]=0), the output *ATOM_OUT_T* [x:x] will not be changed. Afterwards when the channel is re-enabled (*ATOM_OUTEN* [x:x]=1 && *ATOM_ENDIS* [x:x]=1), the output *ATOM_OUT_T* [x:x] that is initially the same state as it is when deactivated(*ATOM_ENDIS* [x:x]=0) last time.

CLK_SRC	
Description	CMU clock source for SOMS mode
Loop	-
Bit Range	[15 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b0000 : *CCM[i]_CLK_RES* [0:0] resolution in use 0b0001 : *CCM[i]_CLK_RES* [1:1] resolution in use 0b0010 : *CCM[i]_CLK_RES* [2:2] resolution in use 0b0011 : *CCM[i]_CLK_RES* [3:3] resolution in use 0b0100 : *CCM[i]_CLK_RES* [4:4] resolution in use 0b0101 : *CCM[i]_CLK_RES* [5:5] resolution in use 0b0110 : *CCM[i]_CLK_RES* [6:6] resolution in use 0b0111 : *CCM[i]_CLK_RES* [7:7] resolution in use 0b1000 : Prohibited 0b1001 : Prohibited 0b1010 : Prohibited 0b1011 : Prohibited 0b1100 : Functional operation stopped, clock resolution tied to zero 0b1101 : *ATOM_CH_TRIGOUT* [x-1:x-1] selected 0b1110 : *EXT_TRIGIN* selected 0b1111 : Prohibited
	Note: The register **ATOM[i]_CH[x]_CTRL_SOMS.CLK_SRC** is updated with the value of **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** together with the update of register **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** . **Note:** After (channel) reset the selected clock source value is the cluster clock (input of Global Clock Divider). **Note:** The clock of the channel is stopped, if "0b1100" value is configured. Restarting a channel by writing a value for a valid clock selection again is possible.

WR_REQ	
Description	CPU Write request bit for late compare register update, but not applicable in SOMS mode.
Loop	-
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

WR_REQ	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

TRIG_PULSE	
Description	Trigger output pulse length of one cluster clock period, , but not applicable in SOMS mode.
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

UDMODE	
Description	Up/down counter mode, but not applicable in SOMS mode.
Loop	–
Bit Range	[19 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Not applicable, initial value 0b01 : Prohibited 0b10 : Prohibited 0b11 : Prohibited

RST_CCU0	
Description	Reset source of CCU0, but not applicable in SOMS mode.
Loop	–

RST_CCU0	
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

OSM_TRIG	
Description	Enable trigger of one-shot pulse by trigger signal ATOM_CH_TRIGOUT[x-1:x-1] or EXT_TRIGIN[x:x], but not applicable in SOMS mode.
Loop	-
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXT_TRIG	
Description	Select EXT_TRIGIN[x:x] as trigger signal, but not applicable in SOMS mode.
Loop	-
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

EXT_TRIG	
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXTTRIGOUT	
Description	Select EXT_TRIGIN[x:x] as potential output signal ATOM_CH_TRIGOUT[x:x]
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : signal *ATOM_CH_TRIGIN* [x:x] is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMS.TRIGOUT** = 0) 1 : signal *EXT_TRIGIN* after synchronization to the selected clock resolution is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMS.TRIGOUT** = 0)

TRIGOUT	
Description	Trigger output selection (output signal ATOM_CH_TRIGOUT[x:x]) of module ATOM channel x.
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN* [x:x]. 1 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_TRIG_CCU0*

SLA	
Description	'Serve last' ARU communication strategy, but not applicable in SOMS mode.
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False

SLA	
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

OSM	
Description	One-shot mode
Loop	-
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Continuous PWM generation after channel enable 1 : A single pulse is generated

ABM	
Description	ARU blocking mode, but not applicable in SOMS mode.
Loop	-
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXT_FUPD	
Description	External forced update

EXT_FUPD	
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : use *ATOM_FUPD* [x:x] signal from AGC to force update 1 : use *EXT_TRIGIN* signal to force update

SOMB	
Description	SOMB mode
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable SOMB mode, initial value 1 : Enable SOMB mode, prohibited
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMS.SOMB** = 0 for SOMS mode.

FREEZE	
Description	ATOM Freeze Mode enable
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

FREEZE	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
	Note: If channel is disabled and output is enabled, the output depends directly on **ATOM[i]_CH[x]_CTRL_SOMS.SL** bit, irrespective of **ATOM[i]_CH[x]_CTRL_SOMS.FREEZE** mode.

15.5.14 ATOM[i]_CH[x]_CTRL_SOMB

Description	ATOM[i] channel [x] control register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
View Condition	ATOM[i]_CH[x]_CTRL.SOMB == 1
Register Reference	*15.5.9 "ATOM[i]_CH[x]_CTRL"*

Interface: CPU
Interface: MCS[i]

MODE	
Description	ATOM channel mode select, but not applicable in SOMB mode.
Loop	–
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited 2 : Prohibited 3 : Prohibited
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMB.MODE** = 0b00 for ATOM Signal Output Mode Compare (SOMB)

TB12_SEL	
Description	Select time base value CCM[i]_TBU_TS1 or CCM[i]_TBU_TS2.
Loop	–
Bit Range	$[2 : 2]$
Access Type	RW
Volatile	False

TB12_SEL	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : *CCM[i]_TBU_TS1* selected for comparison 1 : *CCM[i]_TBU_TS2* selected for comparison
	Note: This bit is only applicable if three time bases are present in the GTM-IP. Otherwise, this bit is reserved.

ARU_EN	
Description	ARU Input stream enable.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : ARU Input stream disabled 1 : ARU Input stream enabled

ACB10	
Description	Signal level control
Loop	–
Bit Range	[5 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

ACB10	
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN == 0 0b00 : No signal level change at output. 0b01 : Set output signal level to 1 when **ATOM[i]_CH[x]_CTRL_SOMB.SL** bit = 0 else output signal level to 0. 0b10 : Set output signal level to 0 when **ATOM[i]_CH[x]_CTRL_SOMB.SL** bit = 0 else output signal level to 1. 0b11 : Toggle output signal level.
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN != 0 0b00 : Not applicable, initial value. 0b01 : Prohibited. 0b10 : Prohibited. 0b11 : Prohibited.
	Note: For details see *38 "ATOM SOMB output control by ATOM[i]_CH[x]_STAT.ACBI[1:0] and ATOM[i]_CH[x]_CTRL_SOMB.SL"* **Note:** These bits are only applicable if **ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN** = 0.

ACB42	
Description	Compare strategy
Loop	–
Bit Range	$[8:6]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN == 0 0b000 : Reserved. Has no effect. 0b001 : Reserved. Has no effect. 0b010 : Compare in CCU0 only against *CCM[i]_TBU_TS0* . 0b011 : Compare in CCU1 only against *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . 0b100 : Compare first in CCU0 and then in CCU1. Use *CCM[i]_TBU_TS0* . 0b101 : Compare first in CCU0 and then in CCU1. Use *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* . 0b110 : Compare first in CCU0 and then in CCU1. Use *CCM[i]_TBU_TS0* in CCU0 and *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* in CCU1. 0b111 : Cancel pending comparisons independent of **ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN** .
RW-Coding	ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN != 0 0b000 : Not applicable, initial value. 0b001 : Prohibited. 0b010 : Prohibited. 0b011 : Prohibited. 0b100 : Prohibited. 0b101 : Prohibited. 0b110 : Prohibited. 0b111 : Cancel pending compare events.

ACB42	
	Note: For details see *37 "ATOM SOMB compare strategies"* **Note:** These bits are only applicable if **ATOM[i]_CH[x]_CTRL_SOMB.ARU_EN** = 0.

CMP_CTRL	
Description	CCU[x] compare strategy select.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Cyclic event compare of the selected time base (*CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2*) against **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** is performed when the selected TBU time base (*CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2*) is up-counting. Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more information about cyclic event compare strategy. 1 : Cyclic event compare of the selected time base (*CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2*) against **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** is performed when the selected TBU time base (*CCM[i]_TBU_TS1* / *CCM[i]_TBU_TS2*) is down-counting. Please refer to chapter *3.11.1 "Cyclic Event Compare"* for more information about cyclic event compare strategy.
	Note: If the selected TBU time base is *CCM[i]_TBU_TS0* that can only be up-counting, cyclic event compare of the selected time base *CCM[i]_TBU_TS0* against **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** is performed independent on **ATOM[i]_CH[x]_CTRL_SOMB.CMP_CTRL** .

EUPM	
Description	Extended update mode
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

EUPM	
RW-Coding	0 : No extended update of **ATOM[i]_CH[x]_CM0.CM0** and **ATOM[i]_CH[x]_CM1.CM1** via configuration interface or ARU 1 : Extended update mode in case of compare strategy 'serve last': update of **ATOM[i]_CH[x]_CM1.CM1** after CCU0 compare match possible via ARU or configuration interface.
	Note: If **ATOM[i]_CH[x]_CTRL_SOMB.EUPM** =1 a write access to **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** never causes an AEI write status 0b10.

SL	
Description	Initial signal level.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	!ATOM_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Low signal level 1 : High signal level
	Note: Reset value depends on the hardware configuration chosen by silicon vendor. **Note:** If the output is disabled (*ATOM_OUTEN* [x:x]=0), the outputs *ATOM_OUT* [x:x] and *ATOM_OUT_T* [x:x] of instance i and channel x are set to ! **ATOM[i]_CH[x]_CTRL_SOMB.SL** . It is independent on **ATOM[i]_CH[x]_CTRL_SOMB.FREEZE** . **Note:** If **ATOM[i]_CH[x]_CTRL_SOMB.FREEZE** =0, when the channel is disabled (*ATOM_ENDIS* [x:x]=0) but the output is still enabled (*ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] is set to the value of **ATOM[i]_CH[x]_CTRL_SOMB.SL** . Thus, if the channel is enabled afterwards (*ATOM_ENDIS* [x:x]=1 && *ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] is initially **ATOM[i]_CH[x]_CTRL_SOMB.SL** . *ATOM_OUT_T* [x:x] is not supported in SOMB mode. **Note:** If **ATOM[i]_CH[x]_CTRL_SOMB.FREEZE** =1, when the channel is disabled (*ATOM_ENDIS* [x:x]=0) but the output is still enabled (*ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] is not changed. Thus, if the channel is enabled afterwards (*ATOM_ENDIS* [x:x]=1 && *ATOM_OUTEN* [x:x]=1), the output *ATOM_OUT* [x:x] is initially the same state as it is before the channel is disabled. *ATOM_OUT_T* [x:x] is not supported in SOMB mode.

CLK_SRC	
Description	CMU clock source but not applicable in SOMB mode
Loop	–
Bit Range	[15 : 12]
Access Type	RW
Volatile	False
Multithread	False

CLK_SRC	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b0000 : Not applicable, initial value 0b0001 : Prohibited 0b0010 : Prohibited 0b0011 : Prohibited 0b0100 : Prohibited 0b0101 : Prohibited 0b0110 : Prohibited 0b0111 : Prohibited 0b1000 : Prohibited 0b1001 : Prohibited 0b1010 : Prohibited 0b1011 : Prohibited 0b1100 : Prohibited 0b1101 : Prohibited 0b1110 : Prohibited 0b1111 : Prohibited

WR_REQ	
Description	CPU Write request bit for late compare register update.
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : No late update requested by writing via configuration interface 1 : Late update requested by writing via configuration interface

WR_REQ	
	Note: If this bit is set to 1, subsequent ARU read requests by the channel can be disabled and a update of the shadow registers with new compare values can be performed by write access via the configuration interface, while the compare units operate on old compare values received by former ARU accesses. Then the compare registers **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** can be updated with the new values stored in the shadow registers by triggering a force update event. **Note:** With a compare match event or 2 compare match events at "serve last" strategy, the **ATOM[i]_CH[x]_CTRL_SOMC.WR_REQ** bit will be reset by hardware. **Note:** At the point of the force update only the shadow registers **ATOM[i]_CH[x]_SR0.SR0** and **ATOM[i]_CH[x]_SR1.SR1** are transferred into the **ATOM[i]_CH[x]_CM0.CM0** , **ATOM[i]_CH[x]_CM1.CM1** registers. The output action is still defined by the **ATOM[i]_CH[x]_STAT.ACBI** bit field described by the ARU together with the old compare values for **ATOM[i]_CH[x]_CM0.CM0** / **ATOM[i]_CH[x]_CM1.CM1** .

TRIG_PULSE	
Description	Trigger output pulse length of one cluster clock period, , but not applicable in SOMB mode.
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

UDMODE	
Description	Up/down counter mode, but not applicable in SOMB mode.
Loop	–
Bit Range	[19 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

UDMODE	
RW-Coding	0b00 : Not applicable, initial value 0b01 : Prohibited 0b10 : Prohibited 0b11 : Prohibited

RST_CCU0	
Description	Reset source of CCU0, but not applicable in SOMB mode.
Loop	-
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

OSM_TRIG	
Description	Enable trigger of one-shot pulse by trigger signal ATOM_CH_TRIGOUT[x-1:x-1] or EXT_TRIGIN[x:x], but not applicable in SOMB mode.
Loop	-
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited
	Note: This bit may only be set if bit **ATOM[i]_CH[x]_CTRL_SOMP.OSM** =1 and bit **ATOM[i]_CH[x]_CTRL_SOMP.RST_CCU0** =0.

EXT_TRIG	
Description	Select EXT_TRIGIN[x:x] as trigger signal, but not applicable in SOMB mode.
Loop	-

EXT_TRIG	
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

EXTTRIGOUT	
Description	Select EXT_TRIGIN[x:x] as potential output signal ATOM_CH_TRIGOUT[x:x]
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : signal *ATOM_CH_TRIGIN* [x:x] is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMB.TRIGOUT** = 0) 1 : signal *EXT_TRIGIN* after synchronization to the selected clock resolution is selected as output on *ATOM_CH_TRIGOUT* [x:x] (if **ATOM[i]_CH[x]_CTRL_SOMB.TRIGOUT** = 0)

TRIGOUT	
Description	Trigger output selection (output signal ATOM_CH_TRIGOUT[x:x]) of module ATOM [i] channel [x].
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

TRIGOUT	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_CH_TRIGIN* [x:x] or *EXT_TRIGIN* [x:x]. 1 : *ATOM_CH_TRIGOUT* [x:x] is *ATOM_TRIG_CCU0*

SLA	
Description	'Serve last' ARU communication strategy, but not applicable in SOMP mode.
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

OSM	
Description	One-shot mode, but not applicable in SOMB mode.
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Not applicable, initial value 1 : Prohibited

ABM	
Description	ARU blocking mode
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False

ABM	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : ARU blocking mode disabled: ATOM reads continuously from ARU and updates **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** and **ATOM[i]_CH[x]_CTRL.ACB** bits, independent on pending compare match event. 1 : ARU blocking mode enabled: after update of **ATOM[i]_CH[x]_SR0.SR0** , **ATOM[i]_CH[x]_SR1.SR1** and **ATOM[i]_CH[x]_CTRL.ACB** bits via ARU, no new data is read via ARU until compare match event occurs.

EXT_FUPD	
Description	External forced update
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : use *ATOM_FUPD* [x:x] signal from AGC to force update 1 : use *EXT_TRIGIN* [x:x] signal to force update

SOMB	
Description	SOMB mode
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable SOMB mode 1 : Enable SOMB mode

SOMB	
	Note: Set **ATOM[i]_CH[x]_CTRL_SOMB.SOMB** = 1 for ATOM SOMB mode.

FREEZE	
Description	ATOM Freeze Mode enable
Loop	–
Bit Range	$[31 : 31]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : a channel disable/enable may change internal register and output register 1 : a channel enable/disable does not change an internal or output register but stops comparison

15.5.15 ATOM[i]_CH[x]_CTRL_SR

Description	ATOM[i] channel [x] control shadow register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ATOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_CTRL_SR
Address	$0x20000 * i + 0x80 * x + 0x1834$
C-Name	GTM.CLS[i].ATOM.CH[x].CTRL_SR

Interface: MCS[i]

Name	ATOM[i]_CH[x]_CTRL_SR
Address	$0x80 * x + 0x1834$
C-Name	

SL_SR	
Description	Shadow register for ATOM[i]_CH[x]_CTRL.SL
Loop	–
Bit Range	$[11 : 11]$
Access Type	RW

SL_SR	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : Low signal level 1 : High signal level
	Note: Reset value depends on the hardware configuration chosen by silicon vendor.

CLK_SRC_SR	
Description	Shadow register for ATOM[i]_CH[x]_CTRL.CLK_SRC
Loop	–
Bit Range	[15 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b0000 : *CCM[i]_CLK_RES* [0:0] resolution in use 0b0001 : *CCM[i]_CLK_RES* [1:1] resolution in use 0b0010 : *CCM[i]_CLK_RES* [2:2] resolution in use 0b0011 : *CCM[i]_CLK_RES* [3:3] resolution in use 0b0100 : *CCM[i]_CLK_RES* [4:4] resolution in use 0b0101 : *CCM[i]_CLK_RES* [5:5] resolution in use 0b0110 : *CCM[i]_CLK_RES* [6:6] resolution in use 0b0111 : *CCM[i]_CLK_RES* [7:7] resolution in use 0b1000 : Reserved 0b1001 : Reserved 0b1010 : Reserved 0b1011 : Reserved 0b1100 : Reserved 0b1101 : *ATOM_CH_TRIGOUT* [x-1:x-1] selected 0b1110 : *EXT_TRIGIN* selected 0b1111 : Reserved

CLK_SRC_SR	
	Note: This register is a shadow register for **ATOM[i]_CH[x]_CTRL.CLK_SRC** . Thus, if the *CCM[i]_CLK_RES* source for PWM generation should be changed during operation, the old *CCM[i]_CLK_RES* has to operate until the ATOM channel's **ATOM[i]_CH[x]_CTRL.CLK_SRC** register is updated with the content of **ATOM[i]_CH[x]_CTRL_SR.CLK_SRC_SR** either by the end of a period or a forced update.

15.5.16 ATOM[i]_CH[x]_CTRL2

Description	ATOM[i] channel [x] control2 register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	$x < \text{ATOM\_HIGH\_RES}[i]$
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_CTRL2
Address	$0x20000 * i + 0x80 * x + 0x1830$
C-Name	GTM.CLS[i].ATOM.CH[x].CTRL2

Interface: MCS[i]

Name	ATOM[i]_CH[x]_CTRL2
Address	$0x80 * x + 0x1830$
C-Name	

HRES	
Description	ATOM[i]_CH[x]_CTRL2.HRES: ATOM high resolution support
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	(x < ATOM_HIGH_RES[i]) && (ATOM[i]_CH[x]_CTRL.MODE == 2) 0 : High resolution support disabled 1 : High resolution support enabled
RW-Coding	(x >= ATOM_HIGH_RES[i]) || (ATOM[i]_CH[x]_CTRL.MODE != 2) 0 : Not applicable, initial value 1 : Prohibited

HRES	
	Note: **ATOM[i]_CH[x]_CTRL2.HRES** mode is available in connection with ATOM SOMP mode only.

15.5.17 ATOM[i]_CH[x]_STAT

Description	ATOM[i] channel [x] status register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ATOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_STAT
Address	$0x20000 * i + 0x80 * x + 0x181C$
C-Name	GTM.CLS[i].ATOM.CH[x].STAT

Interface: MCS[i]

Name	ATOM[i]_CH[x]_STAT
Address	$0x80 * x + 0x181C$
C-Name	

OL	
Description	Output signal level of ATOM_OUT[x:x].
Loop	-
Bit Range	$[0 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$\mathrm{ATOM\_OUT\_RST}$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	0 : Output signal level is low 1 : Output signal level is high
	Note: Reset value is the inverted value of bit **ATOM[i]_CH[x]_CTRL.SL** which depends on the hardware configuration chosen by silicon vendor.

ACBI	
Description	ATOM Mode control bits.

ACBI	
Loop	–
Bit Range	[20 : 16]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
	Note: For ATOM SOMI, SOMC, SOMP and SOMS mode this register serves as a mirror for the five ARU control bits received through the ARU interface. The bits are valid, when the **ATOM[i]_CH[x]_STAT.DV** bit is set. **Note:** For SOMB mode this bit field serves as the work register of the compare strategy. It can be updated with the value of bit field **ATOM[i]_CH[x]_CTRL.ACB** or the value of internal shadow register ACB_SR.

DV	
Description	Valid ARU Data stored in compare registers.
Loop	–
Bit Range	[21 : 21]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	ATOM[i]_CH[x]_CTRL.MODE != 0 || ATOM[i]_CH[x]_CTRL.SOMB != 0 0 : In SOMC/SOMB mode, no valid data (written by AEI or received from ARU) is stored in registers **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_CTRL.ACB/ATOM[i]_CH[x]_STAT.ACBI** . Therefore when the channel is enabled, no comparison is activated. In SOMP/SOMS mode, no valid ARU data stored in register **ATOM[i]_CH[x]_SR0.SR0** and/or **ATOM[i]_CH[x]_SR1.SR1** and **ATOM[i]_CH[x]_STAT.ACBI** . 1 : In SOMC/SOMB mode, new valid data (written by AEI or received from ARU) is stored in registers **ATOM[i]_CH[x]_CM0.CM0** and/or **ATOM[i]_CH[x]_CM1.CM1** and **ATOM[i]_CH[x]_CTRL.ACB/ATOM[i]_CH[x]_STAT.ACBI** . Therefore when the channel is enabled, comparison can be activated depending on **ATOM[i]_CH[x]_CTRL.ACB** [4:2]/ **ATOM[i]_CH[x]_STAT.ACBI** [4:2]. In SOMP/SOMS mode, new valid ARU data stored in register **ATOM[i]_CH[x]_SR0.SR0** and/or **ATOM[i]_CH[x]_SR1.SR1** and **ATOM[i]_CH[x]_STAT.ACBI** .
R-Coding	ATOM[i]_CH[x]_CTRL.MODE == 0 && ATOM[i]_CH[x]_CTRL.SOMB == 0 0 : Not applicable, initial value 1 : Prohibited

DV	
	Note: In SOMI mode, the bit is not supported (**ATOM[i]_CH[x]_STAT.DV** = 0). The status of the ARU transfers can be determined via the configuration interface with this bit. In SOMC/SOMB modes, when channel is enabled, the bit is reset by hardware with the cancel comparison configuration (**ATOM[i]_CH[x]_CTRL_SOMC.ACB4-2/ATOM[i]_CH[x]_CTRL_SOMB.ACB42** = 0b111) or when the specified compare event occurs. In SOMP/SOMS mode, after new data are received from ARU, the bit is reset again with starting of the next period of the received data and a new ARU read request.

WRF	
Description	Write request of CPU failed for late update.
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	0 : Late update was successful, CCU0/CCU1 units wait for comparison. 1 : Late update failed.
W-Coding	0 : No action 1 : Clear status bit to zero
	Note: The bit **ATOM[i]_CH[x]_STAT.WRF** can be cleared by writing a 1 to it. **Note:** This bit is only applicable in SOMC and SOMB mode.

DR	
Description	ARU data rejected flag
Loop	–
Bit Range	[23 : 23]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	0 : received ARU data stored 1 : received ARU data rejected

DR	
	Note: The flag is cleared if valid data is received and stored via ARU.

ACBO	
Description	ATOM Internal status bits.
Loop	-
Bit Range	[28 : 24]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	ATOM[i]_CH[x]_CTRL.MODE == 1 && ATOM[i]_CH[x]_CTRL.SOMB == 0 0b00000 : no compare match occurred 0b01000 : CCU0 compare match occurred 0b10000 : CCU1 compare match occurred
R-Coding	ATOM[i]_CH[x]_CTRL.MODE != 1 || ATOM[i]_CH[x]_CTRL.SOMB == 1 0b00000 : Not applicable, initial value 0b01000 : Prohibited 0b10000 : Prohibited
	Note: These bits are only set in SOMC mode. **Note:** **ATOM[i]_CH[x]_STAT.ACBO** is reset to 0b00000 on an update of register **ATOM[i]_CH[x]_CM0.CM0** or **ATOM[i]_CH[x]_CM1.CM1** (via ARU or configuration interface) **Note:** In SOMC mode these bits are sent as ARU control bits [52:48].

OSM_RTF	
Description	One-shot mode retrigger failed flag.
Loop	-
Bit Range	[29 : 29]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-

OSM_RTF	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	ATOM[i]_CH[x]_CTRL.MODE == 2 && ATOM[i]_CH[x]_CTRL.SOMB == 0 0 : One-shot retrigger operation successful 1 : One-shot retrigger operation failed
RW-Coding	ATOM[i]_CH[x]_CTRL.MODE != 2 || ATOM[i]_CH[x]_CTRL.SOMB != 0 0 : Not applicable, initial value 1 : Prohibited
	Note: This bit will be cleared by writing 1 on the configuration interface. A read access leaves the bit unchanged. **Note:** This bit is only set in SOMP mode.

15.5.18 ATOM[i]_CH[x]_RDADDR

Description	ATOM[i] channel[x] ARU read address register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ATOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_RDADDR
Address	$0x20000 * i + 0x80 * x + 0x1800$
C-Name	GTM.CLS[i].ATOM.CH[x].RDADDR

Interface: MCS[i]

Name	ATOM[i]_CH[x]_RDADDR
Address	$0x80 * x + 0x1800$
C-Name	

RDADDR0	
Description	ARU Read address 0.
Loop	-
Bit Range	$[8 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x1FE$
Protect Enable Cond	ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[x] == 0

RDADDR0	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
	Note: This read address is used by the ATOM channel to receive data from the ARU immediately after the channel and ARU access is enabled (see **ATOM[i]_CH[x]_CTRL** register for details). **Note:** This bit field is only writable if channel is disabled.

RDADDR1	
Description	ARU Read address 1.
Loop	–
Bit Range	$[24 : 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0x1FE$
Protect Enable Cond	ATOM[i]_AGC_ENDIS_STAT.ENDIS_STAT[x] == 0
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
	Note: This read address is only applicable in SOMC mode. In other modes, it is recommended to keep the bit field as the initial value. **Note:** The ATOM channel switches to this read address, when requested in the ARU control bits 52 to 48 with the pattern "111--". The channel switches back to the **ATOM[i]_CH[x]_RDADDR.RDADDR0** after one ARU data package was received on **ATOM[i]_CH[x]_RDADDR.RDADDR1** and the compare match event is occurred. **Note:** This bit field is only writable if channel is disabled.

15.5.19 ATOM[i]_CH[x]_CN0

Description	ATOM[i] channel [x] CCU0 counter register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_CN0
Address	$0x20000 * i + 0x80 * x + 0x1818$

C-Name	GTM.CLS[i].ATOM.CH[x].CN0

Interface: MCS[i]

Name	ATOM[i]_CH[x]_CN0
Address	$0x80 * x + 0x1818$
C-Name	

CN0	
Description	ATOM CCU0 counter register.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

15.5.20 ATOM[i]_CH[x]_CM0

Description	ATOM[i] channel [x] CCU0 compare register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_CM0
Address	$0x20000 * i + 0x80 * x + 0x1810$
C-Name	GTM.CLS[i].ATOM.CH[x].CM0

Interface: MCS[i]

Name	ATOM[i]_CH[x]_CM0
Address	$0x80 * x + 0x1810$
C-Name	

CM0	
Description	ATOM CCU0 compare register.
Loop	-
Bit Range	$[23 : 0]$

CM0	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

Note:
If the channel is configured in SOMC mode 'serve last' compare strategy with **ATOM[i]_CH[x]_CTRL.EUPM** =0, when the first compare event in CCU0 occurs, the register will be write protected and return *AEI_STATUS* =0b10 with write access until the second compare event in CCU1.

15.5.21 ATOM[i]_CH[x]_SR0

Description	ATOM[i] channel [x] CCU0 compare shadow register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_SR0
Address	$0x20000 * i + 0x80 * x + 0x1808$
C-Name	GTM.CLS[i].ATOM.CH[x].SR0

Interface: MCS[i]

Name	ATOM[i]_CH[x]_SR0
Address	$0x80 * x + 0x1808$
C-Name	

SR0	
Description	ATOM channel [x] shadow register SR0.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

SR0	
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
	Note: The **ATOM[i]_CH[x]_SR0.SR0** register is used as shadow register for **ATOM[i]_CH[x]_CM0.CM0** in SOMP and SOMS modes and is used as capture register for time base *CCM[i]_TBU_TS0* in SOMC mode.

Note:
When in SOMC mode a compare event occurs, this register will be write protected and return *AEI_STATUS* =0b10 with write access until the register has been read out via ARU or the configuration interface.

15.5.22 ATOM[i]_CH[x]_CM1

Description	ATOM[i] channel [x] CCU1 compare register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_CM1
Address	$0x20000 * i + 0x80 * x + 0x1814$
C-Name	GTM.CLS[i].ATOM.CH[x].CM1

Interface: MCS[i]

Name	ATOM[i]_CH[x]_CM1
Address	$0x80 * x + 0x1814$
C-Name	

CM1	
Description	ATOM CCU1 compare register.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync

Note:

If the channel is configured in SOMC mode 'serve last' compare strategy with **ATOM[i]_CH[x]_CTRL.EUPM** =0, when the first compare event in CCU0 occurs, the register will be write protected and return *AEI_STATUS* =0b10 with write access until the second compare event in CCU1.

15.5.23 ATOM[i]_CH[x]_SR1

Description	ATOM[i] channel [x] CCU1 compare shadow register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_SR1
Address	$0x20000 * i + 0x80 * x + 0x180C$
C-Name	GTM.CLS[i].ATOM.CH[x].SR1

Interface: MCS[i]

Name	ATOM[i]_CH[x]_SR1
Address	$0x80 * x + 0x180C$
C-Name	

SR1	
Description	ATOM channel [x] shadow register SR1.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
	Note: The **ATOM[i]_CH[x]_SR1.SR1** register is used as shadow register for **ATOM[i]_CH[x]_CM1.CM1** in SOMP and SOMS modes and is used as capture register for time base *CCM[i]_TBU_TS1* or *CCM[i]_TBU_TS2* (as selected in **ATOM[i]_CH[x]_CTRL** register) in SOMC mode. **Note:** When in SOMC mode a compare event occurs, this register will be write protected and return *AEI_STATUS* =0b10 with write access until the register has been read out via ARU or the configuration interface.

15.5.24 ATOM[i]_CH[x]_IRQ_NOTIFY

Description	ATOM[i] channel [x] interrupt notification register

Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	ATOM[i]_CLK_ENABLE $== 1$

Interface: CPU

Name	ATOM[i]_CH[x]_IRQ_NOTIFY
Address	$0x20000 * i + 0x80 * x + 0x1820$
C-Name	GTM.CLS[i].ATOM.CH[x].IRQ_NOTIFY

Interface: MCS[i]

Name	ATOM[i]_CH[x]_IRQ_NOTIFY
Address	$0x80 * x + 0x1820$
C-Name	

CCU0TC	
Description	CCU0 Trigger condition interrupt for channel [x].
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	0 : No interrupt triggered by software or by the compare event in CCU0 unit was raised. 1 : Interrupt triggered by software or by the compare event in CCU0 unit was raised.
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a configuration interface write access of value 1. A read access leaves the bit unchanged.

CCU1TC	
Description	CCU Trigger condition interrupt for channel [x].
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear

CCU1TC	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	0 : No interrupt triggered by software or by the compare event in CCU1 unit was raised. 1 : Interrupt triggered by software or by the compare event in CCU1 unit was raised.
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a configuration interface write access of value 1. A read access leaves the bit unchanged. **Note:** If bit **ATOM[i]_CH[x]_CTRL_SOMP.SR0_TRIG** is set to 1 (only valid in SOMP mode), this interrupt notify flag **ATOM[i]_CH[x]_IRQ_NOTIFY.CCU1TC** is set in case **ATOM[i]_CH[x]_SR0.SR0** is equal to **ATOM[i]_CH[x]_CN0.CN0** and not set in case **ATOM[i]_CH[x]_CM1.CM1** is equal to **ATOM[i]_CH[x]_CN0.CN0** .

15.5.25 ATOM[i]_CH[x]_IRQ_EN

Description	ATOM[i] channel [x] interrupt enable register
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ATOM}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_IRQ_EN
Address	$0x20000 * i + 0x80 * x + 0x1824$
C-Name	GTM.CLS[i].ATOM.CH[x].IRQ_EN

Interface: MCS[i]

Name	ATOM[i]_CH[x]_IRQ_EN
Address	$0x80 * x + 0x1824$
C-Name	

CCU0TC_IRQ_EN	
Description	ATOM_CCU0TC[x]_IRQ interrupt enable.
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

CCU0TC_IRQ_EN	
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : the interrupt signal *ATOM_CCU0TC[x]_IRQ* is not visible outside GTM-IP. 1 : the interrupt signal *ATOM_CCU0TC[x]_IRQ* is visible outside GTM-IP.

CCU1TC_IRQ_EN	
Description	ATOM_CCU1TC[x]_IRQ interrupt enable.
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0 : the interrupt signal *ATOM_CCU1TC[x]_IRQ* is not visible outside GTM-IP. 1 : the interrupt signal *ATOM_CCU1TC[x]_IRQ* is visible outside GTM-IP.

15.5.26 ATOM[i]_CH[x]_IRQ_FORCINT

Description	ATOM[i] channel [x] software interrupt generation
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_IRQ_FORCINT
Address	$0x20000 * i + 0x80 * x + 0x1828$
C-Name	GTM.CLS[i].ATOM.CH[x].IRQ_FORCINT

Interface: MCS[i]

Name	ATOM[i]_CH[x]_IRQ_FORCINT
Address	$0x80 * x + 0x1828$
C-Name	

TRG_CCU0TC	
Description	Trigger the bit ATOM[i]_CH[x]_IRQ_NOTIFY.CCU0TC by software.
Loop	-
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 1, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **ATOM[i]_CH[x]_IRQ_NOTIFY.CCU0TC** = 1
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

TRG_CCU1TC	
Description	Trigger the bit ATOM[i]_CH[x]_IRQ_NOTIFY.CCU1TC by software.
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 1, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **ATOM[i]_CH[x]_IRQ_NOTIFY.CCU1TC** = 1
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by register bit **GTM_CTRL.RF_PROT** .

15.5.27 ATOM[i]_CH[x]_IRQ_MODE

Description	ATOM[i] channel [x] interrupt mode configuration register
Loop	$i = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{ATOM}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	ATOM[i]_CH[x]_IRQ_MODE
Address	$0x20000 * i + 0x80 * x + 0x182C$
C-Name	GTM.CLS[i].ATOM.CH[x].IRQ_MODE

Interface: MCS[i]

Name	ATOM[i]_CH[x]_IRQ_MODE
Address	$0x80 * x + 0x182C$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	-
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async ATOM[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

15.6 ATOM Signal Description

15.6.1 ATOM reset

Loop	$j = \{n : 0 <= n <= \text{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	-

Format	–

ATOM[j]_RESET_CH[x]	
Description	ATOM instance [j] channel [x] reset active signal
Loop	–
Condition	–
Signal Type	ARRAY[NATOM][8]
Assignment	ATOM[j]_AGC_GLB_CTRL.RST_CH[x] == 1

15.6.2 ATOM Interrupt Signals

Loop	–
Condition	–
Format	–

ATOM_CCU0TC[x]_IRQ	
Description	CCU0 Trigger condition interrupt for channel x. Internal interrupt signal when compare event occurs in CCU0 unit.
Loop	$x = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]
Assignment	–

ATOM_CCU1TC[x]_IRQ	
Description	CCU1 Trigger condition interrupt for channel x. Internal interrupt signal when compare event occurs in CCU0 unit.
Loop	$x = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]
Assignment	–

15.6.3 ATOM Signal Group

Loop	–
Condition	–
Format	–

CAP	
Description	signaling a compare match event for capture
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_ENDIS	
Description	A bundle of channel enable signals controlled by AGC unit
Loop	–
Condition	–

ATOM_ENDIS	
Signal Type	INT
Assignment	–

ATOM_CTRL_TRIG	
Description	trigger signal to update the AGC registers ATOM[i]_agc_OUTEN_STAT, ATOM[i]_AGC_ENDIS_STAT and to activate ATOM_FUPD signals
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EXT_TRIGIN_S	
Description	synchronized EXT_TRIGIN signal to selected CMU clock resolution
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_FUPD	
Description	A bundle of channel force update event signals controlled by AGC unit
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_RSTCN0_CH	
Description	A bundle of channel force update event signals controlled AGC unit
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_OUTEN	
Description	output enable signal from AGC by unit
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_TRIG_CCU0	
Description	trigger signal for CCU0 compare match
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_TRIG_CCU1	
Description	trigger signal for CCU1 compare match

ATOM_TRIG_CCU1	
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_UPEN	
Description	A bundle of channel update enable signals controlled by AGC unit
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SEL_CMU_CLKEN	
Description	The selected CMU clock enable signal
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_ARU_VALID	
Description	Valid signal that reflects that the ARU is enabled and the first ARU data is transferred.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_EXT_HRES	
Description	Not internal signal from GTM or ATOM. A signal to explain how the external circuit can generate a signal for each channel x that consists of edges from ATOM_OUT[x:x] with delay of ATOM_OUT_HRES[x].
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_CH_TRIGIN	
Description	Trigger input signal for each channel x. It comes from the preceding channel x-1 or channel 7 of the preceding instance.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ATOM_CH_TRIGOUT	
Description	Trigger output signal for each channel x. It is connected to the trigger input signal of the next channel x+1 or channel 0 of the succeeding instance.
Loop	–
Condition	–
Signal Type	INT

ATOM_CH_TRIGOUT	
Assignment	–

ATOM_CH_HRES_IN[x]	
Description	HRES trigger input signal for each channel x. It is 5 bits value. Together with ATOM_CH_TRIGIN[x:x], it comes from the preceding channel x-1 or channel 7 of the preceding instance.
Loop	$x = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]
Assignment	–

ATOM_CH_HRES_OUT[x]	
Description	HRES trigout signal for each channel x. It is 5 bits value. Together with ATOM_CH_TRIGOUT[x:x], it is connected with the HRES trigger input signal of the next channel x+1 or channel 0 of the next instance.
Loop	$x = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]
Assignment	–

ATOM[i]_CH[x]_HRES_PRIOD	
Description	HRES value of ATOM i channel x that is calculated with regards to ATOM[i]_CH[x]_CM0.CM0.
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[NATOM][8]
Assignment	–

ATOM[i]_CH[x]_HRES_PULSEWIDTH	
Description	HRES value of ATOM i channel x that is calculated with regards to ATOM[i]_CH[x]_CM1.CM1.
Loop	$i = \{n : 0 <= n <= \mathrm{NATOM} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[NATOM][8]
Assignment	–

15.7 ATOM Port Description

15.7.1 ATOM signal interface

Loop	–
Condition	–
Format	–

EXT_TRIGIN	
Description	EXT_TRIGIN[7:0] of all channels of ATOM instance i are coming from TIM_EXT_CAPTURE[7:0] signal from TIM instance i
Loop	–
Condition	–

EXT_TRIGIN	
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

ATOM_TRIGIN	
Description	ATOM trigger chain input of ATOM instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

ATOM_TRIGOUT	
Description	ATOM trigger chain output of ATOM instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

ATOM_TRIGOUT_DEL	
Description	ATOM trigger chain output delayed by 1 cluster clock; will be used only in ATOM [0] instance as ATOM_TRIGIN
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET

ATOM_TRIGOUT_DEL	
Reset Value	0

ATOM_OUT	
Description	Output signal ATOM_OUT
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

ATOM_OUT_T	
Description	Output signal ATOM_OUT_T
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

15.7.2 ATOM interrupt interface

Loop	–
Condition	–
Format	–

ATOM_IRQ	
Description	ATOM irq line signals
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET

ATOM_IRQ	
Reset Value	0

ATOM_IRQ_CLR	
Description	ATOM irq clear signal, hardware clear of ATOM[i]_CH[x]_IRQ_NOTIFY register
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

ATOM_IRQ_OCC	
Description	ATOM irq occurred signal, corresponds to ATOM[i]_CH[x]_IRQ_NOTIFY register
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

15.7.3 ATOM ARU interface

Loop	–
Condition	–
Format	–

ARU_CADDR	
Description	ARU master port active channel
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	aru_cwidth_c DOWNTO 0
Operational Clock	ARU_CLK
Special Function	–
Operational Reset	GTM_RESET

ARU_CADDR	
Reset Value	0

ARUM_RDATA	
Description	ARU master port read data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	aru_dwidth_c DOWNTO 0
Operational Clock	ARU_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

ARU_WDATA	
Description	Data which will be transferred over ARU.
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	0

15.7.4 ATOM Signal HRES interface

Loop	–
Condition	–
Format	–

ATOM_OUT_HRES[x]	
Description	ATOM_OUT high resolution output signal for channel x
Loop	$x = \{n : 0 <= n <= 7\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET

ATOM_OUT_HRES[x]	
Reset Value	0

ATOM_OUT_T_HRES[x]	
Description	ATOM_OUT_T high resolution output signal channel x
Loop	$x = \{n : 0 <= n <= 7\}$
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	-
Operational Reset	GTM_RESET
Reset Value	0

ATOM_HRES_IN	
Description	ATOM high resolution trigger chain input; will be driven by previous ATOM instance i-1
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	-
Operational Reset	GTM_RESET
Reset Value	0

ATOM_HRES_OUT	
Description	ATOM high resolution trigger chain output;will be used by following ATOM instance i+1
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	-
Operational Reset	GTM_RESET
Reset Value	0

ATOM_HRES_OUT_DEL	
Description	ATOM high resolution trigger chain output delayed by 1 cluster clock;will be used only in ATOM instance 0 as ATOM_HRES_IN
Loop	-

ATOM_HRES_OUT_DEL	
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	-
Operational Reset	GTM_RESET
Reset Value	0

16 Dead Time Module (DTM)

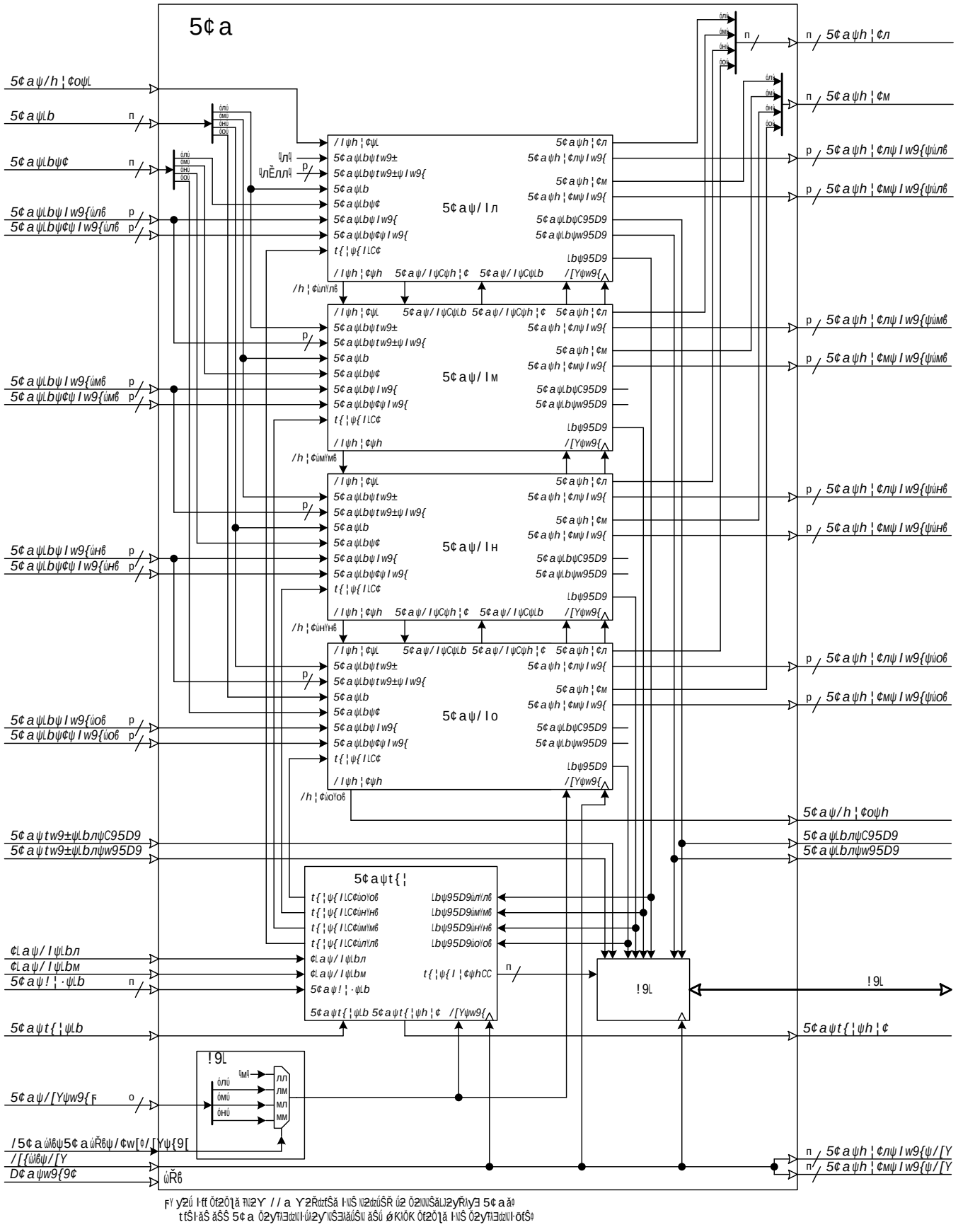
16.1 Overview

The following figure gives an overview of the structure of the Dead Time Module (DTM).

Register name indices used within this module:

- i:={0, 1,..., 11} instance index of cluster
- j:={0, 1,..., 11} instance index in case this index refers to a cluster/module with j!=i
- d:={0, 1,..., 11} index for a dedicated DTM ((d∈{0, 1, 2, 3}): DTM after TOM; (d∈{4, 5}): DTM after ATOM; (d∈{6, 7,..., 11}): DTM after TIO)
- x:={0, 1,..., 3} channel index in a DTM instance
- y:={0, 1,..., N−1} with N:={NTOM, NATOM, NTIO} channel index of another module e.g. TOM, ATOM, TIO
- k:={0, 1} channel index with a special meaning e.g. limited value subset

Figure 115 DTM Overview



The DTM module can enhance the functions of modules TOM, ATOM and TIO by modifying their output signals.

In general, the DTM instances are placed behind TOM, ATOM and TIO instances, i.e., the outputs *TOM_OUT* [y:y] and *TOM_OUT_T* [y:y], *ATOM_OUT* [y:y] and *ATOM_OUT_T* [y:y], and *TIO_G[g]_OUT* [y:y] and *TIO_G[g]_S_OUT* [y:y] are each routed to the DTM instance inputs

DTM_IN and *DTM_IN_T* .
Four DTM instances behind a TOM instance i, two DTM instances behind an ATOM instance i and six DTM instances behind a TIO instance are grouped together in a Cluster DTM hierarchy.
The connections between DTM and the modules TOM, ATOM and TIO are depicted in Figure *116 "Connections of TOM, ATOM and TIO to DTM Inputs DTM_IN[x:x]/DTM_IN_T[x:x]"* .

Attention:
The initial configuration of a DTM channel or any asynchronous channel configuration to the running application is only allowed, if the corresponding TOM, ATOM and TIO channel is disabled. A configuration of DTM channel realized from the shadow registers synchronous to the running application can be executed without disabling the interfaced TOM, ATOM and TIO channels.

The main function of the DTM is to derive the individual inverse signal (*DTM_OUT1* [3:0]) for each input *DTM_IN* [3:0] and to apply an edge specific delay between the edge of the original signal and the derived inverted signal (i.e., the dead time). This function is mainly used for controlling of half bridges.

A second function provided by DTM is to set the outputs of one channel to the value of the preceding channel if requested by a trigger on input *TIM_CH_IN0* , *TIM_CH_IN1* or *DTM_AUX_IN* [0:0]. This feature allows a phase shift on one PWM signal to the phase of the preceding PWM signal up to the next edge on this channel.

The third function provided by DTM is to (N)AND/(N)OR/X(N)OR combine the input *DTM_IN* [x:x] signal of one DTM channel with the signal on inputs *TIM_CH_IN0* , *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] (selected inside PSU and assigned to one of the signals *PSU_SHIFT* [x:x]) or with the combinational output (signal *COUT* [x−1:x−1]) of the preceding channel.
As a result, *COUT* [1:1] will be the combined signal of *DTM_IN* [0:0] and *TIM_CH_IN0* , *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] and the signal *DTM_IN* [1:1]. For *COUT* [2:2], this chain can be combined again with signal *DTM_IN* [2:2].
For *COUT* [3:3], this chain can be combined again with signal *DTM_IN* [3:3].

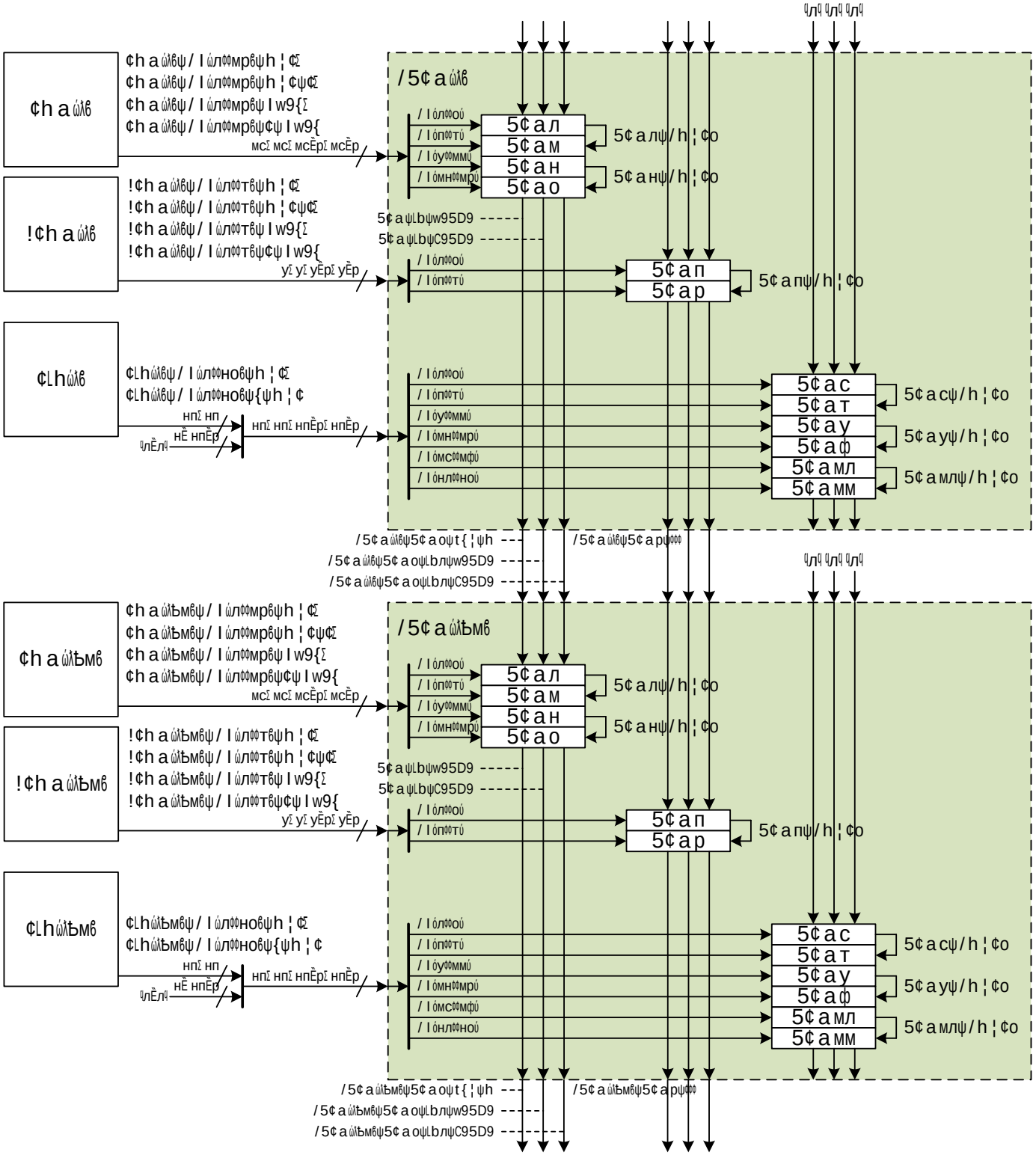
- The outputs of each channel may be swapped individually to provide the function of combining signals on each output of a channel.
- For channel 0 the function is controlled by **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** .
- For channel 1 the function is controlled by **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** .
- For channel 2 the function is controlled by **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** .
- For channel 3 the function is controlled by **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** .

The 4th function is the high resolution PWM support. For each of the two input signals of a DTM channel, a high resolution value is given by a corresponding input signal. Here, a configurable value is added to the high resolution input value.

The high resolution mode in DTM is only supported for standard dead time mode and for phase shift mode, where standard dead time is used.

Depending on the device configuration, not every DTM instance is available. E.g. a device may only have one DTM connected to the first four channels of ATOM. In this case, the other four channels (channel 4 to channel 7) are connected over a buffer stage to GTM outputs.
For detailed information, which DTM instance is available, refer to the corresponding device specific documentations.

Figure 116 Connections of TOM, ATOM and TIO to DTM Inputs DTM_IN[x:x]/DTM_IN_T[x:x]



Additionally, the DTM instances have inputs *TIM_CH_IN0* , *TIM_CH_IN1* which are driven by TIM output signals *F_OUT* of instances i and channel y.
Two configurations of TIM to DTM connections are possible depending on the DTM channel specific configuration bit **CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL** .
In case of **CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL** =0, the connected TIM input may not be of the same cluster as the DTM.
In case of **CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL** =1, the TIM input is of the same cluster as the DTM.

Figure 117 Connections of TIM to DTM Inputs TIM_CH_IN0/TIM_CH_IN1 for CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL=0

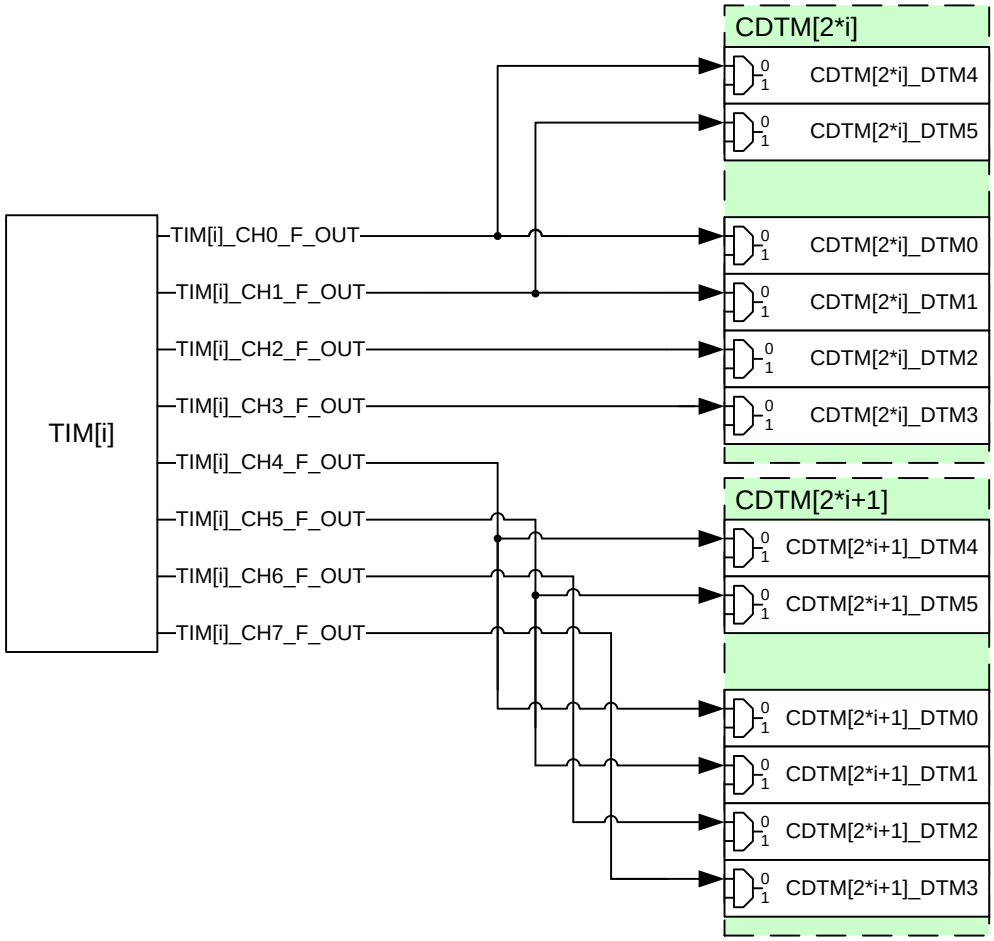
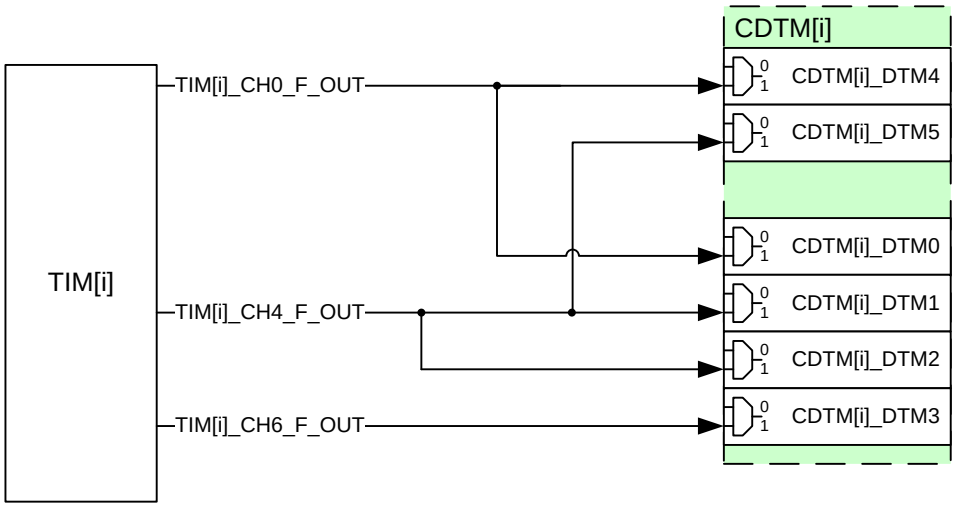


Figure 118 Connections of TIM to DTM Inputs TIM_CH_IN0/TIM_CH_IN1 for CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL=1



Note:

The multiplexer shown in Figures *117 "Connections of TIM to DTM Inputs TIM_CH_IN0/TIM_CH_IN1 for CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL=0"* and *118 "Connections of TIM to DTM Inputs TIM_CH_IN0/TIM_CH_IN1 for CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL=1"* and driven by **CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL** is located in the block DTM_PSU.

There are also connections between DTM instances inside one cluster.
For each pair of DTM instance 2d and instance 2d+1 the combinatorial output *COUT* [3:3] (*COUT* [3:3] routed to *DTM_COUT3_O*) of DTM instance 2d channel 3 is connected to DTM instance 2d+1 channel 0 at signal *DTM_COUT3_I* .
With this, a combinatorial chain over two neighbored DTM instances can be configured.
If one of the two neighbored DTM instances is not available (i.e. it is an empty instance) the inputs used for connections between two neighbored DTM instances are hard wired to 0.
These connections are shown in Figure *115 "DTM Overview"* and Figure *116 "Connections of TOM, ATOM and TIO to DTM Inputs DTM_IN[x:x]/DTM_IN_T[x:x]"* .

- For channel 0 of DTM instance 2d input signal *COUT* [x−1:x−1] is unused and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_0** is defined as 0.
- For channel 0 of DTM instance 2d input signal *COUT* [x−1:x−1] is unused and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_1** is defined as 0.
- For channel 0 of DTM instance 2d input signal *COUT* [x−1:x−1] is unused and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_2** is defined as 0.
- For channel 0 of DTM instance 2d input signal *COUT* [x−1:x−1] is unused and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_3** is defined as 0.

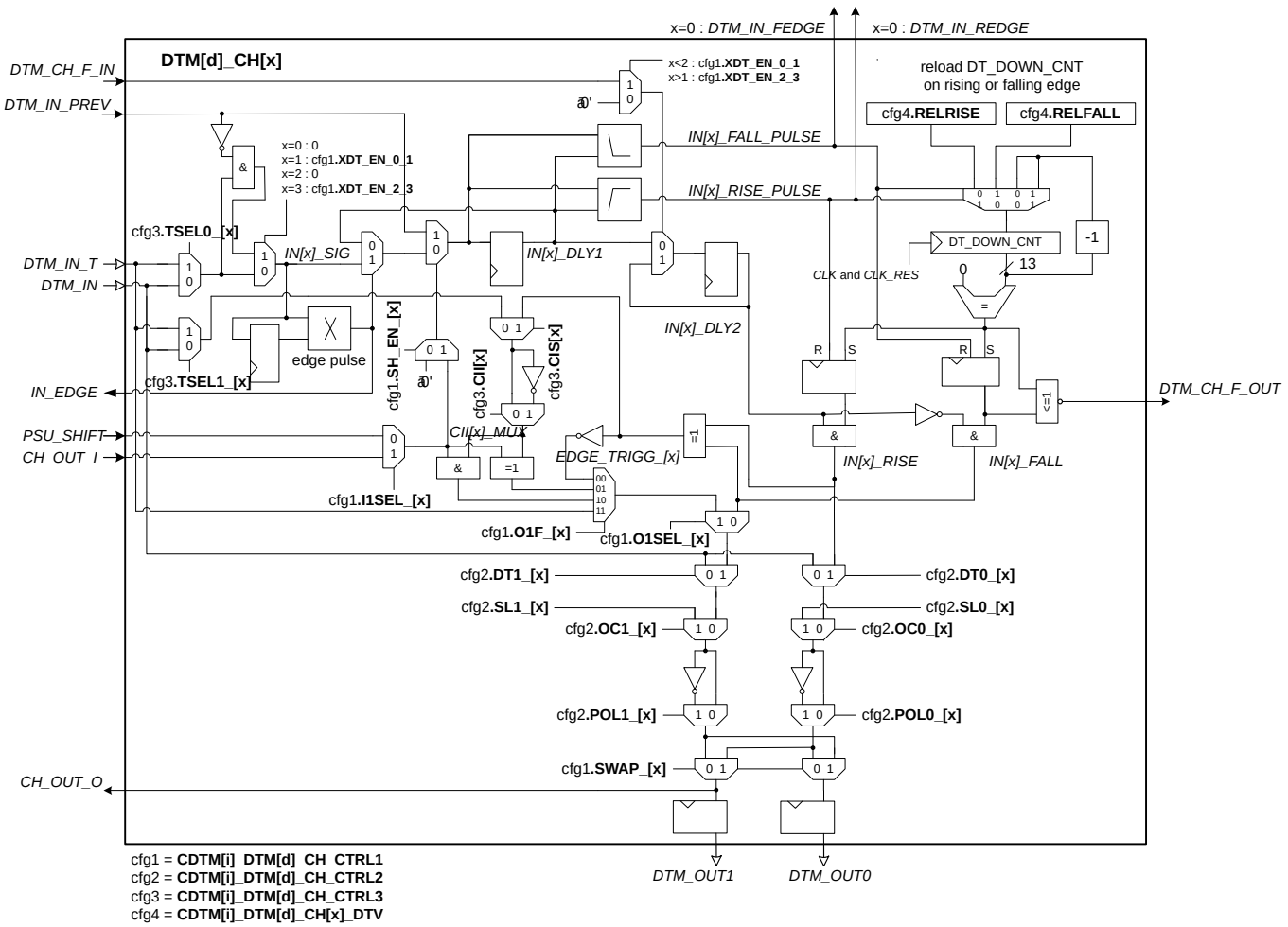
An additional link between DTM instances behind an ATOM is a forwarding of output *DTM_PSU_OUT* of DTM instance d to the input *DTM_PSU_IN* of next available instance d+1 behind an ATOM.
The same link is available between all available DTM instances behind a TOM and a TIO.

For unavailable DTM[d] instances (i.e. the instance DTM[d] is called empty) the signal *DTM_PSU_OUT* of DTM instance d−1 is passed through empty instance DTM[d] to *DTM_PSU_IN* of DTM instance d+1, *DTM_IN0_FEDGE* and *DTM_IN0_REDGE* of DTM instance d−1 are passed through DTM[d] to *DTM_PREV_IN0_FEDGE* and *DTM_PREV_IN0_REDGE* of DTM instance d+1.

16.2 DTM Channel

The following figure depicts the circuit of a DTM channel.

Figure 119 DTM Channel Overview



The main feature of each channel is to derive the inverse signal out of the input signal *DTM_IN* [x:x], apply an edge dependent delay to the two resulting signal paths and provide these signals at the outputs *DTM_OUT0* [x:x] and *DTM_OUT1* [x:x].

There are two possibilities to apply dead time on GTM output signals.
One is to use one DTM channel per TOM/ATOM/TIO channel and generate inside the DTM the second inverse signal. This is called the

standard dead time generation.
The second way is to generate two signals out of two TOM/ATOM/TIO channel and to apply inside the DTM only the dead time by using two cross linked DTM channels. This is called the cross dead time generation.

16.2.1 Standard Dead Time Generation

The dead time can be configured for each edge individually. The bit field **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** contains the reload value for the counter and defines the delay for rising edges in multiples of selected clock ticks.
The bit field **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** contains the reload value for the counter and defines the delay for falling edges in multiples of selected clock ticks.

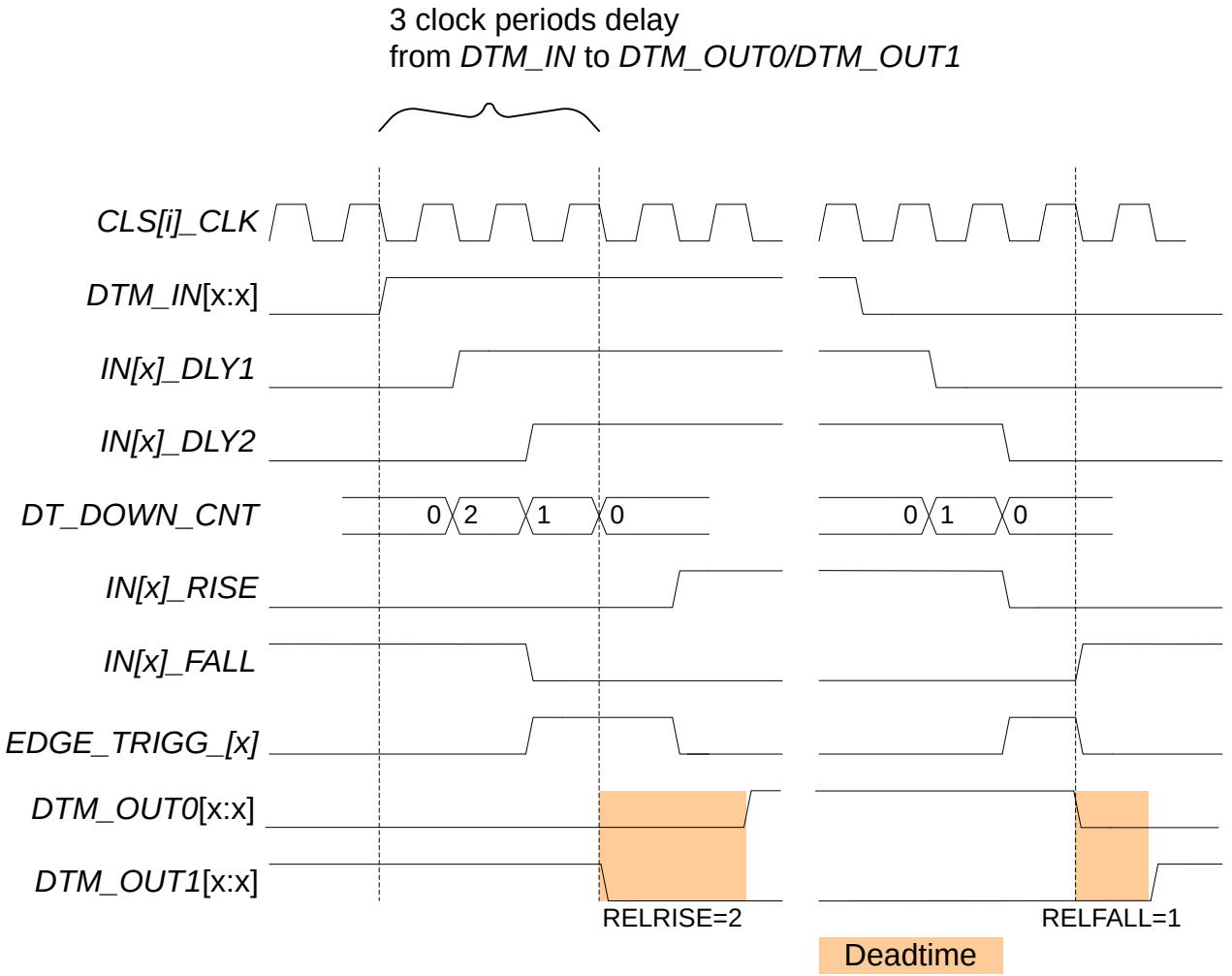
Note:

If the selected clock ticks are modified by writing **CDTM[i]_DTM[d]_CTRL.CLK_SEL** during a running application, the configured times (e.g. DT_DOWN_CNT) may differ, because the counters can run once partly with the old and the new clock ticks. It is recommended not to reprogram **CDTM[i]_DTM[d]_CTRL.CLK_SEL** during operation.

The counter is reloaded with the value of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** on a rising edge and reloaded with the value of **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** on a falling edge on input *DTM_IN* [x:x] (or *DTM_IN_PREV* in case of shift enable **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_1** , **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_2** , **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_3**).(x=1)

The signal flow for function of standard dead time signal generation is depicted in Figure *120 "Wave Signals for Function of Dead Time Generation"* .

Figure 120 Wave Signals for Function of Dead Time Generation



Note:

The figure above shows the timing for **CDTM[i]_DTM[d]_CTRL.CLK_SEL** =0b00. If another clock resolution is configured, be aware that only the counter DT_DOWN_CNT runs with the reduced clock resolution.

- The delay from the input signal *DTM_IN* [0:0] to the output signals *DTM_OUT0* [0:0] and *DTM_OUT1* [0:0] is three system clock periods, if **CDTM[i]_DTM[d]_CH_CTRL2.DT0_0** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_0** =1.
- The delay from the input signal *DTM_IN* [1:1] to the output signals *DTM_OUT0* [1:1] and *DTM_OUT1* [1:1] is three system clock periods, if **CDTM[i]_DTM[d]_CH_CTRL2.DT0_1** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_1** =1.

- The delay from the input signal *DTM_IN* [2:2] to the output signals *DTM_OUT0* [2:2] and *DTM_OUT1* [2:2] is three system clock periods, if **CDTM[i]_DTM[d]_CH_CTRL2.DT0_2** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_2** =1.
- The delay from the input signal *DTM_IN* [3:3] to the output signals *DTM_OUT0* [3:3] and *DTM_OUT1* [3:3] is three system clock periods, if **CDTM[i]_DTM[d]_CH_CTRL2.DT0_3** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_3** =1.

- The delay from the input signal *DTM_IN* [0:0] to the output signals *DTM_OUT0* [0:0] and *DTM_OUT1* [0:0] is one system clock period if **CDTM[i]_DTM[d]_CH_CTRL2.DT0_0** =0 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_0** =0.
- The delay from the input signal *DTM_IN* [1:1] to the output signals *DTM_OUT0* [1:1] and *DTM_OUT1* [1:1] is one system clock period if **CDTM[i]_DTM[d]_CH_CTRL2.DT0_1** =0 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_1** =0.
- The delay from the input signal *DTM_IN* [2:2] to the output signals *DTM_OUT0* [2:2] and *DTM_OUT1* [2:2] is one system clock period if **CDTM[i]_DTM[d]_CH_CTRL2.DT0_2** =0 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_2** =0.
- The delay from the input signal *DTM_IN* [3:3] to the output signals *DTM_OUT0* [3:3] and *DTM_OUT1* [3:3] is one system clock period if **CDTM[i]_DTM[d]_CH_CTRL2.DT0_3** =0 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_3** =0.

- The delay from the input signal *DTM_IN_T* [0:0] to the output signals *DTM_OUT0* [0:0] and *DTM_OUT1* [0:0] is three system clock periods, in case **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** =0.
- The delay from the input signal *DTM_IN_T* [1:1] to the output signals *DTM_OUT0* [1:1] and *DTM_OUT1* [1:1] is three system clock periods, in case **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_1** =0.
- The delay from the input signal *DTM_IN_T* [2:2] to the output signals *DTM_OUT0* [2:2] and *DTM_OUT1* [2:2] is three system clock periods, in case **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_2** =0.
- The delay from the input signal *DTM_IN_T* [3:3] to the output signals *DTM_OUT0* [3:3] and *DTM_OUT1* [3:3] is three system clock periods, in case **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_3** =0.

- The delay from the input signal *DTM_IN_T* [0:0] to the output signals *DTM_OUT0* [0:0] and *DTM_OUT1* [0:0] is one system clock period in case of **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** =0b11 and **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** =1.
- The delay from the input signal *DTM_IN_T* [1:1] to the output signals *DTM_OUT0* [1:1] and *DTM_OUT1* [1:1] is one system clock period in case of **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** =0b11 and **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_1** =1.
- The delay from the input signal *DTM_IN_T* [2:2] to the output signals *DTM_OUT0* [2:2] and *DTM_OUT1* [2:2] is one system clock period in case of **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** =0b11 and **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_2** =1.
- The delay from the input signal *DTM_IN_T* [3:3] to the output signals *DTM_OUT0* [3:3] and *DTM_OUT1* [3:3] is one system clock period in case of **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** =0b11 and **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_3** =1.

For **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** =1 and **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** !=0b11 there are further configurations possible which result in different paths from the selected input signal to the output signals causing different delays of 1 or 3 clock periods. For details see Figure *119 "DTM Channel Overview"* .

The reset level of the output signals *DTM_OUT0* [x:x] connected from ATOM module depends on the hardware configuration value ATOM_OUT_RST chosen by the silicon vendor.

The reset level of the output signals *DTM_OUT1* [x:x] connected from ATOM module is defined by the inverse hardware configuration value ATOM_OUT_RST chosen by the silicon vendor.

The reset level of the output signals *DTM_OUT0* [x:x] connected from TOM module is defined by the hardware configuration value TOM_OUT_RST chosen by the silicon vendor.

The reset level of the output signals *DTM_OUT1* [x:x] connected from TOM module is defined by the inverse hardware configuration value TOM_OUT_RST chosen by the silicon vendor.

The reset level of the output signals *DTM_OUT0* [x:x] connected from TIO module is defined by the hardware configuration value TIO_OUT_RST chosen by the silicon vendor.

The reset level of the output signals *DTM_OUT1* [x:x] connected from TIO module is defined by the inverse hardware configuration value TIO_OUT_RST chosen by the silicon vendor.

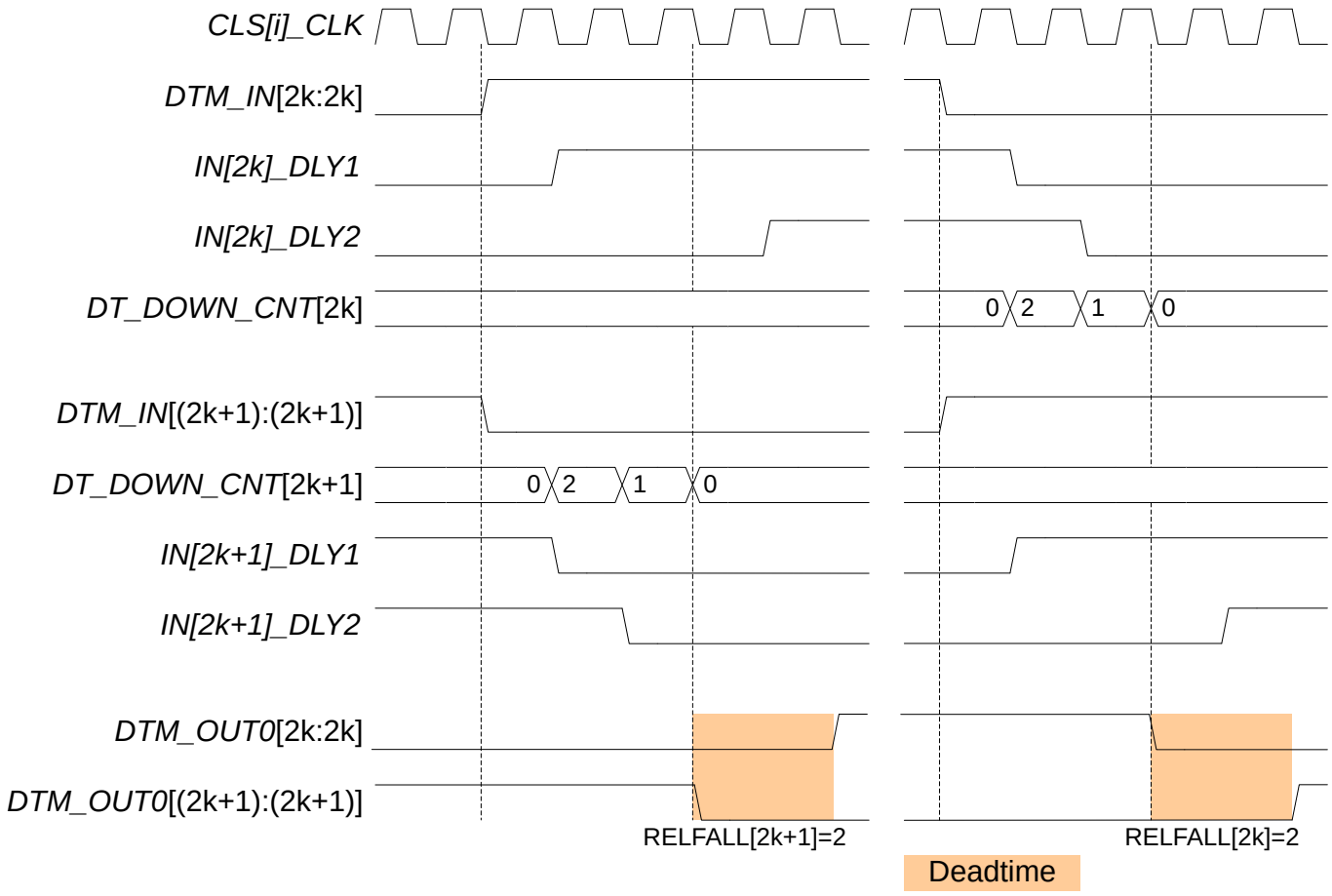
16.2.2 Cross Channel Dead Time

A second way to apply a dead time value on two output signals is the cross channel dead time.
In opposite to the dead time described in chapter *16.2.1 "Standard Dead Time Generation"* the cross channel dead time mode generates a dead time on the input signals of two neighbored DTM channels.
To do this, two neighbored DTM input signals (on DTM channel (2k) and (2k+1) for (k∈{0, 1})) are cross linked together in the way that a falling edge on one channel leads to a hold phase of current signal value on the cross linked channel.

This behavior is reached by the following:
A falling edge on input signal of DTM channel 2k reloads the counter DT_DOWN_CNT with the value of **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** . While this counter is counting down, the output signal of the cross linked channel (2k+1) keeps its value. If the counter DT_DOWN_CNT has reached 0 again (*DT_DOWN_CNT* =0), the channel (2k+1) output is released and can follow the value on its input.
The timing of the cross channel dead time is depicted in Figure *121 "Cross Channel Dead Time Timing Diagram"* .

The following Figure *121 "Cross Channel Dead Time Timing Diagram"* shows the behavior in case of input edges at *DTM_IN* [2k:2k] and *DTM_IN* [2k+1:2k+1] occur at the same point in time.
Then the falling edge is forwarded immediately (with only three clock cycles delay) and the rising edge is delayed additionally by the number of clock ticks specified by the **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** parameter of the cross linked channel.

Figure 121 Cross Channel Dead Time Timing Diagram



Note:

The figure above shows the timing for **CDTM[i]_DTM[d]_CTRL.CLK_SEL** =0b00. If another clock resolution is configured, be aware that only the counters DT_DOWN_CNT run with the reduced clock resolution.

In case of high level (i.e. 1) at the DTM inputs *DTM_IN* [2k:2k] and *DTM_IN* [(2k+1):(2k+1)] at the same point in time, the channel of (2k) has higher priority than the corresponding channel (2k+1). This means that in this case the input *DTM_IN* [(2k+1):(2k+1)] is forced immediately at channel input to low level (i.e. 0).
As a result the DTM output of channel *DTM_OUT0* [(2k+1):(2k+1)] can never be high if the cross linked channel *DTM_OUT0* [2k:2k] is high.

16.2.3 Dead Time Shadow Register

The values **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** and **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** define the dead time as described in the prior chapters. To be able to change these values synchronized to a generated waveform an additional shadow register **CDTM[i]_DTM[d]_CH[x]_DTV_SR** is implemented. The register is accessible over the configuration interface.

The register **CDTM[i]_DTM[d]_CH[x]_DTV_SR** includes the bit fields **CDTM[i]_DTM[d]_CH[x]_DTV_SR.RELRISE_SR** , **CDTM[i]_DTM[d]_CH[x]_DTV_SR.RELFALL_SR** which are the shadow bit fields of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** and **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** and further four bits to control the update mechanism.

CDTM[i]_DTM[d]_CH[x]_DTV_SR.RELRISE_SR is the shadow value for **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** . **CDTM[i]_DTM[d]_CH[x]_DTV_SR.RELFALL_SR** is the shadow value for **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** .

The update mechanism of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** is enabled if **CDTM[i]_DTM[d]_CH[x]_DTV_SR.RELRISE_UPD_EN** =1 is set.

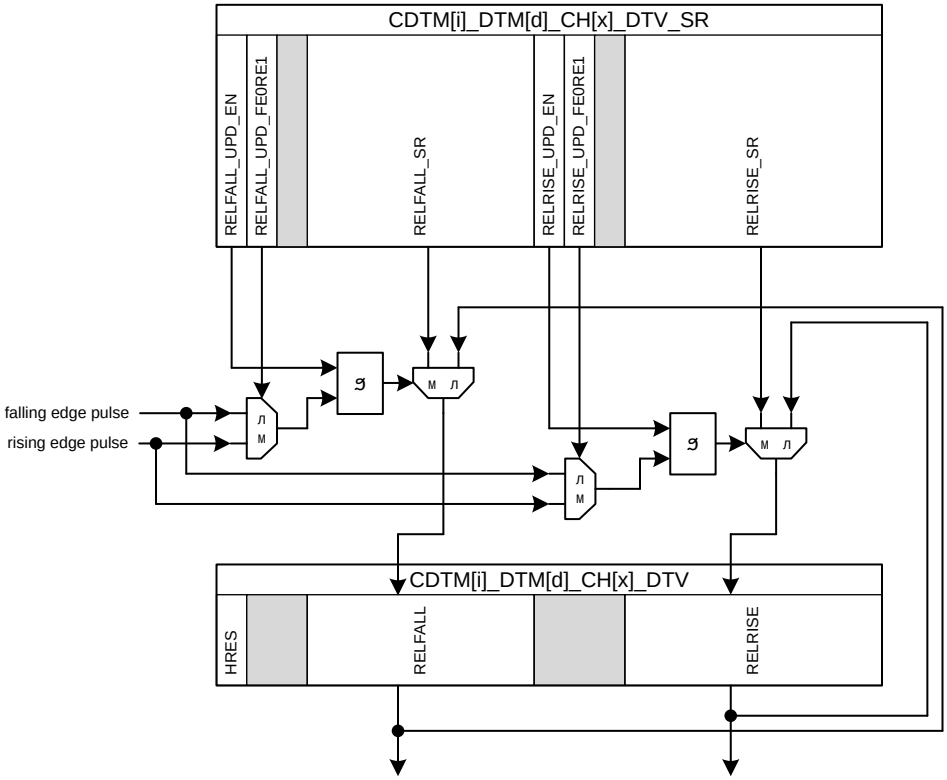
The update mechanism of **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** is enabled if **CDTM[i]_DTM[d]_CH[x]_DTV_SR.RELFALL_UPD_EN** =1 is set.

Bit **CDTM[i]_DTM[d]_CH[x]_DTV_SR.RELRISE_UPD_FE0RE1** controls if the falling edge (=0) or the rising edge (=1) of *DTM_IN* [x:x] triggers the update of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** .

Bit **CDTM[i]_DTM[d]_CH[x]_DTV_SR.RELFALL_UPD_FE0RE1** controls if the falling edge (=0) or the rising edge (=1) of *DTM_IN* [x:x] triggers the update of **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** .

Figure *122 "Dead Time Shadow Register"* shows the extension to Figure *119 "DTM Channel Overview"* .

Figure 122 Dead Time Shadow Register



16.3 Phase Shift Unit – Overcurrent Shutoff Feature for Full Bridge Push–Pull Converters

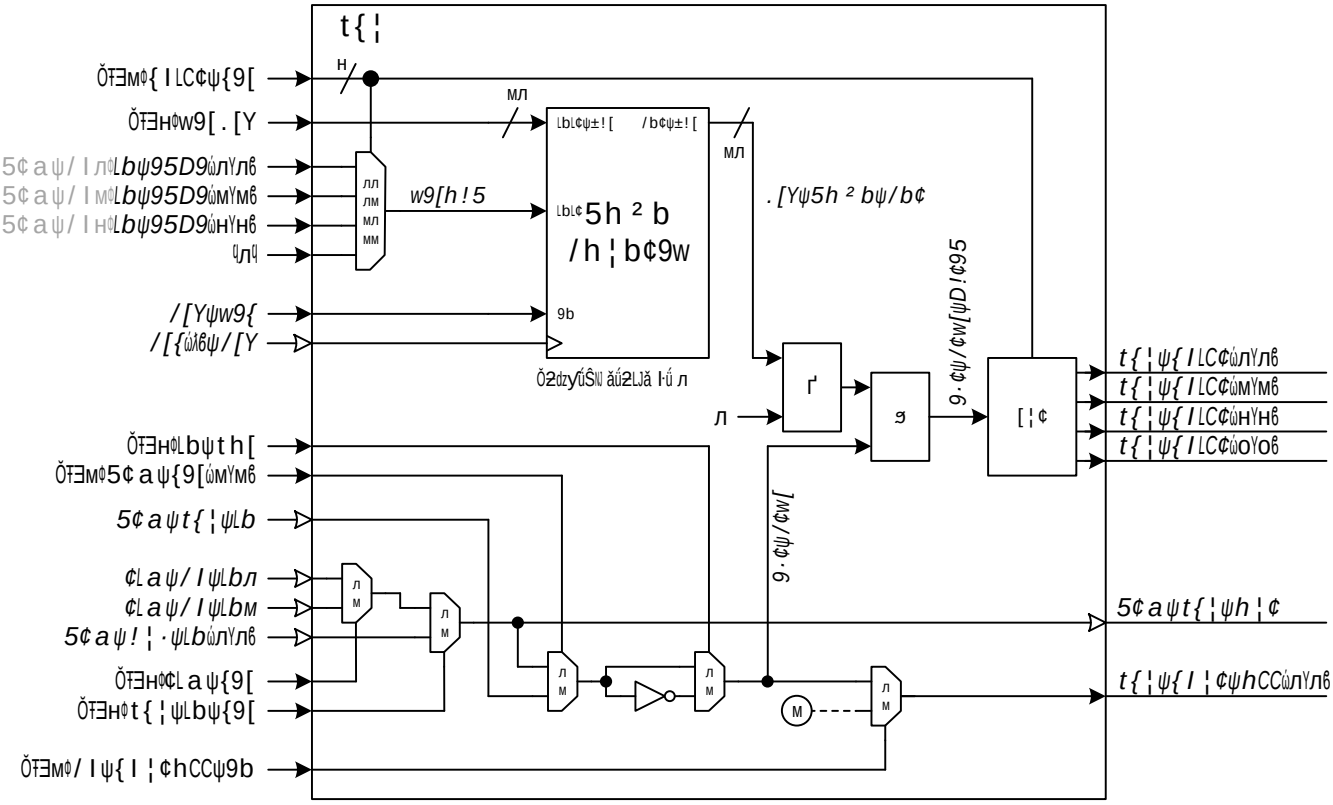
The Phase Shift Unit (PSU) is depicted in the following figure. It supports the second major function of the DTM module to copy a PWM signal from a preceding channel to the current channel beginning with the next edge on the preceding channel.
The feature can be used for applications like the phase–shift control of a full bridge push–pull converter. For such an application two consecutive (A)TOM or TIO channels generate a PWM signal with 50 percent duty cycle. The phase shift between the two PWM signals defines how long two of the four full bridge semiconductors are active and a current flows. An over current detection can trigger the phase shift unit to copy the PWM signal of one DTM channel to the other and the semiconductors are switched off directly.

During the development of the function, the following assumptions were made:

- Two consecutive DTM channels [x] and [x–1] get e.g. from two (A)TOM or TIO channels a PWM signal.
- The PWM signals have a phase shift of 180 degrees to each other.
- The configured blanking window value is greater than or equal to the configured dead time value.
- The configured dead time is much smaller than the high phase of the PWM signal.
- It was never planned to use this feature in combination with the cross channel dead time. This is forbidden, because the behavior is undefined.

Figure *123 "Phase Shift Unit"* shows the corresponding hardware block. The block realizes the blanking window and generates the signal for the DTM channel which controls the switch to copy a PWM signal from another DTM channel.

Figure 123 Phase Shift Unit



LUT

cfg1.**SHIFT_SEL**	*PSU_SHIFT*[0:0]	*PSU_SHIFT*[1:1]	*PSU_SHIFT*[2:2]	*PSU_SHIFT*[3:3]
00	0	*EXT_CTRL_GATED*	0	0
01	0	0	*EXT_CTRL_GATED*	0
10	0	0	0	*EXT_CTRL_GATED*
11	*EXT_CTRL_GATED*	0	0	0

The input signals *TIM_CH_IN0* , *TIM_CH_IN1* , *DTM_AUX_IN* and *DTM_PSU_IN* can be used to signalize e.g. an overcurrent situation and activate the phase shift feature in one of the four DTM channels. Over the bit fields **CDTM[i]_DTM[d]_PS_CTRL.PSU_IN_SEL** , **CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL** and **CDTM[i]_DTM[d]_CTRL.DTM_SEL** [1:1] one of the input signals can be selected. Further the polarity of the selected input signal can be inverted by bit field **CDTM[i]_DTM[d]_PS_CTRL.IN_POL** . The resulting signal *EXT_CTRL* is routed over signal *EXT_CTRL_GATED* to the phase shift unit output signal *PSU_SHIFT* [3:0] which controls the phase shift feature in the four DTM channel blocks, see Figure *119 "DTM Channel Overview"* .

Bit field **CDTM[i]_DTM[d]_PS_CTRL.SHIFT_SEL** controls which DTM channel is used for the phase shift feature by selecting the corresponding signal *IN_EDGE* of one of the four DTM channels. The bit field controls in the same manner to which DTM channel the signal *EXT_CTRL_GATED* is routed. For all other, not selected DTM channels signal *PSU_SHIFT* is set to 0 (see LUT table in Figure *123 "Phase Shift Unit"* .
The counter (DOWN COUNTER) counts down while its value is greater than 0 and if the counter reaches 0 it stops. While the counter is running (*BLK_DOWN_CNT* >0) signal *EXT_CTRL_GATED* is 0. Thus, the phase shift output signal *PSU_SHIFT* is 0x0 and the phase shift feature can't be activated. If the counter is 0 *EXT_CTRL_GATED* == *EXT_CTRL* and the phase shift feature can be activated depending on the signal level of the selected input signal *TIM_CH_IN0* or *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] or *DTM_PSU_IN* .
The blanking window usage allows that the phase shift condition (exception condition e.g. overcurrent detected) can be masked for a certain period after an input signal change (e.g. at *DTM_IN*) has occurred.

The activation of the phase shift feature by signal *EXT_CTRL_GATED* =1 can result in an edge at signal *IN[x]_DLY1* .
The deactivation of the phase shift feature by signal *EXT_CTRL_GATED* =0 results in an edge on signal *IN[x]_DLY1* earliest with a signal change at *IN_SIG* (see Figure *119 "DTM Channel Overview"* .

If phase shift feature for DTM channel 1 is used (**CDTM[i]_DTM[d]_PS_CTRL.SHIFT_SEL** ==0b00 signal *PSU_SHIFT* [1:1] activates (if **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_1** =1) the update of the first storage element on channel 1 (i.e. representing *IN[x]_DLY1*) to the input value *DTM_IN_PREV* (*DTM_IN* of channel x-1). If this update leads to an edge, the succeeding part of DTM channel derives the inverse signal and applies the corresponding dead time (i.e. the edge delay) to the output signals of the channel (x=1).
If phase shift feature for DTM channel 2 is used (**CDTM[i]_DTM[d]_PS_CTRL.SHIFT_SEL** ==0b01 signal *PSU_SHIFT* [2:2] activates (if **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_2** =1) the update of the first storage element on channel 2 (i.e. representing *IN[x]_DLY1*) to the input value *DTM_IN_PREV* (*DTM_IN* of channel x-1). If this update leads to an edge, the succeeding part of DTM channel derives the inverse signal and applies the corresponding dead time (i.e. the edge delay) to the output signals of the channel (x=2).
If phase shift feature for DTM channel 3 is used (**CDTM[i]_DTM[d]_PS_CTRL.SHIFT_SEL** ==0b10 signal *PSU_SHIFT* [3:3] activates (if

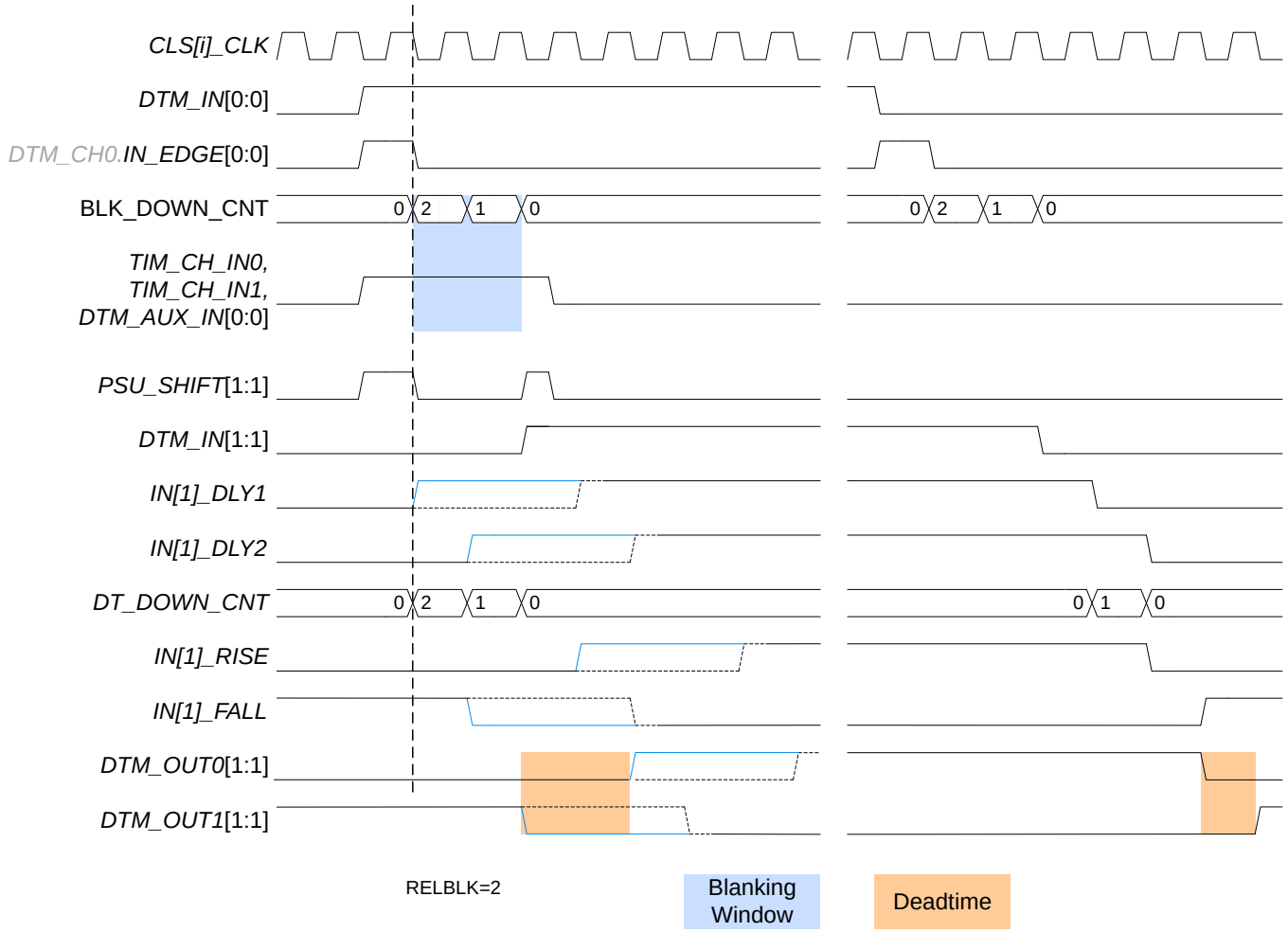
CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_3 =1) the update of the first storage element on channel 3 (i.e. representing *IN[x]_DLY1*) to the input value *DTM_IN_PREV* (*DTM_IN* of channel x−1). If this update leads to an edge, the succeeding part of DTM channel derives the inverse signal and applies the corresponding dead time (i.e. the edge delay) to the output signals of the channel (x=3).

For channel x=0 no blanking window feature and no phase shift feature is available.

Further, the exception signal *PSU_SHIFT* [x:x] can be used as an input to the Boolean function, see *16.4 "Multiple Output Signal Combination"* (next section: Multiple output signal combination). This feature is available for all four DTM channels.

The following figure shows an example of phase shifting on channel 1.

Figure 124 Example Wave of Phase Shift on Channel 1



Note:

The figure above shows the timing for **CDTM[i]_DTM[d]_CTRL.CLK_SEL** =0b00. If another clock resolution is configured, be aware that only the counter DT_DOWN_CNT runs with the reduced clock resolution.

The bit field **CDTM[i]_DTM[d]_CTRL.CH_SHUTOFF_EN** shown in Figure *123 "Phase Shift Unit"* is used for the DTM output shutoff feature which is described in chapter *16.6 "DTM Shut Off"* .

16.4 Multiple Output Signal Combination

Additionally, each DTM channel (see Figure *119 "DTM Channel Overview"*) provides the possibility to combine the channel input signals *DTM_IN* or *DTM_IN_T* with signals *PSU_SHIFT* or *CH_OUT_I* by an AND or an XOR gate.

The relevant input signals of DTM channel [0] for this feature are:

- *DTM_IN* [0:0]
- *DTM_IN_T* [0:0]
- *PSU_SHIFT* [0:0]
- *CH_OUT_I*

Note:

Signal *CH_OUT_I* of channel [0] is equal to signal *DTM_COUT3_I* , see Figure *115 "DTM Overview"* .

The relevant input signals of DTM channel [1] for this feature are:

- *DTM_IN* [1:1]
- *DTM_IN_T* [1:1]
- *PSU_SHIFT* [1:1]
- *CH_OUT_I*

Note:
Signal *CH_OUT_I* of channel [1] is equal to signal *CH_OUT_O* of channel [0], see Figure *115 "DTM Overview"* .

The relevant input signals of DTM channel [2] for this feature are:

- *DTM_IN* [2:2]
- *DTM_IN_T* [2:2]
- *PSU_SHIFT* [2:2]
- *CH_OUT_I*

Note:
Signal *CH_OUT_I* of channel [2] is equal to signal *CH_OUT_O* of channel [1], see Figure *115 "DTM Overview"* .

The relevant input signals of DTM channel [3] for this feature are:

- *DTM_IN* [3:3]
- *DTM_IN_T* [3:3]
- *PSU_SHIFT* [3:3]
- *CH_OUT_I*

Note:
Signal *CH_OUT_I* of channel [3] is equal to signal *CH_OUT_O* of channel [2], see Figure *115 "DTM Overview"* .

The registers **CDTM[i]_DTM[d]_CH_CTRL3.TSEL1_0** , **CDTM[i]_DTM[d]_CH_CTRL3.TSEL1_1** , **CDTM[i]_DTM[d]_CH_CTRL3.TSEL1_2** and **CDTM[i]_DTM[d]_CH_CTRL3.TSEL1_3** select if the signal *DTM_IN* or *DTM_IN_T* is used.

The registers **CDTM[i]_DTM[d]_CH_CTRL3.CII0** , **CDTM[i]_DTM[d]_CH_CTRL3.CII1** , **CDTM[i]_DTM[d]_CH_CTRL3.CII2** and **CDTM[i]_DTM[d]_CH_CTRL3.CII3** configure if the selected input signal *DTM_IN* or *DTM_IN_T* is inverted.

The registers **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_0** , **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_1** , **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_2** and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_3** select if the signal *PSU_SHIFT* or signal *CH_OUT_I* is used.

It is recommended to use the combination of signals only if bit field **CDTM[i]_DTM[d]_PS_CTRL.RELBLK** is 0. Otherwise, the signal *TIM_CH_IN0* , *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] may be disturbed by the blanking window counter.

Together with the inverter inside submodule PSU (see Figure *123 "Phase Shift Unit"* , the inverter on each output of a channel and the possibility to invert the selected input signal (*DTM_IN* or *DTM_IN_T*) inside connected TOM / ATOM / TIO channel, a (N)AND, (N)OR or X(N)OR combination of the signals is possible (see e.g. Table *39 "Function of Combination on DTM Channel x=0 Output"*).

The inverter inside the submodule PSU can be configured by **CDTM[i]_DTM[d]_PS_CTRL.IN_POL** .

The inverter on each output of a DTM channel can be configured by
CDTM[i]_DTM[d]_CH_CTRL2.POL0_0 , **CDTM[i]_DTM[d]_CH_CTRL2.POL1_0** ,
CDTM[i]_DTM[d]_CH_CTRL2.POL0_1 , **CDTM[i]_DTM[d]_CH_CTRL2.POL1_1** ,
CDTM[i]_DTM[d]_CH_CTRL2.POL0_2 , **CDTM[i]_DTM[d]_CH_CTRL2.POL1_2** ,
CDTM[i]_DTM[d]_CH_CTRL2.POL0_3 and **CDTM[i]_DTM[d]_CH_CTRL2.POL1_3** .

16.4.1 Combination of Input Signal TIM_CH_INy/DTM_AUX_IN with TOM/ATOM/TIO Signal

Note:
The description below is given for channel [0], but the same applies for the other DTM channels [1..3] using the corresponding configuration register bit fields.

If the input selection **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_0** is set to 0, the output selection **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL2.DT1_0** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL2.OC1_0** is set to 0 and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** is set to 0, depending on **CDTM[i]_DTM[d]_PS_CTRL.PSU_IN_SEL** either *TIM_CH_IN0* , *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] can be combined with signal *DTM_IN* [0:0] or *DTM_IN_T* [0:0].

The function of combination on DTM output *DTM_OUT1* [0:0] (and also *CH_OUT_O* of channel [0]) is defined by **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** and **CDTM[i]_DTM[d]_CH_CTRL2.POL1_0** in the following way:

Table 39 Function of Combination on DTM Channel x=0 Output

	CDTM[i]_DTM[d]_CH_CTRL1.O1F_0	CDTM[i]_DTM[d]_CH_CTRL2.POL1_0	CDTM[i]_DTM[d]_PS_CTRL.IN_POL	TOM/ATOM/TIO output inverted
XOR	01	0	0	no
AND	10	0	0	no
XNOR	01	1	0	no
NAND	10	1	0	no
XNOR	01	1	1	yes
OR	10	1	1	yes
XOR	01	0	1	yes
NOR	10	0	1	yes

Table 40 Function of Combination on DTM Channel x=1 Output

	CDTM[i]_DTM[d]_CH_CTRL1.O1F_1	CDTM[i]_DTM[d]_CH_CTRL2.POL1_1	CDTM[i]_DTM[d]_PS_CTRL.IN_POL	TOM/ATOM/TIO output inverted
XOR	01	0	0	no
AND	10	0	0	no
XNOR	01	1	0	no
NAND	10	1	0	no
XNOR	01	1	1	yes
OR	10	1	1	yes
XOR	01	0	1	yes
NOR	10	0	1	yes

Table 41 Function of Combination on DTM Channel x=2 Output

	CDTM[i]_DTM[d]_CH_CTRL1.O1F_2	CDTM[i]_DTM[d]_CH_CTRL2.POL1_2	CDTM[i]_DTM[d]_PS_CTRL.IN_POL	TOM/ATOM/TIO output inverted
XOR	01	0	0	no
AND	10	0	0	no
XNOR	01	1	0	no
NAND	10	1	0	no
XNOR	01	1	1	yes
OR	10	1	1	yes
XOR	01	0	1	yes
NOR	10	0	1	yes

Table 42 Function of Combination on DTM Channel x=3 Output

	CDTM[i]_DTM[d]_CH_CTRL1.O1F_3	CDTM[i]_DTM[d]_CH_CTRL2.POL1_3	CDTM[i]_DTM[d]_PS_CTRL.IN_POL	TOM/ATOM/TIO output inverted
XOR	01	0	0	no
AND	10	0	0	no
XNOR	01	1	0	no
NAND	10	1	0	no
XNOR	01	1	1	yes
OR	10	1	1	yes
XOR	01	0	1	yes
NOR	10	0	1	yes

Note:

The inversion of the (A)TOM output can be achieved by switching the **ATOM[i]_CH[x]_CTRL.SL** bit (for TOM and ATOM SOMP/SOMC mode).

16.4.2 Combination of Multiple TOM/ATOM/TIO Output Signals

Note:
The description below is given for channel [1], but the same applies for the DTM channels [2..3] using the corresponding configuration registers bit fields.

If the input selection **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_1** is set to 1, the output selection **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_1** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL2.DT1_1** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL2.OC1_1** is set to 0 and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** is set to 0, the output of the preceding DTM channel *CH_OUT_I* can be combined with signal *DTM_IN* [1:1] or with signal *DTM_IN_T* [1:1] depending on **CDTM[i]_DTM[d]_CH_CTRL3.TSEL1_1** .
The function of combination on DTM output *DTM_OUT1* [1:1] is defined by **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** , **CDTM[i]_DTM[d]_CH_CTRL2.POL1_1** and **CDTM[i]_DTM[d]_CH_CTRL2.POL1_0** in the following way:

Table 43 Function of Combination on DTM on Channel [1] Output DTM_OUT1[1:1]

	CDTM[i]_DTM[d]_CH_CTRL1.O1F_1	CDTM[i]_DTM[d]_CH_CTRL2.POL1_1	CDTM[i]_DTM[d]_CH_CTRL2.POL1_0	TOM/ATOM/TIO output inverted
XOR	01	0	0	no
AND	10	0	0	no
XNOR	01	1	0	no
NAND	10	1	0	no
XNOR	01	1	1	yes
OR	10	1	1	yes
XOR	01	0	1	yes
NOR	10	0	1	yes

Table 44 Function of Combination on DTM on Channel [2] Output DTM_OUT1[2:2]

	CDTM[i]_DTM[d]_CH_CTRL1.O1F_2	CDTM[i]_DTM[d]_CH_CTRL2.POL1_2	CDTM[i]_DTM[d]_CH_CTRL2.POL1_1	TOM/ATOM/TIO output inverted
XOR	01	0	0	no
AND	10	0	0	no
XNOR	01	1	0	no
NAND	10	1	0	no
XNOR	01	1	1	yes
OR	10	1	1	yes
XOR	01	0	1	yes
NOR	10	0	1	yes

Table 45 Function of Combination on DTM on Channel [3] Output DTM_OUT1[3:3]

	CDTM[i]_DTM[d]_CH_CTRL1.O1F_3	CDTM[i]_DTM[d]_CH_CTRL2.POL1_3	CDTM[i]_DTM[d]_CH_CTRL2.POL1_2	TOM/ATOM/TIO output inverted
XOR	01	0	0	no
AND	10	0	0	no
XNOR	01	1	0	no
NAND	10	1	0	no
XNOR	01	1	1	yes
OR	10	1	1	yes
XOR	01	0	1	yes
NOR	10	0	1	yes

By setting **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_0** to 1, **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_1** to 1, **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_2** to 1 and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_3** to 1, a combination of all four signals *DTM_IN* [0:0], *DTM_IN* [1:1], *DTM_IN* [2:2] and *DTM_IN* [3:3] can be achieved (combinatorial chain).

If the result of the combination of the signals should be visible at the DTM output *DTM_OUT0* [x:x], the internal signals just in front of the output register stage can be swapped by setting **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** to 1 for channel 0, **CDTM[i]_DTM[d]_CH_CTRL1.-**

SWAP_1 to 1 for channel 1, **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** to 1 for channel 2 and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** to 1 for channel 3.

16.4.3 Pulse Generation on Edge

Another feature of the DTM is to generate on the second output *DTM_OUT1* [x:x] a pulse on every edge of corresponding input signal *DTM_IN* [x:x].

This can be achieved for channel [0] by configuring **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** to 1, i.e. selecting signal *EDGE_TRIGG_[0]* as the output signal (**CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** has to be 0b00).

This can be reached for channel [1] by configuring **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_1** to 1, i.e. selecting signal *EDGE_TRIGG_[1]* as the output signal (**CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** has to be 0b00).

This can be reached for channel [2] by configuring **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_2** to 1, i.e. selecting signal *EDGE_TRIGG_[2]* as the output signal (**CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** has to be 0b00).

This can be reached for channel [3] by configuring **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_3** to 1, i.e. selecting signal *EDGE_TRIGG_[3]* as the output signal (**CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** has to be 0b00).

The signal *EDGE_TRIGG_[x]* is depicted in Figure *120 "Wave Signals for Function of Dead Time Generation"* .

The pulse length can be adjusted individually for each edge type by the configuration value **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** and **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** .
The parameter **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** defines the pulse length in case of a rising edge on input *DTM_IN* [x:x], the parameter **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** defines the pulse length in case of a falling edge on input *DTM_IN* [x:x].

The generated edge signal *EDGE_TRIGG_[x]* can be combined with the output signal of the preceding DTM channel [x−1] at channel input *CH_OUT_I* .

To route the output of the XOR gate to *DTM_OUT1* , here for example for channel 0, the **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** is set to 0b01, **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL2.DT1_0** is set to 1 and **CDTM[i]_DTM[d]_CH_CTRL2.SL1_0** is set to 0. For the other channels the corresponding configuration bits have to be set.

With the configuration **CDTM[i]_DTM[d]_CH_CTRL3.CIS0** is set to 1 and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_0** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL3.CII0** is set to 0 and **CDTM[i]_DTM[d]_CH_CTRL2.POL1_3** is set to 1, the signal *EDGE_TRIGG_[0]* is XORed with the inverse signal *DTM_COUT3_I* (*CH_OUT_O* of channel [3]).

With the configuration **CDTM[i]_DTM[d]_CH_CTRL3.CIS1** is set to 1 and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_1** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL3.CII1** is set to 0 and **CDTM[i]_DTM[d]_CH_CTRL2.POL1_0** is set to 1, the signal *EDGE_TRIGG_[1]* is XORed with the inverse signal *COUT* [0:0] (*CH_OUT_O* of channel [0]).

With the configuration **CDTM[i]_DTM[d]_CH_CTRL3.CIS2** is set to 1 and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_2** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL3.CII2** is set to 0 and **CDTM[i]_DTM[d]_CH_CTRL2.POL1_1** is set to 1, the signal *EDGE_TRIGG_[2]* is XORed with the inverse signal *COUT* [1:1] (*CH_OUT_O* of channel [1]).

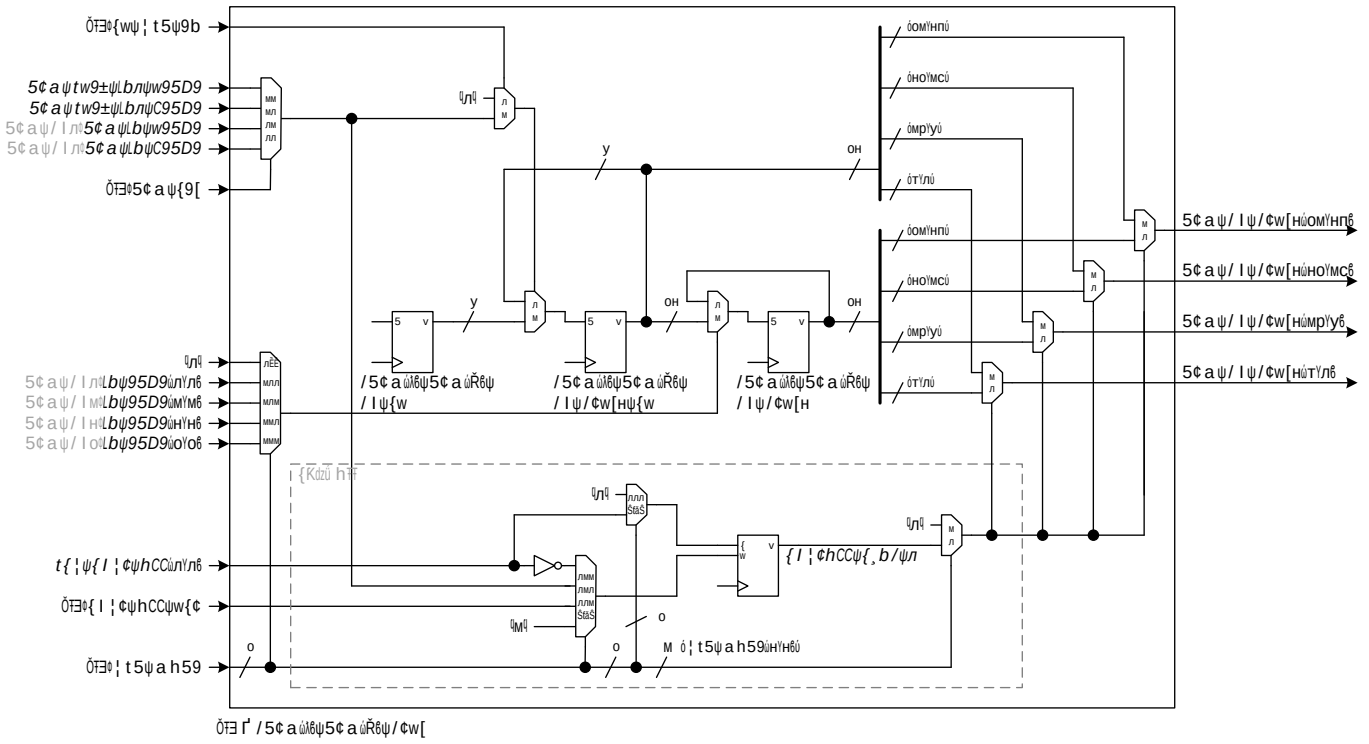
With the configuration **CDTM[i]_DTM[d]_CH_CTRL3.CIS3** is set to 1 and **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_3** is set to 1, **CDTM[i]_DTM[d]_CH_CTRL3.CII3** is set to 0 and **CDTM[i]_DTM[d]_CH_CTRL2.POL1_2** is set to 1, the signal *EDGE_TRIGG_[3]* is XORed with the inverse signal *COUT* [2:2] (*CH_OUT_O* of channel [2]).

As a result of this configuration, one can generate at each edge on DTM input *DTM_IN* [x:x] a pulse signal and OR-combine these generated pulse signals with the generated signal of preceding DTM channel. If the combinatorial chain is configured over all four DTM channels the final signal is available at last DTM output, which is, according to Figure *115 "DTM Overview"* the *DTM_COUT3_O* and with one clock delay the *DTM_OUT1* [3:3].

16.5 Synchronous Update of Channel Control Register 2

It is possible to use the shadow register **CDTM[i]_DTM[d]_CH_CTRL2_SR** and a selected edge of one of the channel 0 to channel 3 to update the operation register **CDTM[i]_DTM[d]_CH_CTRL2** .
The update mechanism and its configuration are depicted in the following figure.

Figure 125 Synchronous Update Mechanism Of Register CDTM[i]_DTM[d]_CH_CTRL2



If enabled by the bit field **CDTM[i]_DTM[d]_CTRL.UPD_MODE** (i.e. **CDTM[i]_DTM[d]_CTRL.UPD_MODE** =1xx), the register **CDTM[i]_DTM[d]_CH_CTRL2_SR** serves as a shadow register of register **CDTM[i]_DTM[d]_CH_CTRL2** . The update is then triggered by an edge of signal *IN_EDGE* [3:0].

The synchronous update allows the user to change output polarity. The selection of the constant signal level and the routing from DTM inputs (*DTM_IN* , *DTM_IN_T*) to outputs (*DTM_OUT0* , *DTM_OUT1*) is synchronized for all four channels in parallel to one of the input edges on *IN_EDGE* [3:0].

Any asynchronous configuration change (asynchronous means a configuration change will be done immediately without any edge detection) can cause a change of *DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* outputs without any input edge detection, even if the corresponding output *DTM_OUT0* [x:x] or *DTM_OUT1* [x:x] also changes.

16.6 DTM Shut Off

A fast shutoff for the eight outputs of DTM instance d can be triggered by one of the two assigned inputs *TIM_CH_IN0* , *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] or the corresponding three inputs of the previous DTM instance d-1 by selecting signal *DTM_PSU_IN* . The selection of the trigger signal source is done by the bits **CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL** , **CDTM[i]_DTM[d]_PS_CTRL.PSU_IN_SEL** and **CDTM[i]_DTM[d]_CTRL.DTM_SEL** [1:1] (see Figure *123 "Phase Shift Unit"*). The selected trigger signal (output signal in *123 "Phase Shift Unit"* and input signal *125 "Synchronous Update Mechanism Of Register CDTM[i]_DTM[d]_CH_CTRL2"*) is named *PSU_SHUT_OFF* .
Enabling of the shutoff feature is done by setting **CDTM[i]_DTM[d]_CTRL.UPD_MODE** [2:0] to one of the values 0b001, 0b010 or 0b011.

The shutoff behavior of the DTM outputs is defined by the value of register **CDTM[i]_DTM[d]_CH_CTRL2_SR** .
If the shutoff feature is enabled by **CDTM[i]_DTM[d]_CTRL.UPD_MODE** , the register **CDTM[i]_DTM[d]_CH_CTRL2** defines the output signal behavior as long as signal *SHUTOFF_SYNC_0* is 0. If signal *SHUTOFF_SYNC_0* is 1 the output signal behavior is defined by register **CDTM[i]_DTM[d]_CH_CTRL2_SR** .
Signal *SHUTOFF_SYNC_0* is set to 1 if signal *PSU_SHUT_OFF* switches to 1. The reset depends on value of **CDTM[i]_DTM[d]_CTRL.UPD_MODE** .

There are four different ways to reset the signal *SHUTOFF_SYNC_0* to 0:

- A write access of value 1 via the configuration interface to bit **CDTM[i]_DTM[d]_CTRL.SHUT_OFF_RST** (with **CDTM[i]_DTM[d]_CTRL.UPD_MODE** = 0b001).
- Synchronous to an edge on DTM channel 0 input of this DTM instance d or on an edge on DTM channel 0 input of preceding DTM instance d-1 (with **CDTM[i]_DTM[d]_CTRL.UPD_MODE** = 0b010).
- Asynchronous to any DTM channel input edge, if signal *PSU_SHUT_OFF* switches back to 0 (with **CDTM[i]_DTM[d]_CTRL.UPD_MODE** = 0b011).
- Additionally, setting **CDTM[i]_DTM[d]_CTRL.UPD_MODE** to a value 0b000 or 0b1xx resets also the signal *SHUTOFF_SYNC_0* , except *PSU_SHUT_OFF* is set and **CDTM[i]_DTM[d]_CTRL.UPD_MODE** is unequal to 0b000.

Figure *125 "Synchronous Update Mechanism Of Register CDTM[i]_DTM[d]_CH_CTRL2"* depicts the shutoff feature and the different shutoff release possibilities.

The reset of *SHUTOFF_SYNC_0* has lower priority than the set of this signal.

A second shadow register **CDTM[i]_DTM[d]_CH_SR** exists for the eight bits of the shadow register **CDTM[i]_DTM[d]_CH_CTRL2_SR** . If enabled by configuration bit **CDTM[i]_DTM[d]_CTRL.SR_UPD_EN** , the update of **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_0_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_0_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_1_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_1_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_2_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_2_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_3_SR** and **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_3_SR** bits can be triggered by one of the signals selected by bit field **CDTM[i]_DTM[d]_CTRL.DTM_SEL** . This trigger signal is either the rising or the falling edge detected on DTM channel [0] (*DTM_IN_FEDGE* or *DTM_IN_REDGE*) or the corresponding edges of the DTM channel [0] of a preceding instance d−1 (*DTM_PREV_IN0_FEDGE* or *DTM_PREV_IN0_REDGE*).

If an update trigger occurs in the same clock cycle, a write access to **CDTM[i]_DTM[d]_CH_CTRL2_SR** is done and the register is written except the eight bits: **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_0_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_0_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_1_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_1_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_2_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_2_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_3_SR** and **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_3_SR** . The eight bits are copied from **CDTM[i]_DTM[d]_CH_SR** . In this case **CDTM[i]_DTM[d]_CH_SR** has priority over AEI write data.

As depicted in Figure *117 "Connections of TIM to DTM Inputs TIM_CH_IN0/TIM_CH_IN1 for CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL=0"* , Figure *118 "Connections of TIM to DTM Inputs TIM_CH_IN0/TIM_CH_IN1 for CDTM[i]_DTM[d]_PS_CTRL.TIM_SEL=1"* and Figure *123 "Phase Shift Unit"* the DTM input signal *TIM_CH_IN0* , *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] can be forwarded to the succeeding instance d+1. Thus, it can be used to trigger shutoff in two consecutive DTM instances.

16.7 DTM Individual Channel Shut Off

The shutoff feature can be enhanced by setting bit **CDTM[i]_DTM[d]_CTRL.CH_SHUTOFF_EN** = 1. Then all four DTM channels can be shutoff individually. If an individual shutoff is done (*SHUTOFF_SYNC_0* , *SHUTOFF_SYNC_1* , *SHUTOFF_SYNC_2* and *SHUTOFF_SYNC_3* is set) the configuration bits of the corresponding DTM channel (e.g. for channel 0 the bits [7:0]) are copied from the shadow register **CDTM[i]_DTM[d]_CH_CTRL2_SR** to output **CDTM[i]_DTM[d]_CH_CTRL2** .

As described above, there are four different ways to reset the signal *SHUTOFF_SYNC_0* to 0. The configuration bits **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_0** are described below:

- Asynchronous to any DTM channel input edge, if signal *PSU_SHUT_OFF* [0:0] switches back to 0 (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_0** = 0b11)
- Synchronous to an edge on DTM channel 0 input of this DTM instance d or on an edge on DTM channel 0 input of preceding DTM instance d−1 (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_0** = 0b10).
- The CPU writes a 1 to bit **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_0** (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_0** = 0b01).
- Additionally, setting **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_0** to a value 0b00.

As described above, there are four different ways to reset the signal *SHUTOFF_SYNC_1* to 0. The configuration bits **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_1** are described below:

- Asynchronous to any DTM channel input edge, if signal *PSU_SHUT_OFF* [1:1] switches back to 0 (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_1** = 0b11)
- Synchronous to an edge on DTM channel 0 input of this DTM instance d or on an edge on DTM channel 0 input of preceding DTM instance d−1 (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_1** = 0b10).
- The CPU writes a 1 to bit **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_1** (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_1** = 0b01).
- Additionally, setting **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_1** to a value 0b00.

As described above, there are four different ways to reset the signal *SHUTOFF_SYNC_2* to 0. The configuration bits **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_2** are described here:

- Asynchronous to any DTM channel input edge, if signal *PSU_SHUT_OFF* [2:2] switches back to 0 (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_2** = 0b11)
- Synchronous to an edge on DTM channel 0 input of this DTM instance d or on an edge on DTM channel 0 input of preceding DTM instance d−1 (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_2** = 0b10).
- The CPU writes a 1 to bit **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_2** (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_2** = 0b01).
- Additionally, setting **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_2** to a value 0b00.

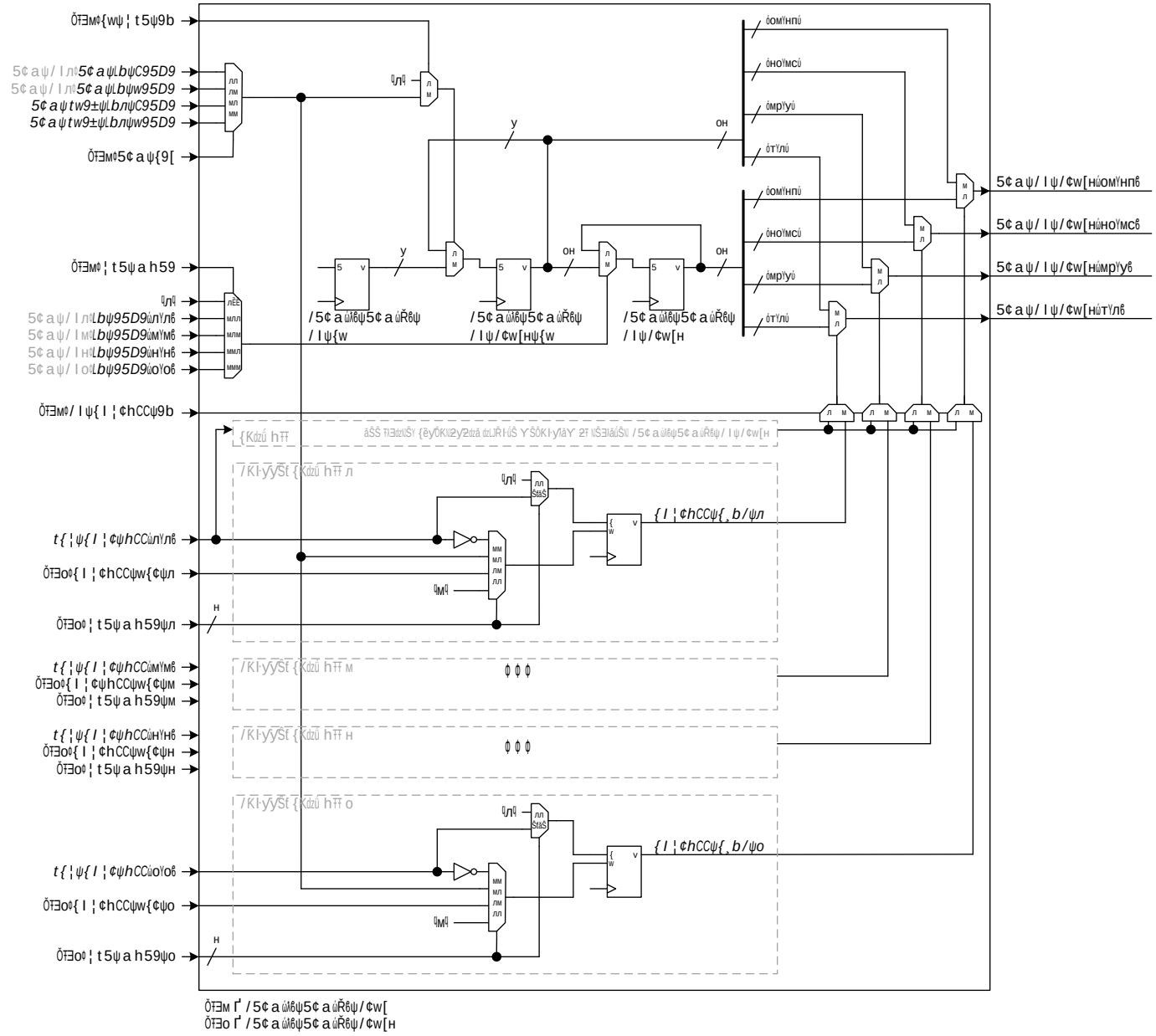
As described above, there are four different ways to reset the signal *SHUTOFF_SYNC_3* to 0. The configuration bits **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_3** are described here:

- Asynchronous to any DTM channel input edge, if signal *PSU_SHUT_OFF* [3:3] switches back to 0 (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_3** = 0b11)
- Synchronous to an edge on DTM channel 0 input of this DTM instance d or on an edge on DTM channel 0 input of preceding DTM instance d−1 (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_3** = 0b10).
- The CPU writes a 1 to bit **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_3** (with **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_3** = 0b01).

- Additionally, setting **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_3** to a value 0b00.

Figure *126 "Individual Channel Shut Off"* depicts the channel shutoff feature and the different shutoff release possibilities.

Figure 126 Individual Channel Shut Off



The reset of a *SHUTOFF_SYNC_0* , *SHUTOFF_SYNC_1* , *SHUTOFF_SYNC_2* and *SHUTOFF_SYNC_3* has lower priority than the set of such a signal.

If the individual channel shutoff feature is activated (**CDTM[i]_DTM[d]_CTRL.CH_SHUTOFF_EN** = 1) the selection of the trigger source located in DTM block Phase Shift Unit (PSU) is also enhanced.

By using the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_0** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_1** , **CDTM[i]_DTM[d]_CTRL2.-SHUTOFF_SEL_2** and **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_3** for each DTM channel one of the following sources can be selected:

- *TIM_CH_IN0*
- *TIM_CH_IN1*
- *DTM_AUX_IN* [0:0]
- *DTM_AUX_IN* [1:1]
- *DTM_AUX_IN* [2:2]
- *DTM_AUX_IN* [3:3]
- constant zero (0b0)

- constant one (0b1)

The routing from the external ports *GTM_CDTM[i]_DTM[d]_AUX_IN* (with d being the index of the DTM within the CDTM and i being the index of the hosting cluster) to *DTM_AUX_IN* is summarized below (see also *9 "Routing of DTM AUX Input Signals"*):
The following list shows the possible input sources.

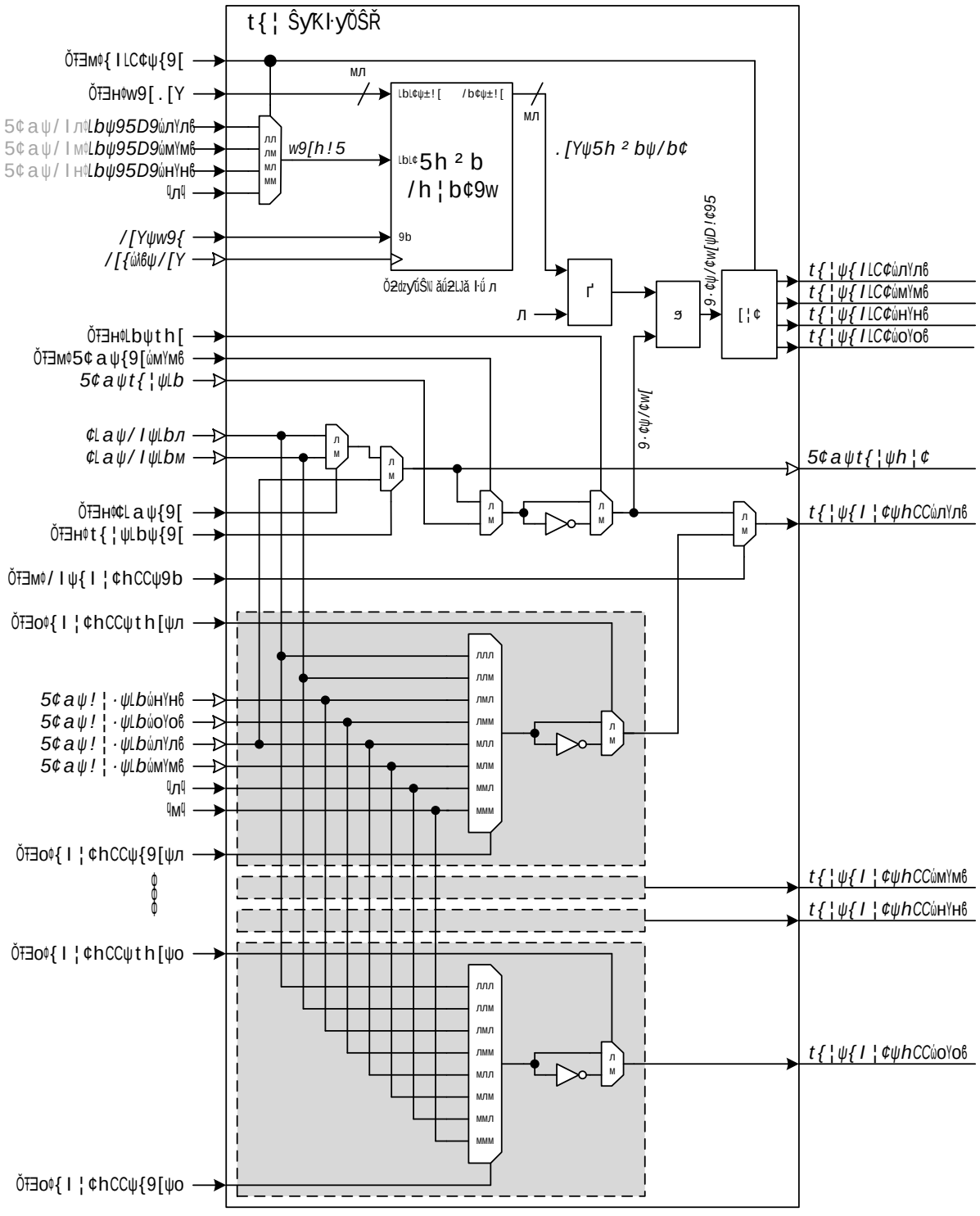
- *DTM_AUX_IN* [0:0] is wired to the respective *GTM_CDTM[i]_DTM[d]_AUX_IN* [0:0].
- *DTM_AUX_IN* [1:1] is wired to the respective *GTM_CDTM[i]_DTM[d]_AUX_IN* [1:1].
- *DTM_AUX_IN* [2:2] is wired to *GTM_CDTM[i]_DTM[d-1]_AUX_IN* [0:0]. If the previous DTM is not implemented or is placed after a module of a different type from the current one (i.e. d ∈ {4,6}), *DTM_AUX_IN* [2:2] is tied to '0'.
- *DTM_AUX_IN* [3:3] is wired to *GTM_CDTM[i]_DTM[d-1]_AUX_IN* [1:1]. If the previous DTM is not implemented or is placed after a module of a different type from the current one (i.e. d ∈ {4,6}), *DTM_AUX_IN* [3:3] is tied to '0'.

Note:

For sake of compatibility to previous GTM versions, *DTM_AUX_IN* [0:0] has to be driven for each CDTM[i]_DTM[d] instance by the *DTM_AUX_IN* [0:0] primary input.

After each source multiplexer there is an inverter. Over bit **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_0** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_1** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_2** and **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_3** signal *PSU_SHUT_OFF* [0:0], *PSU_SHUT_OFF* [1:1], *PSU_SHUT_OFF* [2:2] and *PSU_SHUT_OFF* [3:3] can be inverted.

Figure 127 PSU Enhanced



LUT

cfg1.**SHIFT_SEL**	*PSU_SHIFT*[0:0]	*PSU_SHIFT*[1:1]	*PSU_SHIFT*[2:2]	*PSU_SHIFT*[3:3]
00	0	*DTM_IN_POL*	0	0
01	0	0	*DTM_IN_POL*	0
10	0	0	0	*DTM_IN_POL*
11	*DTM_IN_POL*	0	0	0

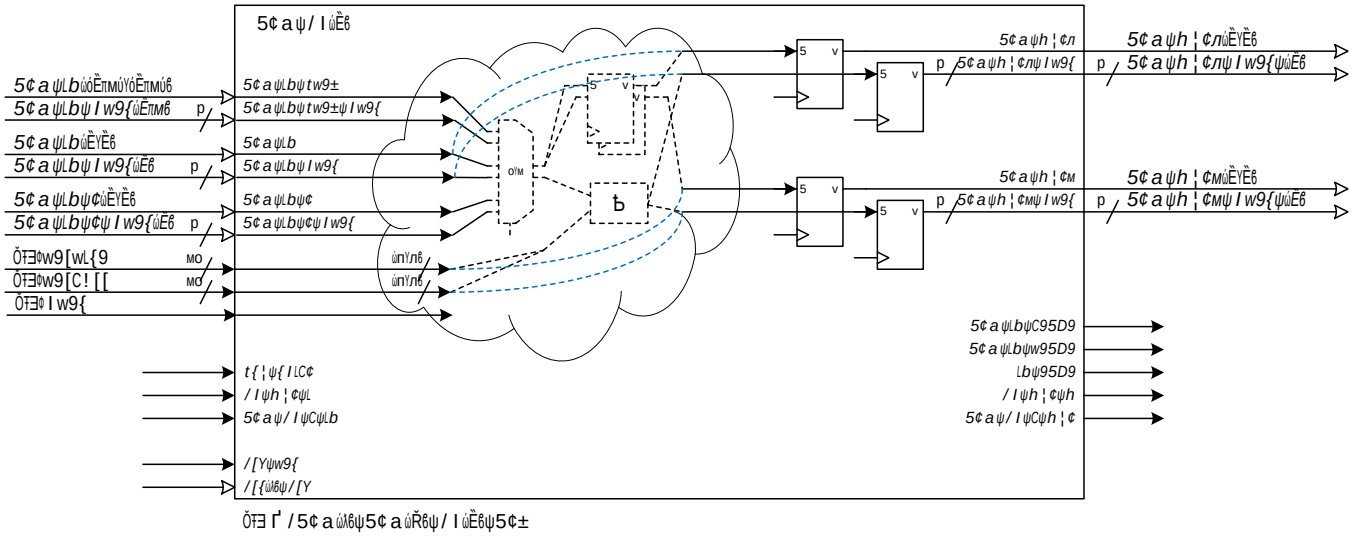
The register layout of **CDTM[i]_DTM[d]_CTRL2** ensures that atomic operations on multiple DTM channels are possible. Therefore, the register includes the bits **CDTM[i]_DTM[d]_CTRL2.WR_EN_0** , **CDTM[i]_DTM[d]_CTRL2.WR_EN_1** , **CDTM[i]_DTM[d]_CTRL2.WR_EN_2** and **CDTM[i]_DTM[d]_CTRL2.WR_EN_3** . The bits of **CDTM[i]_DTM[d]_CTRL2** of a DTM channel (x∈{0...3}) can be written only if

the corresponding **CDTM[i]_DTM[d]_CTRL2.WR_EN_0** , **CDTM[i]_DTM[d]_CTRL2.WR_EN_1** , **CDTM[i]_DTM[d]_CTRL2.WR_EN_2** and **CDTM[i]_DTM[d]_CTRL2.WR_EN_3** bits are also written to 1.

16.8 High Resolution PWM Support

Additionally to each incoming PWM signal *DTM_IN* [x:x], *DTM_IN_T* [x:x] x:={0, 1,..., 3} another input signal is delivered from TOM/ATOM submodule which is called *DTM_IN_HRES[x]* , *DTM_IN_T_HRES[x]* . Further each channel x has an additional input signal pair *DTM_IN_PREV* [x:x] and *DTM_IN_PREV_HRES[x]* , which is the input signal pair of the previous DTM channel. All these signals are directly connected to the related TOM/ATOM channel. Please refer to chapter *3.8 "High Resolution PWM Support"* for additional information of the high resolution PWM support of the GTM-IP. Figure *128 "DTM Channel Enhancement for High Resolution PWM Support"* shows the high resolution PWM related DTM channel.

Figure 128 DTM Channel Enhancement for High Resolution PWM Support

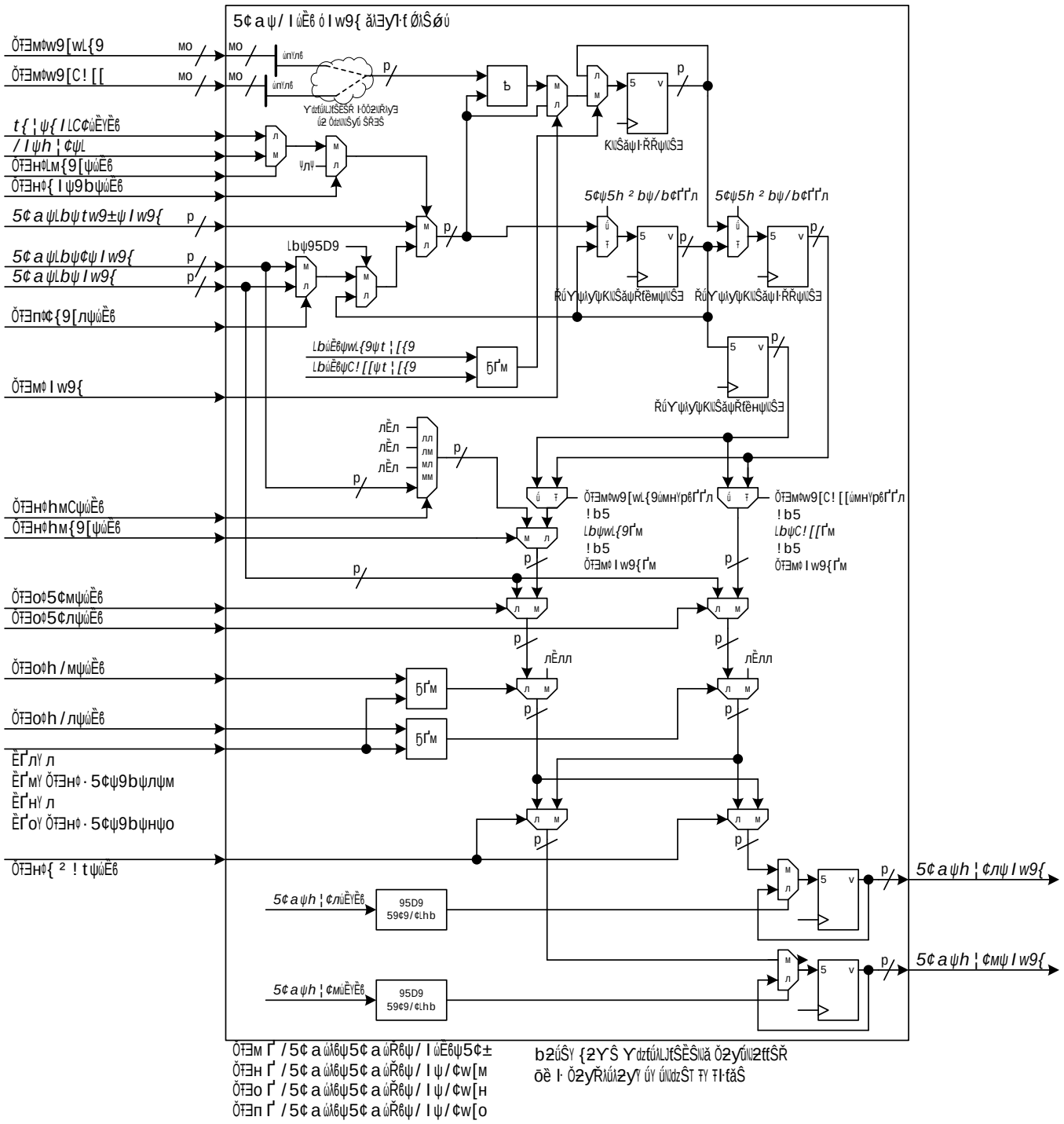


- As shown in the figure above, one of the three input signals *DTM_IN* [3:0], *DTM_IN_T* [3:0] and *DTM_IN_PREV* [3:0] (DTM_IN_PREV is the DTM_IN signal of the previous channel [x-1]) that can be routed to the DTM outputs. The DTM module delivers two output signals *DTM_OUT0* [3:0] and *DTM_OUT1* [3:0] for the selected input signal.
- The corresponding high resolution signals *DTM_IN_HRES[x]* [4:0], *DTM_IN_T_HRES[x]* [4:0] and *DTM_IN_PREV_HRES[x]* [4:0] are routed to the outputs *DTM_OUT0_HRES_[x]* [4:0] and *DTM_OUT1_HRES_[x]* [4:0] in the same manner.

Any additional delay on the path from the selected DTM input to the DTM output inside the logic cloud are also added to the high resolution signals. Thus for each channel the *DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* signals are switching synchronous to the *DTM_OUT0* [x:x], *DTM_OUT1* [x:x] signals respectively. Details about the routing through the logic cloud are described in the previous chapters of the DTM, figure *119 "DTM Channel Overview"* and figure *129 "Routing of High Resolution Signals inside DTM Channel"* .
The blue lines in the figure above show the direct path from the inputs to the outputs through the logic cloud. If this path is configured the high resolution input values from TOM/ATOM are not modified by the DTM module.

The next figure shows the principal routing of the high resolution signals only through the DTM channel.

Figure 129 Routing of High Resolution Signals inside DTM Channel



A configuration bit **CDTM[i]_DTM[d]_CH[x]_DTV.HRES** exists to switch on/off the high resolution support.

If the configuration bit **CDTM[i]_DTM[d]_CH[x]_DTV.HRES** = 0, the output signals *DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* are switching synchronous to the *DTM_OUT0* , *DTM_OUT1* signals and always follows exactly the selected input signal *DTM_IN_HRES[x]* , *DTM_IN_T_HRES[x]* , and *DTM_IN_PREV_HRES[x]* with the same number of clock cycles delay as the PWM output signals *DTM_OUT0* [3:0], *DTM_OUT1* [3:0] relating to the PWM input signal *DTM_IN* [3:0].

If the DTM output signals are swapped (**CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** =1, **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** =1, **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** =1 or **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** =1), then the high resolution output signals are also swapped. This is independent of **CDTM[i]_DTM[d]_CH[x]_DTV.HRES** .

Note:

- *DTM_OUT0_HRES_[x]* = 0, when **CDTM[i]_DTM[d]_CH_CTRL2.OC0_0** = 1/ **CDTM[i]_DTM[d]_CH_CTRL2.OC0_1** = 1/ **CDTM[i]_DTM[d]_CH_CTRL2.OC0_2** = 1/ **CDTM[i]_DTM[d]_CH_CTRL2.OC0_3** = 1 and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** = 0

- *DTM_OUT1_HRES_[x]* = 0, when **CDTM[i]_DTM[d]_CH_CTRL2.OC1_0** = 1/ **CDTM[i]_DTM[d]_CH_CTRL2.OC1_1** = 1/ **CDTM[i]_DTM[d]_CH_CTRL2.OC1_2** = 1/ **CDTM[i]_DTM[d]_CH_CTRL2.OC1_3** = 1 = 1 and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** = 0
- *DTM_OUT1_HRES_[x]* = 0, when **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** = 1/ **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_1** = 1/ **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_2** = 1/ **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_3** = 1 and **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** != 0b11/ **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** != 0b11/ **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** != 0b11/ **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** != 0b11 and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** = 0
- For the above three statements, **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** = 0/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** = 0 is assumed. Please see description above for **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** = 1/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** = 1/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** = 1/ **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** = 1.

Note:

The high resolution input signal changes its value exactly (!! and only !!) with its dedicated input signal edge. If there are no edges on the PWM input signal (DTM_IN/DTM_IN_T), then there are also no changes on the dedicated high resolution input signal. This depends only on the signal generation in the (A)TOM modules.

Note:

The high resolution output signals *DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* will change only, if there is an edge on the associated output signal *DTM_OUT0* [x:x], *DTM_OUT1* [x:x].
But this is only true, if the output edge occurs due to an input edge or due to a change of the configuration bits DT0/DT1 for standard dead time. This configuration change will also lead to an edge on one of the output signals.

- If **CDTM[i]_DTM[d]_CH_CTRL2.OC0_[0]** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.OC1_[0]** =1 (and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** =0) then the signal level bits **CDTM[i]_DTM[d]_CH_CTRL2.SL0_0** and **CDTM[i]_DTM[d]_CH_CTRL2.SL1_0** define the values of *DTM_OUT0* [0:0] and *DTM_OUT1* [0:0] and the high resolution signals *DTM_OUT0_HRES_[0]* and *DTM_OUT1_HRES_[0]* are set to 0x0. Thus the activation of the SL path can lead to a change of *DTM_OUT0_HRES_[0]* and *DTM_OUT1_HRES_[0]* from its current value to 0x0. After activation of the SL path a configuration change of the signal level bits **CDTM[i]_DTM[d]_CH_CTRL2.SL0_0** , **CDTM[i]_DTM[d]_CH_CTRL2.SL1_0** will lead to an edge on the output signals *DTM_OUT0* [0:0] and *DTM_OUT1* [0:0], but not to an update of the high resolution output signals *DTM_OUT0_HRES_[0]* and *DTM_OUT1_HRES_[0]* .
- If **CDTM[i]_DTM[d]_CH_CTRL2.OC0_[1]** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.OC1_[1]** =1 (and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** =0) then the signal level bits **CDTM[i]_DTM[d]_CH_CTRL2.SL0_1** and **CDTM[i]_DTM[d]_CH_CTRL2.SL1_1** define the values of *DTM_OUT0* [1:1] and *DTM_OUT1* [1:1] and the high resolution signals *DTM_OUT0_HRES_[1]* and *DTM_OUT1_HRES_[1]* are set to 0x0. Thus the activation of the SL path can lead to a change of *DTM_OUT0_HRES_[1]* and *DTM_OUT1_HRES_[1]* from its current value to 0x0. After activation of the SL path a configuration change of the signal level bits **CDTM[i]_DTM[d]_CH_CTRL2.SL0_1** , **CDTM[i]_DTM[d]_CH_CTRL2.SL1_1** will lead to an edge on the output signals *DTM_OUT0* [1:1] and *DTM_OUT1* [1:1], but not to an update of the high resolution output signals *DTM_OUT0_HRES_[1]* and *DTM_OUT1_HRES_[1]* .
- If **CDTM[i]_DTM[d]_CH_CTRL2.OC0_[2]** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.OC1_[2]** =1 (and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** =0) then the signal level bits **CDTM[i]_DTM[d]_CH_CTRL2.SL0_2** and **CDTM[i]_DTM[d]_CH_CTRL2.SL1_2** define the values of *DTM_OUT0* [2:2] and *DTM_OUT1* [2:2] and the high resolution signals *DTM_OUT0_HRES_[2]* and *DTM_OUT1_HRES_[2]* are set to 0x0. Thus the activation of the SL path can lead to a change of *DTM_OUT0_HRES_[2]* and *DTM_OUT1_HRES_[2]* from its current value to 0x0. After activation of the SL path a configuration change of the signal level bits **CDTM[i]_DTM[d]_CH_CTRL2.SL0_2** , **CDTM[i]_DTM[d]_CH_CTRL2.SL1_2** will lead to an edge on the output signals *DTM_OUT0* [2:2] and *DTM_OUT1* [2:2], but not to an update of the high resolution output signals *DTM_OUT0_HRES_[2]* and *DTM_OUT1_HRES_[2]* .
- If **CDTM[i]_DTM[d]_CH_CTRL2.OC0_[3]** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.OC1_[3]** =1 (and **CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** =0) then the signal level bits **CDTM[i]_DTM[d]_CH_CTRL2.SL0_3** and **CDTM[i]_DTM[d]_CH_CTRL2.SL1_3** define the values of *DTM_OUT0* [3:3] and *DTM_OUT1* [3:3] and the high resolution signals *DTM_OUT0_HRES_[3]* and *DTM_OUT1_HRES_[3]* are set to 0x0. Thus the activation of the SL path can lead to a change of *DTM_OUT0_HRES_[3]* and *DTM_OUT1_HRES_[3]* from its current value to 0x0. After activation of the SL path a configuration change of the signal level bits **CDTM[i]_DTM[d]_CH_CTRL2.SL0_3** , **CDTM[i]_DTM[d]_CH_CTRL2.SL1_3** will lead to an edge on the output signals *DTM_OUT0* [3:3] and *DTM_OUT1* [3:3], but not to an update of the high resolution output signals *DTM_OUT0_HRES_[3]* and *DTM_OUT1_HRES_[3]* .

If the configuration bit **CDTM[i]_DTM[d]_CH[x]_DTV.HRES** = 1, it is possible to insert dead time between the two inverted output signals *DTM_OUT0* [3:0] and *DTM_OUT1* [3:0] on a micro-step level by calculating the output signals *DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* in the following manner.

The output values are calculated by following function for rising and falling edges of the output signals *DTM_OUT0* [3:0], *DTM_OUT1* [3:0].

For DTM input rising edge detection:

- *DTM_OUT0_HRES_[x]* = *DTM_IN_HRES[x]* + **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** [4:0]
- *DTM_OUT1_HRES_[x]* = *DTM_IN_HRES[x]*

For DTM input falling edge detection:

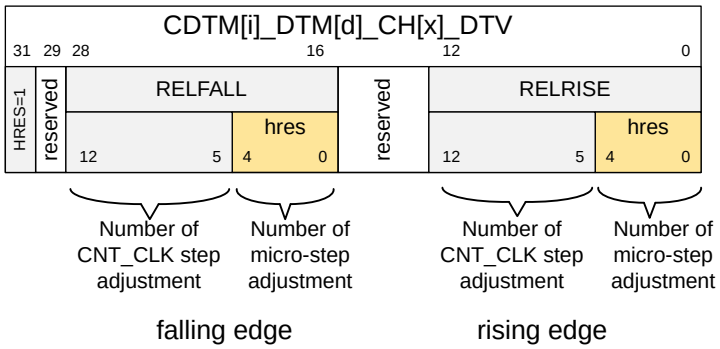
- *DTM_OUT0_HRES_[x]* = *DTM_IN_HRES[x]*
- *DTM_OUT1_HRES_[x]* = *DTM_IN_HRES[x]* + **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** [4:0]

In this case the bit field **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** and **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** of configuration now will represent the number of micro-steps as shown in Figure *130 "Configuration Register CDTM[i]_DTM[d]_CH[x]_DTV Layout for Activated HRES Support"* .

Attention:

To prevent the wraparound of the calculated *DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* result, the configured values of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** and **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** may not exceed 0x1FE0.

Figure 130 Configuration Register CDTM[i]_DTM[d]_CH[x]_DTV Layout for Activated HRES Support



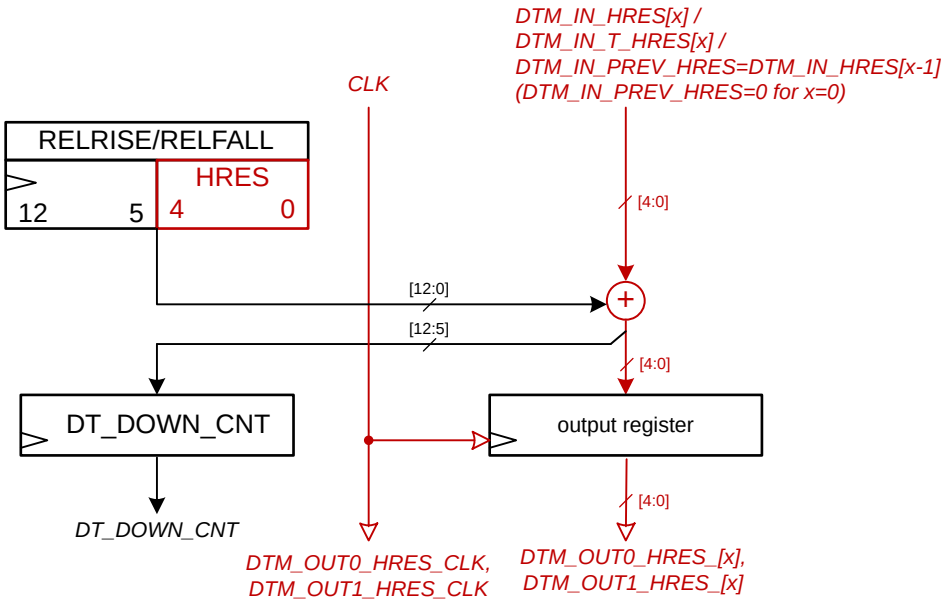
Note:

If the high resolution PWM support is activated by setting configuration bit **CDTM[i]_DTM[d]_CH[x]_DTV.HRES** , it is necessary to configure clock *CLK* as clock resolution cluster. This is done by setting configuration bit field **CDTM[i]_DTM[d]_CTRL.CLK_SEL** to 0b00.

Figure *131 "Calculation of HRES Output Signal"* shows the principle schematic view of the calculation of the output values of *DTM_OUT0_HRES_[x]* / *DTM_OUT1_HRES_[x]* .

As shown in figure, the input value of *DTM_IN_HRES[0]* [4:0] is added to **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** [12:0] / **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** [12:0] value. The output signals *DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* are the bits [4:0] of the result. The counter DT_DOWN_CNT gets the bits [12:5] of the result and the DT_DOWN_CNT result (counter output signal is *DT_DOWN_CNT*) includes the value of the carry bit or adding the lower bits [4:0].

Figure 131 Calculation of HRES Output Signal



Note:

In the figure above all three possible input signals are shown, but only one of them is selected corresponding to the selected DTM input signal (*DTM_IN_PREV* , *DTM_IN* , *DTM_IN_T*).

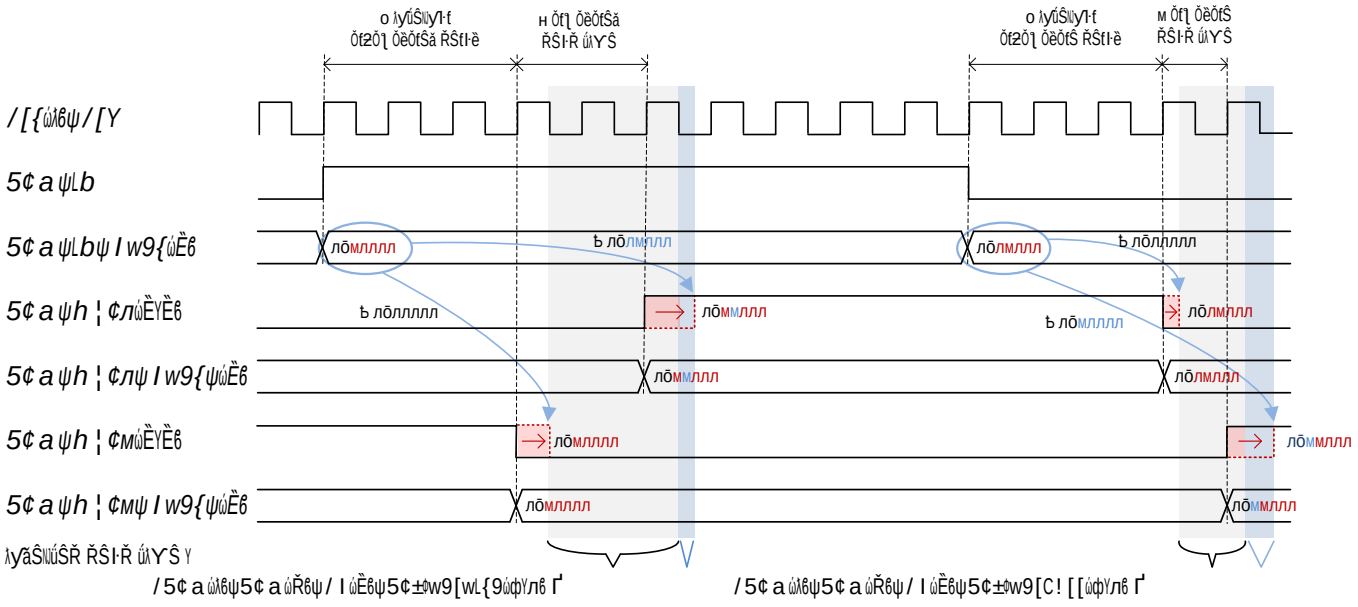
It has to be considered that the maximum range of dead time insertion is reduced to 2^8 clock cycles when high resolution support is switched on in DTM. Without high resolution support the maximum range is 2^13 clock cycles based on DTM internal clock resolution, see figure *115 "DTM Overview"* .

Note:

The clock cycles or clock frequency depends on the DTM internal clock resolution *CLK_RES* ; see Figure *115 "DTM Overview"* .

The signal diagram in Figure *132 "Standard Dead Time Adjustment on Micro-Step Level"* shows an example of how the dead time adjustment on micro-step level works. The red and blue delays are realized by customer logic outside of the DTM/GTM. The red dotted lines in Figure *132 "Standard Dead Time Adjustment on Micro-Step Level"* are the PWM signals after the external delay chain.

Figure 132 Standard Dead Time Adjustment on Micro-Step Level



In Figure *132 "Standard Dead Time Adjustment on Micro-Step Level"* it can be seen that for the rising edge of input signal *DTM_IN* [0:0] the value of the output signal *DTM_OUT1_HRES_[0]* is the same as the value of input signal *DTM_IN_HRES[0]* whereas the value of *DTM_OUT0_HRES_[0]* is calculated by adding the value of bit field **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** [4:0] to *DTM_IN_HRES[0]* .

For the falling edge of input signal *DTM_IN* [0:0] the same is done in opposite way whereas the value of the output signal *DTM_OUT0_HRES_[0]* has the same value as of the input signal *DTM_IN_HRES[0]* and the value of *DTM_OUT1_HRES_[0]* is calculated by adding the value of bit field **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** [4:0] to *DTM_IN_HRES[0]* .

The Figure *132 "Standard Dead Time Adjustment on Micro-Step Level"* shows the input signals *DTM_IN* [x:x] and *DTM_IN_HRES[x]* [4:0], but the same applies if the input signals *DTM_IN_T* [x:x], *DTM_IN_T_HRES[x]* [4:0] or *DTM_IN_PREV* [x:x], *DTM_IN_PREV_HRES[x]* [4:0] are selected.

The selection of the DTM input signals *DTM_IN* , *DTM_IN_T* and *DTM_IN_PREV* are shown in *119 "DTM Channel Overview"* . The selection of *DTM_IN* or *DTM_IN_T* is controlled by **CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_0** , **CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_1** , **CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_2** , **CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_3** . To select *DTM_IN_PREV* the signals *PSU_SHIFT* or *CH_OUT_I* and the bit fields **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_0** , **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_1** , **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_2** , **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_3** and **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_1** , **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_2** , **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_3** have to be considered. Accordingly, the selection of the corresponding high resolution input is done automatically.

The high resolution mode in DTM is only supported for standard dead time (as described before) and for the phase shift mode, where standard dead time is used.

In all other DTM modes (cross channel dead time, multiple output signal combination, pulse generation on edge) the high resolution PWM is not supported. In this case, the output signals *DTM_OUT0_HRES_[x]* , *DTM_OUT1_HRES_[x]* are overwritten to zero. Setting the configuration bit **CDTM[i]_DTM[d]_CH[x]_DTV.HRES** has no effect in this DTM mode.

Table 46 High Resolution PWM Support Depending on DTM Modes

DTM mode	high resolution PWM support	comment
standard dead time	yes	*DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* output calculated
phase shift mode	yes	*DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* output calculated
cross channel dead time	no	*DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]* outputs overwritten to zero

DTM mode	high resolution PWM support	comment
multiple output signal combination	no	high resolution outputs connected to corresponding high resolution input (see *16.8.1 "Special Configurations, Paths or Behavior"* , too)
pulse generation on edge	no	high resolution outputs connected to corresponding high resolution input

16.8.1 Special Configurations, Paths or Behavior

If special function on output 1 (*DTM_OUT1*) is selected (**CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.-DT1_0** =1), the following high resolution input is connected to high resolution output depending on **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** :

- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** =0b00: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** =0b01: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** =0b10: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** =0b11: *DTM_IN_T_HRES[0]*

If the DTM output signals are swapped (**CDTM[i]_DTM[d]_CH_CTRL1.SWAP_0** =1), then the high resolution output signals are also swapped.

Further, there is an exception if **CDTM[i]_DTM[d]_CH_CTRL2.OC0_0** =1 or **CDTM[i]_DTM[d]_CH_CTRL2.OC1_0** =1 then the corresponding high resolution output value is set to 0x0.

Note:

This ensures that the result of an active shutoff is available at an output signal as soon as possible.

If special function on output 1 (*DTM_OUT1*) is selected (**CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_1** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.-DT1_1** =1), the following high resolution input is connected to high resolution output depending on **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** :

- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** =0b00: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** =0b01: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** =0b10: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** =0b11: *DTM_IN_T_HRES[1]*

If the DTM output signals are swapped (**CDTM[i]_DTM[d]_CH_CTRL1.SWAP_1** =0), then the high resolution output signals are also swapped.

Further, there is an exception if **CDTM[i]_DTM[d]_CH_CTRL2.OC0_1** =1 or **CDTM[i]_DTM[d]_CH_CTRL2.OC1_1** =1 then the corresponding high resolution output value is set to 0x0.

Note:

This ensures that the result of an active shutoff is available at an output signal as soon as possible.

If special function on output 1 (*DTM_OUT1*) is selected (**CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_2** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.-DT1_2** =1), the following high resolution input is connected to high resolution output depending on **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** :

- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** =0b00: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** =0b01: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** =0b10: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** =0b11: *DTM_IN_T_HRES[2]*

If the DTM output signals are swapped (**CDTM[i]_DTM[d]_CH_CTRL1.SWAP_2** =0, then the high resolution output signals are also swapped.

Further, there is an exception if **CDTM[i]_DTM[d]_CH_CTRL2.OC0_2** =1 or **CDTM[i]_DTM[d]_CH_CTRL2.OC1_2** =1 then the corresponding high resolution output value is set to 0x0.

Note:

This ensures that the result of an active shutoff is available at an output signal as soon as possible.

If special function on output 1 (*DTM_OUT1*) is selected (**CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_3** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.-DT1_3** =1), the following high resolution input is connected to high resolution output depending on **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** :

- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** =0b00: 0x00

- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** =0b01: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** =0b10: 0x00
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** =0b11: *DTM_IN_T_HRES[3]*

If the DTM output signals are swapped (**CDTM[i]_DTM[d]_CH_CTRL1.SWAP_3** =0), then the high resolution output signals are also swapped.

Further, there is an exception if **CDTM[i]_DTM[d]_CH_CTRL2.OC0_3** =1 or **CDTM[i]_DTM[d]_CH_CTRL2.OC1_3** =1 then the corresponding high resolution output value is set to 0x0.

Note:

This ensures that the result of an active shutoff is available at an output signal as soon as possible.

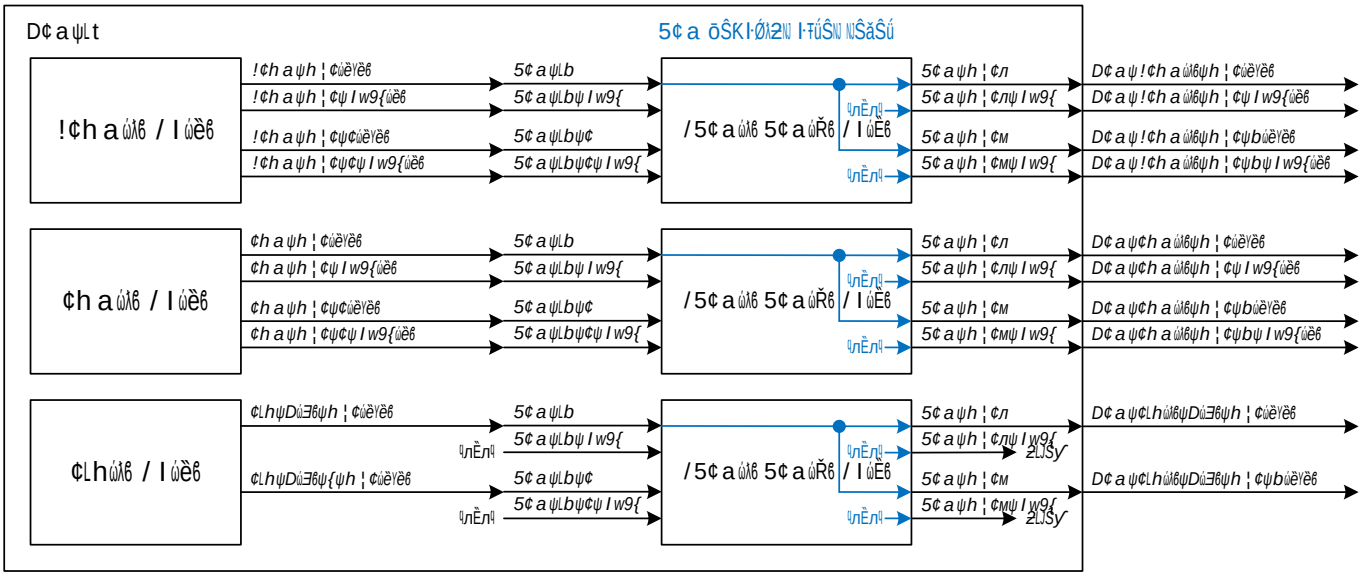
16.9 DTM Connections on GTM-IP Top Level

If DTM instance is present, it is placed behind the outputs of a TOM, ATOM or TIO depending on the GTM-IP device configuration.
In case DTM exists, the outputs *TOM_OUT* [y:y], *ATOM_OUT* [y:y] and *TOM_OUT_T* [y:y], *ATOM_OUT_T* [y:y], *TIO_G[0]_OUT* [y:y], *TIO_G[1]_OUT* [y:y], *TIO_G[2]_OUT* [y:y] and *TIO_G[0]_S_OUT* [y:y], *TIO_G[1]_S_OUT* [y:y], *TIO_G[2]_S_OUT* [y:y] are connected to DTM inputs DTM_IN and DTM_IN_T. The outputs of the DTM are routed directly to the top level of GTM-IP.
The behavior of DTM after reset is shown in Figure *133 "DTM Behavior After Reset"* .
For a DTM behind a TIO all high resolution inputs of DTM (*DTM_IN_HRES[x]* , *DTM_IN_T_HRES[x]*) are clamped to zero.

Note:

If a device configuration is used without high resolution support (device configuration parameter DTM_HIGH_RES_G=0) then all high resolution outputs of DTM (*DTM_OUT0_HRES_[x]* and *DTM_OUT1_HRES_[x]*) are zero.

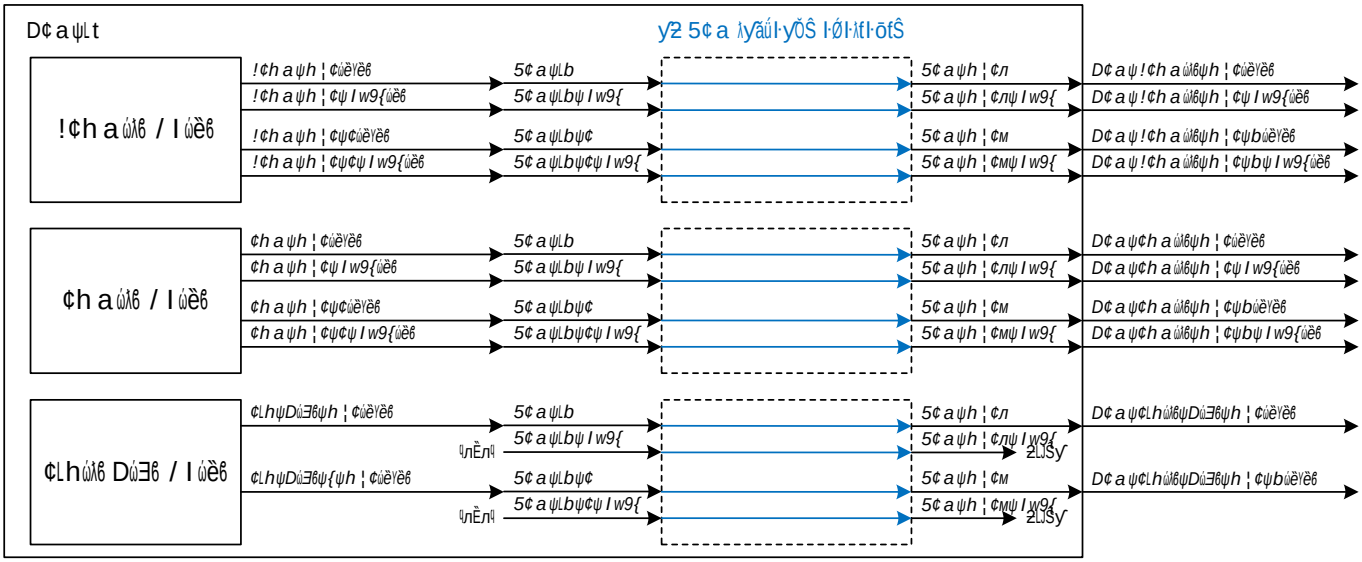
Figure 133 DTM Behavior After Reset



- To enable the second pass through route of signal *DTM_IN_T* [x:x] to the DTM output *DTM_OUT1* [x:x], the following DTM channel configuration has to be chosen:
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0** = 0b11, **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_0** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_0** =1.
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1** = 0b11, **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_1** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_1** =1.
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2** = 0b11, **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_2** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_2** =1.
- **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3** = 0b11, **CDTM[i]_DTM[d]_CH_CTRL1.O1SEL_3** =1 and **CDTM[i]_DTM[d]_CH_CTRL2.DT1_3** =1.

The signal names and the signal routing in the case of no DTM instance placed behind a TOM, ATOM or TIO, is shown in figure *134 "(A)TOM and TIO Output Signal Routing in Case of No DTM Instance Available"* .

Figure 134 (A)TOM and TIO Output Signal Routing in Case of No DTM Instance Available



Note:

The figures above *133 "DTM Behavior After Reset"* and *134 "(A)TOM and TIO Output Signal Routing in Case of No DTM Instance Available"* show only the routing without implemented register stages.

16.10 DTM Configuration Registers Description

16.10.1 CDTM[i]_DTM[d]_CTRL

Description	CDTM[i]_DTM[d] global configuration and control register
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	
Storage Type	REGISTER
Clock Active Cond	CDTM[i]_DTM[d]_CLK_ENABLE == 1

Interface: CPU

Name	CDTM[i]_DTM[d]_CTRL
Address	$0x20000 * i + 0x40 * d + 0x4400$
C-Name	GTM.CLS[i].CDTM.DTM[d].CTRL

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CTRL
Address	$0x40 * d + 0x4400$
C-Name	

CLK_SEL	
Description	Clock resolution selection
Loop	-
Bit Range	[1 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

CLK_SEL	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	d < 4 0b00 : *CLS[i]_CLK* resolution selected (cluster clock) 0b01 : *CCM[i]_CLK_RES* [0:0] resolution in use, selected input *DTM_CLK_RES* [0:0] 0b10 : *CCM[i]_FXCLK_RES* [0:0] resolution in use, selected input *DTM_CLK_RES* [1:1](DTM is connected to TOM) 0b11 : *CCM[i]_FXCLK_RES* [1:1] resolution in use, selected input *DTM_CLK_RES* [2:2] (DTM is connected to TOM)
RW-Coding	d > 3 && d <= 11 0b00 : *CLS[i]_CLK* resolution selected (cluster clock) 0b01 : *CCM[i]_CLK_RES* [0:0] resolution in use, selected input *DTM_CLK_RES* [0:0] 0b10 : *CCM[i]_CLK_RES* [1:1] resolution in use, selected input *DTM_CLK_RES* [1:1] (DTM is connected to ATOM or TIO) 0b11 : *CCM[i]_CLK_RES* [2:2] resolution in use, selected input *DTM_CLK_RES* [2:2] (DTM is connected to ATOM or TIO)

DTM_SEL	
Description	Select DTM update and PSU_SHUT_OFF reset signal
Loop	–
Bit Range	[3 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : select falling edge on DTM[d] channel 0 input 0b01 : select rising edge on DTM[d] channel 0 input 0b10 : select falling edge on DTM[d-1] channel 0 input 0b11 : select rising edge on DTM[d-1] channel 0 input
	On value 0b00 and 0b01 shutoff is executed by signal *TIM_CH_IN0* , *TIM_CH_IN1* or *DTM_AUX_IN* . On value 0b10 and 0b11 shutoff is executed by signal *DTM_PSU_IN* .

UPD_MODE	
Description	Update mode
Loop	–
Bit Range	[6 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

UPD_MODE	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : asynchronous update – **CDTM[i]_DTM[d]_CH_CTRL2_SR** not used for update of **CDTM[i]_DTM[d]_CH_CTRL2** 0b001 : shutoff release by writing 1 to bit **CDTM[i]_DTM[d]_CTRL.SHUT_OFF_RST** 0b010 : shutoff release by signal *DTM_PREV_IN0_REDGE* or *DTM_PREV_IN0_FEDGE* or *DTM_IN_REDGE* or *DTM_IN_FEDGE* defined by bit field **CDTM[i]_DTM[d]_CTRL.DTM_SEL** 0b011 : shutoff release by shutoff signal *PSU_SHUT_OFF* (defined by bits **CDTM[i]_DTM[d]_PS_CTRL.PSU_IN_SEL** and **CDTM[i]_DTM[d]_PS_CTRL.IN_POL** and **CDTM[i]_DTM[d]_CTRL.DTM_SEL** [2:2]) 0b100 : Signal *IN_EDGE* [0:0] (from channel 0) used to trigger update of **CDTM[i]_DTM[d]_CH_CTRL2** with content of **CDTM[i]_DTM[d]_CH_CTRL2_SR** 0b101 : Signal *IN_EDGE* [1:1] (from channel 1) used to trigger update of **CDTM[i]_DTM[d]_CH_CTRL2** with content of **CDTM[i]_DTM[d]_CH_CTRL2_SR** 0b110 : Signal *IN_EDGE* [2:2] (from channel 2) used to trigger update of **CDTM[i]_DTM[d]_CH_CTRL2** with content of **CDTM[i]_DTM[d]_CH_CTRL2_SR** 0b11 : Signal *IN_EDGE* [3:3] (from channel 3) used to trigger update of **CDTM[i]_DTM[d]_CH_CTRL2** with content of **CDTM[i]_DTM[d]_CH_CTRL2_SR**
	Note: If an INx_edge is not implemented the value is unused.

CH_SHUTOFF_EN	
Description	Individual shutoff feature enable
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : individual shutoff feature disabled 1 : individual shutoff feature enabled

SR_UPD_EN	
Description	Shadow register update enable
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

SR_UPD_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : no update of **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_0_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_1_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_2_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_3_SR** bits, **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_0_SR** bits, **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_1_SR** bits, **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_2_SR** bits and **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_3_SR** bits. 1 : update of **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_0_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_1_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_2_SR** , **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_3_SR** bits, **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_0_SR** bits, **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_1_SR** bits, **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_2_SR** bits and **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_3_SR** bits on trigger.

SHUT_OFF_RST	
Description	Shut off reset
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No action
W–Coding	0 : No action 1 : shutoff reset
	Note: Writing a 0b1 releases shutoff (resets signal *SHUTOFF_SYNC_0* if selected by **CDTM[i]_DTM[d]_CTRL.UPD_MODE** [2:0]=0b001)

16.10.2 CDTM[i]_DTM[d]_CH_CTRL1

Description	CDTM[i]_DTM[d] channel control register 1
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CDTM}[i]\_\text{DTM}[d]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CDTM[i]_DTM[d]_CH_CTRL1
Address	$0x20000 * i + 0x40 * d + 0x4404$
C–Name	GTM.CLS[i].CDTM.DTM[d].CH_CTRL1

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CH_CTRL1
Address	$0x40 * d + 0x4404$
C-Name	

O1SEL_0	
Description	Output 1 select channel 0
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : inverse dead time signal selected 1 : special function on output 1 selected (defined by **CDTM[i]_DTM[d]_CH_CTRL1.O1F_0**)

I1SEL_0	
Description	Input 1 select channel 0
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{mod}(d, 2) == 1$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : signal *PSU_SHIFT* [0:0] selected 1 : signal *DTM_COUT3_I* from DTM[d-1] selected
	Note: If d is even, **CDTM[i]_DTM[d]_CH_CTRL1.I1SEL_0** is not implemented.

SWAP_0	
Description	Swap outputs DTM_OUT0[0:0] of instance d and DTM_OUT1[0:0] of instance d (before final output register)
Loop	-
Bit Range	$[3:3]$
Access Type	RW
Volatile	False

SWAP_0	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : outputs not swapped 1 : swap outputs *DTM_OUT0* [0:0] of instance d and *DTM_OUT1* [0:0] of instance d

O1F_0	
Description	Output 1 function channel 0
Loop	–
Bit Range	[5 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b00 : Signal *EDGE_TRIGG_[0]* of channel [x=0] is selected 0b01 : XOR of signal *CII[x]_MUX* of channel [x=0] of instance [d] and signal *PSU_SHIFT* [0:0] / *DTM_COUT3_I* 0b10 : AND of signal *CII[x]_MUX* of channel [x=0] of instance [d] and signal *PSU_SHIFT* [0:0] / *DTM_COUT3_I* 0b11 : *DTM_IN_T* [0:0] of instance d selected

XDT_EN_0_1	
Description	Cross dead time enable on channel 0 and 1
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : cross dead time disabled on channel 0 and 1 1 : cross dead time enabled on channel 0 and 1

XDT_EN_0_1	
	Note: When a '1' is written to bit **CDTM[i]_DTM[d]_CH_CTRL1.XDT_EN_0_1** , the internal register *IN[x]_DLY1* , *IN[x]_DLY2* and the counter DT_DOWN_CNT are reset to '0' (x∈{0, 1}) *119 "DTM Channel Overview"* **Note:** **CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_0** and **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_1** must be '0' for using cross dead time to avoid wrong input signals. **CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_1** and **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_1** must be '0' for using cross dead time to avoid wrong input signals.

O1SEL_1	
Description	Output 1 select channel 1
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : inverse dead time signal selected 1 : special function on output 1 selected (defined by **CDTM[i]_DTM[d]_CH_CTRL1.O1F_1**)

I1SEL_1	
Description	Input 1 select channel 1
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : signal *PSU_SHIFT* [1:1] selected 1 : signal *COUT* [0:0] selected

SH_EN_1	
Description	Shift enable channel 1
Loop	–
Bit Range	[10 : 10]
Access Type	RW

SH_EN_1	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : switch between DTM inputs of channel [x] and DTM_IN_PREV by PSU_SHIFT or CH_OUT_I (depends to **C-DTM[i]_DTM[d]_CH_CTRL1.I1SEL_1**) is disabled 1 : switch between DTM inputs of channel [x] and DTM_IN_PREV by PSU_SHIFT or CH_OUT_I (depends to **C-DTM[i]_DTM[d]_CH_CTRL1.I1SEL_1**) is enabled

SWAP_1	
Description	Swap outputs DTM_OUT0[1:1] of instance d and DTM_OUT1[1:1] of instance d (before final output register)
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : outputs not swapped 1 : swap outputs *DTM_OUT0* [1:1] of instance d and *DTM_OUT1* [1:1] of instance d

O1F_1	
Description	Output 1 function channel 1
Loop	–
Bit Range	[13 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Signal *EDGE_TRIGG_[x]* of channel [x=1] is selected 0b01 : XOR of signal *CII[x]_MUX* of channel [x=1] of instance [d] and signal *PSU_SHIFT* [1:1] / *COUT* [0:0] 0b10 : AND of signal *CII[x]_MUX* of channel [x=1] of instance [d] and signal *PSU_SHIFT* [1:1] / *COUT* [0:0] 0b11 : *DTM_IN_T* [1:1] of instance d selected

O1SEL_2	
Description	Output 1 select channel 2
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : inverse dead time signal selected 1 : special function on output 1 selected (defined by **CDTM[i]_DTM[d]_CH_CTRL1.O1F_2**)

I1SEL_2	
Description	Input 1 select channel 2
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : signal *PSU_SHIFT* [2:2] selected 1 : signal *COUT* [1:1] selected

SH_EN_2	
Description	Shift enable channel 2
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

SH_EN_2	
RW-Coding	0 : switch between DTM inputs of channel [x] and DTM_IN_PREV by PSU_SHIFT or CH_OUT_I (depends to **C-DTM[i]_DTM[d]_CH_CTRL1.I1SEL_2**) is disabled 1 : switch between DTM inputs of channel [x] and DTM_IN_PREV by PSU_SHIFT or CH_OUT_I (depends to **C-DTM[i]_DTM[d]_CH_CTRL1.I1SEL_2**) is enabled

SWAP_2	
Description	Swap outputs DTM_OUT0[2:2] of instance d and DTM_OUT1[2:2] of instance d (before final output register)
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : outputs not swapped 1 : swap outputs *DTM_OUT0* [2:2] of instance d and *DTM_OUT1* [2:2] of instance d

O1F_2	
Description	Output 1 function channel 2
Loop	–
Bit Range	[21 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Signal *EDGE_TRIGG_[x]* of channel [x=2] is selected 0b01 : XOR of signal *CII[x]_MUX* of channel [x=2] of instance [d] and signal *PSU_SHIFT* [2:2] / *COUT* [1:1] 0b10 : AND of signal *CII[x]_MUX* of channel [x=2] of instance [d] and signal *PSU_SHIFT* [2:2] / *COUT* [1:1] 0b11 : *DTM_IN_T* [2:2] of instance d selected

XDT_EN_2_3	
Description	Cross dead time enable on channel 2 and 3
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False

XDT_EN_2_3	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : cross dead time disabled on channel 2 and 3 1 : cross dead time enabled on channel 2 and 3
	Note: When a '1' is written to bit **CDTM[i]_DTM[d]_CH_CTRL1.XDT_EN_2_3** , the internal register *IN[x]_DLY1* , *IN[x]_DLY2* and DT_DOWN_CNT are reset to '0' (x∈{2, 3}) **Note:** **CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_2** , **CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_3** , **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_2** and **CDTM[i]_DTM[d]_CH_CTRL1.SH_EN_3** must be '0' for using cross dead time to avoid wrong input signals.

O1SEL_3	
Description	Output 1 select channel 3
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : inverse dead time signal selected 1 : special function on output 1 selected (defined by **CDTM[i]_DTM[d]_CH_CTRL1.O1F_3**)

I1SEL_3	
Description	Input 1 select channel 3
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

I1SEL_3	
RW-Coding	0 : signal *PSU_SHIFT* [3:3] selected 1 : signal *COUT* [2:2] selected

SH_EN_3	
Description	Shift enable channel 3
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : switch between DTM inputs of channel [x] and DTM_IN_PREV by PSU_SHIFT or CH_OUT_I (depends to **C-DTM[i]_DTM[d]_CH_CTRL1.I1SEL_3**) is disabled 1 : switch between DTM inputs of channel [x] and DTM_IN_PREV by PSU_SHIFT or CH_OUT_I (depends to **C-DTM[i]_DTM[d]_CH_CTRL1.I1SEL_3**) is enabled

SWAP_3	
Description	Swap outputs DTM_OUT0[3:3] of instance d and DTM_OUT1[3:3] of instance d (before final output register)
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : outputs not swapped 1 : swap outputs *DTM_OUT0* [3:3] of instance d and *DTM_OUT1* [3:3] of instance d

O1F_3	
Description	Output 1 function channel 3
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

O1F_3	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b00 : Signal *EDGE_TRIGG_[x]* of channel [x=3] is selected 0b01 : XOR of signal *CII[x]_MUX* of channel [x=3] of instance [d] and signal *PSU_SHIFT* [3:3] / *COUT* [2:2] 0b10 : AND of signal *CII[x]_MUX* of channel [x=3] of instance [d] and signal *PSU_SHIFT* [3:3] / *COUT* [2:2] 0b11 : *DTM_IN_T* [3:3] of instance d selected

16.10.3 CDTM[i]_DTM[d]_CH_CTRL2

Description	CDTM[i]_DTM[d] channel control register 2
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CDTM}[i]\_\text{DTM}[d]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CDTM[i]_DTM[d]_CH_CTRL2
Address	$0x20000 * i + 0x40 * d + 0x4408$
C–Name	GTM.CLS[i].CDTM.DTM[d].CH_CTRL2

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CH_CTRL2
Address	$0x40 * d + 0x4408$
C–Name	

POL0_0	
Description	Polarity on output 0 channel 0
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Output signal not inverted 1 : Output signal inverted

OC0_0	
Description	Output 0 control channel 0
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2.SL0_0**

SL0_0	
Description	Signal level on output 0 channel 0
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT0_0	
Description	Dead time path enable on output 0 channel 0
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

DT0_0	
RW-Coding	0 : direct path from *DTM_IN* [0:0] to *DTM_OUT0* [0.0] of instance d enabled 1 : dead time path enabled

POL1_0	
Description	Polarity on output 1 channel 0
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC1_0	
Description	Output 1 control channel 0
Loop	–
Bit Range	$[5:5]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2.SL1_0**

SL1_0	
Description	Signal level on output 1 channel 0
Loop	–
Bit Range	$[6:6]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

SL1_0	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT1_0	
Description	Dead time path enable on output 1 channel 0
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [0:0] to *DTM_OUT1* [0:0] of instance d enabled 1 : path for DTM functions enabled like dead time or multiple signal combinations

POL0_1	
Description	Polarity on output 0 channel 1
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC0_1	
Description	Output 0 control channel 1
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–

OC0_1	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2.SL0_1**

SL0_1	
Description	Signal level on output 0 channel 1
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT0_1	
Description	Dead time path enable on output 0 channel 1
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [1:1] to *DTM_OUT0* [1:1] of instance d enabled 1 : dead time path enabled

POL1_1	
Description	Polarity on output 1 channel 1
Loop	–
Bit Range	[12 : 12]
Access Type	RW

POL1_1	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC1_1	
Description	Output 1 control channel 1
Loop	–
Bit Range	[13 : 13]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2.SL1_1**

SL1_1	
Description	Signal level on output 1 channel 1
Loop	–
Bit Range	[14 : 14]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT1_1	
Description	Dead time path enable on output 1 channel 1
Loop	–

DT1_1	
Bit Range	[15 : 15]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [1:1] to *DTM_OUT1* [1:1] of instance d enabled 1 : path for DTM functions enabled like dead time or multiple signal combinations

POL0_2	
Description	Polarity on output 0 channel 2
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC0_2	
Description	Output 0 control channel 2
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2.SL0_2**

SL0_2	
Description	Signal level on output 0 channel 2
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT0_2	
Description	Dead time path enable on output 0 channel 2
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [2:2] to *DTM_OUT0* [2:2] of instance d enabled 1 : dead time path enabled

POL1_2	
Description	Polarity on output 1 channel 2
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

POL1_2	
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC1_2	
Description	Output 1 control channel 2
Loop	–
Bit Range	[21 : 21]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2.SL1_2**

SL1_2	
Description	Signal level on output 1 channel 2
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT1_2	
Description	Dead time path enable on output 1 channel 2
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

DT1_2	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [2:2] to *DTM_OUT1* [2:2] of instance d enabled 1 : path for DTM functions enabled like dead time or multiple signal combinations

POL0_3	
Description	Polarity on output 0 channel 3
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC0_3	
Description	Output 0 control channel 3
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2.SL0_3**

SL0_3	
Description	Signal level on output 0 channel 3
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	True
Multithread	False

SL0_3	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT0_3	
Description	Dead time path enable on output 0 channel 3
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [3:3] of instance d to *DTM_OUT0* [3:3] of instance d enabled 1 : dead time path enabled

POL1_3	
Description	Polarity on output 1 channel 3
Loop	–
Bit Range	[28 : 28]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC1_3	
Description	Output 1 control channel 3
Loop	–
Bit Range	[29 : 29]

OC1_3	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2.SL1_3**

SL1_3	
Description	Signal level on output 1 channel 3
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT1_3	
Description	Dead time path enable on output 1 channel 3
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [3:3] to *DTM_OUT1* [3:3] of instance d enabled 1 : path for DTM functions enabled like dead time or multiple signal combinations

16.10.4 CDTM[i]_DTM[d]_CH_CTRL2_SR

Description	CDTM[i] DTM[j] channel control register 2 shadow
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CDTM}[i]\_\text{DTM}[d]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CDTM[i]_DTM[d]_CH_CTRL2_SR
Address	$0x20000 * i + 0x40 * d + 0x440C$
C-Name	GTM.CLS[i].CDTM.DTM[d].CH_CTRL2_SR

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CH_CTRL2_SR
Address	$0x40 * d + 0x440C$
C-Name	

POL0_0_SR	
Description	Polarity on output 0 channel 0 shadow register
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC0_0_SR	
Description	Output 0 control channel 0 shadow register
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

OC0_0_SR	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_0_SR**

SL0_0_SR	
Description	Signal level on output 0 channel 0 shadow register
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT0_0_SR	
Description	Dead time path enable on output 0 channel 0 shadow register
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [0:0] to *DTM_OUT0* [0:0] of instance d enabled 1 : dead time path enabled

POL1_0_SR	
Description	Polarity on output 1 channel 0 shadow register
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	False
Multithread	False

POL1_0_SR	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC1_0_SR	
Description	Output 1 control channel 0 shadow register
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_0_SR**

SL1_0_SR	
Description	Signal level on output 1 channel 0 shadow register
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT1_0_SR	
Description	Dead time path enable on output 1 channel 0 shadow register
Loop	–
Bit Range	[7 : 7]

DT1_0_SR	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [0:0] to *DTM_OUT1* [0:0] of instance d enabled 1 : path for DTM functions enabled like dead time or multiple signal combinations

POL0_1_SR	
Description	Polarity on output 0 channel 1 shadow register
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC0_1_SR	
Description	Output 0 control channel 1 shadow register
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_1_SR**

SL0_1_SR	
Description	Signal level on output 0 channel 1 shadow register

SL0_1_SR	
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT0_1_SR	
Description	Dead time path enable on output 0 channel 1 shadow register
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [1:1] to *DTM_OUT0* [1:1] of instance d enabled 1 : dead time path enabled

POL1_1_SR	
Description	Polarity on output 1 channel 1 shadow register
Loop	–
Bit Range	[12 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

POL1_1_SR	
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC1_1_SR	
Description	Output 1 control channel 1 shadow register
Loop	-
Bit Range	[13 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_1_SR**

SL1_1_SR	
Description	Signal level on output 1 channel 1 shadow register
Loop	-
Bit Range	[14 : 14]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT1_1_SR	
Description	Dead time path enable on output 1 channel 1 shadow register
Loop	-
Bit Range	[15 : 15]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

DT1_1_SR	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [1:1] to *DTM_OUT1* [1:1] of instance d 1 : path for DTM functions enabled like dead time or multiple signal combinations

POL0_2_SR	
Description	Polarity on output 0 channel 2 shadow register
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC0_2_SR	
Description	Output 0 control channel 2 shadow register
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_2_SR**

SL0_2_SR	
Description	Signal level on output 0 channel 2 shadow register
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	True
Multithread	False

SL0_2_SR	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT0_2_SR	
Description	Dead time path enable on output 0 channel 2 shadow register
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [2:2] to *DTM_OUT0* [2:2] of instance d 1 : dead time path enabled

POL1_2_SR	
Description	Polarity on output 1 channel 2 shadow register
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC1_2_SR	
Description	Output 1 control channel 2 shadow register
Loop	–
Bit Range	[21 : 21]

OC1_2_SR	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_2_SR**

SL1_2_SR	
Description	Signal level on output 1 channel 2 shadow register
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT1_2_SR	
Description	Dead time path enable on output 1 channel 2 shadow register
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : direct path from *DTM_IN* [2:2] to *DTM_OUT1* [2:2] of instance d 1 : path for DTM functions enabled like dead time or multiple signal combinations

POL0_3_SR	
Description	Polarity on output 0 channel 3 shadow register

POL0_3_SR	
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC0_3_SR	
Description	Output 0 control channel 3 shadow register
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_3_SR**

SL0_3_SR	
Description	Signal level on output 0 channel 3 shadow register
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

SL0_3_SR	
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT0_3_SR	
Description	Dead time path enable on output 0 channel 3 shadow register
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [3:3] to *DTM_OUT0* [3:3] of instance d 1 : dead time path enabled

POL1_3_SR	
Description	Polarity on output 1 channel 3 shadow register
Loop	–
Bit Range	[28 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Output signal not inverted 1 : Output signal inverted

OC1_3_SR	
Description	Output 1 control channel 3 shadow register
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

OC1_3_SR	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Functional output 1 : Constant output defined by **CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_3_SR**

SL1_3_SR	
Description	Signal level on output 1 channel 3 shadow register
Loop	–
Bit Range	$[30 : 30]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : set Signal Level to 0 1 : set Signal Level to 1

DT1_3_SR	
Description	Dead time path enable on output 1 channel 3 shadow register
Loop	–
Bit Range	$[31 : 31]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : direct path from *DTM_IN* [3:3] to *DTM_OUT1* [3:3] of instance d 1 : path for DTM functions enabled like dead time or multiple signal combinations

16.10.5 CDTM[i]_DTM[d]_CH_CTRL3

Description	CDTM[i]_DTM[d] channel control register 3
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	
Storage Type	REGISTER

Clock Active Cond	CDTM[i]_DTM[d]_CLK_ENABLE == 1

Interface: CPU

Name	CDTM[i]_DTM[d]_CH_CTRL3
Address	$0x20000 * i + 0x40 * d + 0x4428$
C-Name	GTM.CLS[i].CDTM.DTM[d].CH_CTRL3

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CH_CTRL3
Address	$0x40 * d + 0x4428$
C-Name	

CII0	
Description	Combinational input invert channel 0
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : do not invert input 1 : invert input

CIS0	
Description	Combinational input select channel 0
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : select the input *DTM_IN* [0:0] or *DTM_IN_T* [0:0] of instance d (**CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_-[0]** selects one of the two input signals) 1 : select internal signal *EDGE_TRIGG_[0]*

TSEL0_0	
Description	Input selection for dead time / edge trigger generation
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : use *DTM_IN* [0:0] of instance d as input for dead time / edge trigger generation 1 : use *DTM_IN_T* [0:0] of instance d as input for dead time / edge trigger generation

TSEL1_0	
Description	Input selection combinational logic path
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : use *DTM_IN* [0:0] of instance d as input for combinational logic path 1 : use *DTM_IN_T* [0:0] of instance d as input for combinational logic path

CII1	
Description	Combinational input invert channel 1
Loop	–
Bit Range	$[8:8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

CII1	
RW-Coding	0 : do not invert input 1 : invert input

CIS1	
Description	Combinational input select channel 1
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : select the input *DTM_IN* [1:1] or *DTM_IN_T* [1:1] of instance d (**CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_[1]** selects one of the two input signals) 1 : select internal signal *EDGE_TRIGG_[1]*

TSEL0_1	
Description	Input selection for dead time / edge trigger generation
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : use *DTM_IN* [1:1] of instance d as input for dead time / edge trigger generation 1 : use *DTM_IN_T* [1:1] of instance d as input for dead time / edge trigger generation

TSEL1_1	
Description	Input selection combinational logic path
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

TSEL1_1	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : use *DTM_IN* [1:1] of instance d as input for combinational logic path 1 : use *DTM_IN_T* [1:1] of instance d as input for combinational logic path

CII2	
Description	Combinational input invert channel 2
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : do not invert input 1 : invert input

CIS2	
Description	Combinational input select channel 2
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : select the input *DTM_IN* [2:2] or *DTM_IN_T* [2:2] of instance d (**CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_[2]** selects one of the two input signals) 1 : select internal signal *EDGE_TRIGG_[2]*

TSEL0_2	
Description	Input selection for dead time / edge trigger generation
Loop	–
Bit Range	[18 : 18]
Access Type	RW

TSEL0_2	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : use *DTM_IN* [2:2] of instance d as input for dead time / edge trigger generation 1 : use *DTM_IN_T* [2:2] of instance d as input for dead time / edge trigger generation

TSEL1_2	
Description	Input selection combinational logic path
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : use *DTM_IN* [2:2] of instance d as input for combinational logic path 1 : use *DTM_IN_T* [2:2] of instance d as input for combinational logic path

CII3	
Description	Combinational input invert channel 3
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : do not invert input 1 : invert input

CIS3	
Description	Combinational input select channel 3
Loop	–

CIS3	
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : select the input *DTM_IN* [3:3] or *DTM_IN_T* [3:3] of instance d (**CDTM[i]_DTM[d]_CH_CTRL3.TSEL0_[3]** selects one of the two input signals) 1 : select internal signal *EDGE_TRIGG_[3]*

TSEL0_3	
Description	Input selection for dead time / edge trigger generation
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : use *DTM_IN* [3:3] of instance d as input for dead time / edge trigger generation 1 : use *DTM_IN_T* [3:3] of instance d as input for dead time / edge trigger generation

TSEL1_3	
Description	Input selection combinational logic path
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

TSEL1_3	
RW-Coding	0 : use *DTM_IN* [3:3] of instance d as input for combinational logic path 1 : use *DTM_IN_T* [3:3] of instance d as input for combinational logic path

16.10.6 CDTM[i]_DTM[d]_PS_CTRL

Description	CDTM[i]_DTM[d] phase shift unit configuration and control register
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CDTM}[i]\_\text{DTM}[d]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CDTM[i]_DTM[d]_PS_CTRL
Address	$0x20000 * i + 0x40 * d + 0x4410$
C-Name	GTM.CLS[i].CDTM.DTM[d].PS_CTRL

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_PS_CTRL
Address	$0x40 * d + 0x4410$
C-Name	

RELBLK	
Description	Reload value blanking window
Loop	-
Bit Range	$[9 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Note: A value of 0x000 resets counter BLK_DOWN_CNT

PSU_IN_SEL	
Description	PSU input select
Loop	-
Bit Range	$[16 : 16]$
Access Type	RW
Volatile	False

PSU_IN_SEL	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : *TIM_CH_IN0* or *TIM_CH_IN1* selected 1 : *DTM_AUX_IN* selected

IN_POL	
Description	Input polarity
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : input signal is not inverted 1 : input signal is inverted

TIM_SEL	
Description	TIM input select
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : select *TIM_CH_IN0* 1 : select *TIM_CH_IN1*

SHIFT_SEL	
Description	Shift select
Loop	–

SHIFT_SEL	
Bit Range	$[21 : 20]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b00 : trigger signal for reloading DOWN COUNTER is *IN_EDGE* [0:0] of DTM channel [0]; signal *TIM_CH_IN0* or *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] or *DTM_PSU_IN* via signal *EXT_CTRL_GATED* is connected via signal *PSU_SHIFT* [1:1] to DTM channel 1 0b01 : trigger signal for reloading DOWN COUNTER is *IN_EDGE* [1:1] of DTM channel [1]; signal *TIM_CH_IN0* or *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] or *DTM_PSU_IN* via signal *EXT_CTRL_GATED* is connected via signal *PSU_SHIFT* [2:2] to DTM channel 2 0b10 : trigger signal for reloading DOWN COUNTER is *IN_EDGE* [2:2] of DTM channel [2]; signal *TIM_CH_IN0* or *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] or *DTM_PSU_IN* via signal *EXT_CTRL_GATED* is connected via signal *PSU_SHIFT* [3:3] to DTM channel 3 0b11 : no loading of DOWN COUNTER (blanking window feature deactivated); signal *TIM_CH_IN0* or *TIM_CH_IN1* or *DTM_AUX_IN* [0:0] or *DTM_PSU_IN* via signal *EXT_CTRL_GATED* is connected via signal *PSU_SHIFT* [0:0] to DTM channel 0
	Note: If a channel is not implemented the value is unused. A write with this unused value returns "10" on status.

16.10.7 CDTM[i]_DTM[d]_CH[x]_DTV

Description	CDTM[i]_DTM[d] channel [x] dead time reload values
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]; x = \{n : 0 <= n <= 3\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CDTM}[i]\_\text{DTM}[d]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CDTM[i]_DTM[d]_CH[x]_DTV
Address	$0x20000 * i + 0x40 * d + 0x4 * x + 0x4414$
C–Name	GTM.CLS[i].CDTM.DTM[d].CH_DTV[x]

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CH[x]_DTV
Address	$0x40 * d + 0x4 * x + 0x4414$
C–Name	

RELRISE	
Description	Reload value for rising edge dead time
Loop	–

RELRISE	
Bit Range	[12 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

RELFALL	
Description	Reload value for falling edge dead time
Loop	–
Bit Range	[28 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

HRES	
Description	high resolution PWM support
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$(d < 4\&\&i < \text{NTOM}\&\&d * 4 < \text{TOM\_HIGH\_RES}[i])\|\|(d >= 4\&\&i < \text{NATOM}\&\&(d-4) * 4 < \text{ATOM\_HIGH\_RES}[i])$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : high resolution PWM support off 1 : high resolution PWM support on

16.10.8 CDTM[i]_DTM[d]_CH_SR

Description	CDTM[i]_DTM[d] channel shadow register

Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CDTM}[i]\_\text{DTM}[d]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CDTM[i]_DTM[d]_CH_SR
Address	$0x20000 * i + 0x40 * d + 0x4424$
C-Name	GTM.CLS[i].CDTM.DTM[d].CH_SR

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CH_SR
Address	$0x40 * d + 0x4424$
C-Name	

SL0_0_SR_SR	
Description	Shadow register for bit CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_0_SR
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	

SL1_0_SR_SR	
Description	Shadow register for bit CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_0_SR
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

SL1_0_SR_SR	
RW-Coding	

SL0_1_SR_SR	
Description	Shadow register for bit CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_1_SR
Loop	-
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	

SL1_1_SR_SR	
Description	Shadow register for bit CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_1_SR
Loop	-
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	

SL0_2_SR_SR	
Description	Shadow register for bit CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_2_SR
Loop	-
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

SL0_2_SR_SR	
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	

SL1_2_SR_SR	
Description	Shadow register for bit CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_2_SR
Loop	-
Bit Range	$[5:5]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	

SL0_3_SR_SR	
Description	Shadow register for bit CDTM[i]_DTM[d]_CH_CTRL2_SR.SL0_3_SR
Loop	-
Bit Range	$[6:6]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	

SL1_3_SR_SR	
Description	Shadow register for bit CDTM[i]_DTM[d]_CH_CTRL2_SR.SL1_3_SR
Loop	-
Bit Range	$[7:7]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

SL1_3_SR_SR	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	

16.10.9 CDTM[i]_DTM[d]_CTRL2

Description	CDTM[i]_DTM[d] global configuration and control register 2
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CDTM}[i]\_\text{DTM}[d]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CDTM[i]_DTM[d]_CTRL2
Address	$0x20000 * i + 0x40 * d + 0x442C$
C-Name	GTM.CLS[i].CDTM.DTM[d].CTRL2

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CTRL2
Address	$0x40 * d + 0x442C$
C-Name	

SHUTOFF_SEL_0	
Description	Channel 0: Select input signal to be used as shutoff signal.
Loop	–
Bit Range	$[2 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 7, 7) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : *TIM_CH_IN0* selected 0b001 : *TIM_CH_IN1* selected 0b010 : *DTM_AUX_IN* [2:2] of instance d (*DTM_AUX_IN* [0:0] of instance d-1) selected 0b011 : *DTM_AUX_IN* [3:3] of instance d (*DTM_AUX_IN* [1:1] of instance d-1) selected 0b100 : *DTM_AUX_IN* [0:0] of instance d selected 0b101 : *DTM_AUX_IN* [1:1] of instance d selected 0b110 : '0' selected 0b111 : '1' selected

SHUTOFF_SEL_0	
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_0** = 0.

SHUTOFF_POL_0	
Description	Channel 0: Configure if the selected shutoff input signal used as shutoff output signal is inverted or not.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 7, 7) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b0 : inverter disabled 0b1 : inverter enabled
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_0** = 0.

UPD_MODE_0	
Description	Channel 0: Control the update mode of the internal SHUTOFF_SYNC_0 signal.
Loop	–
Bit Range	[5 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 7, 7) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b00 : internal signal *SHUTOFF_SYNC_0* is cleared; a set is not possible 0b01 : signal *PSU_SHUT_OFF* [0:0] sets internal signal *SHUTOFF_SYNC_0* ; internal signal *SHUTOFF_SYNC_0* is cleared depending on bit field **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_0** 0b10 : signal *PSU_SHUT_OFF* [0:0] sets internal signal *SHUTOFF_SYNC_0* ; internal signal *SHUTOFF_SYNC_0* is cleared by signal *DTM_PREV_IN0_REDGE* or *DTM_PREV_IN0_FEDGE* or *DTM_IN_REDGE* of channel 0 or *DTM_IN_FEDGE* of channel 0 defined by bit field **CDTM[i]_DTM[d]_CTRL.DTM_SEL** 0b11 : signal *PSU_SHUT_OFF* [0:0] =1 sets internal signal *SHUTOFF_SYNC_0* ; signal *PSU_SHUT_OFF* [0:0] =0 clears internal signal *SHUTOFF_SYNC_0*

UPD_MODE_0	
	Note: The reset of *SHUTOFF_SYNC_0* has lower priority than the set. **Note:** This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_0** = 0.

SHUT_OFF_RST_0	
Description	Channel 0: Clear of internal signal SHUTOFF_SYNC_0 if selected as control source.
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 7, 7) == 0
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No action
W-Coding	0 : No action 1 : shutoff reset (clear of internal signal *SHUTOFF_SYNC_0* enabled)
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_0** = 0.

WR_EN_0	
Description	Channel 0: Write enable of Bitfields
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no action
W-Coding	0 : writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_0** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_0** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_0** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_0** disabled 1 : writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_0** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_0** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_0** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_0** enabled
	Note: Write enable bit field always read as zero, because no storage element implemented.

SHUTOFF_SEL_1	
Description	Channel 1: Select input signal to be used as shutoff signal.
Loop	–
Bit Range	[10 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 15, 15) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : *TIM_CH_IN0* selected 0b001 : *TIM_CH_IN1* selected 0b010 : *DTM_AUX_IN* [2:2] of DTM instance d (*DTM_AUX_IN* [0:0] of DTM instance d−1) selected 0b011 : *DTM_AUX_IN* [3:3] of DTM instance d (*DTM_AUX_IN* [1:1] of DTM instance d−1) selected 0b100 : *DTM_AUX_IN* [0:0] of DTM instance d selected 0b101 : *DTM_AUX_IN* [1:1] of DTM instance d selected 0b110 : '0' selected 0b111 : '1' selected
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_1** = 0.

SHUTOFF_POL_1	
Description	Channel 1: Configure if the selected shutoff input signal used as shutoff output signal is inverted or not.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 15, 15) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b0 : inverter disabled 0b1 : inverter enabled
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_1** = 0.

UPD_MODE_1	
Description	Channel 1: Control the update mode of the internal SHUTOFF_SYNC_1 signal.
Loop	–
Bit Range	[13 : 12]

UPD_MODE_1	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 15, 15) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b00 : internal signal *SHUTOFF_SYNC_1* is cleared; a set is not possible 0b01 : signal *PSU_SHUT_OFF* [1:1] sets internal signal *SHUTOFF_SYNC_1* ; internal signal *SHUTOFF_SYNC_1* is cleared depending to bit field **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_1** 0b10 : signal *PSU_SHUT_OFF* [1:1] sets internal signal *SHUTOFF_SYNC_1* ; internal signal *SHUTOFF_SYNC_1* is cleared by signal *DTM_PREV_IN0_REDGE* or *DTM_PREV_IN0_FEDGE* or *DTM_IN_REDGE* of channel 0 or *DTM_IN_FEDGE* of channel 0 defined by bit field **CDTM[i]_DTM[d]_CTRL.DTM_SEL** 0b11 : signal *PSU_SHUT_OFF* [1:1]=1 sets internal signal *SHUTOFF_SYNC_1* ; signal *PSU_SHUT_OFF* [1:1]=0 clears internal signal *SHUTOFF_SYNC_1*
	Note: The reset of *SHUTOFF_SYNC_1* has lower priority than the set. **Note:** This bit field is write protected by bit CDTM[i]_DTM[d]_CTRL2.WR_EN_1 = 0.

SHUT_OFF_RST_1	
Description	Channel 1: Clear of internal signal SHUTOFF_SYNC_1 if selected as control source.
Loop	–
Bit Range	[14 : 14]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 15, 15) == 0
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No action
W–Coding	0 : No action 1 : shutoff reset (clear of internal signal *SHUTOFF_SYNC_1* enabled)
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_1** = 0.

WR_EN_1	
Description	Channel 1: Write enable of Bitfields
Loop	–
Bit Range	[15 : 15]
Access Type	RW

WR_EN_1	
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no action
W-Coding	0 : writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_1** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_1** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_1** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_1** disabled 1 : writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_1** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_1** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_1** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_1** enabled
	Writing a 0b1 enables the writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_1** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_1** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_1** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_1** . **Note:** Write enable bit field always read as zero, because no storage element implemented.

SHUTOFF_SEL_2	
Description	Channel 2: Select input signal to be used as shutoff signal.
Loop	–
Bit Range	[18 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 23, 23) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : *TIM_CH_IN0* selected 0b001 : *TIM_CH_IN1* selected 0b010 : *DTM_AUX_IN* [2:2] of DTM instance d (*DTM_AUX_IN* [0:0] of DTM instance d-1) selected 0b011 : *DTM_AUX_IN* [3:3] of DTM instance d (*DTM_AUX_IN* [1:1] of DTM instance d-1) selected 0b100 : *DTM_AUX_IN* [0:0] of DTM instance d selected 0b101 : *DTM_AUX_IN* [1:1] of DTM instance d selected 0b110 : '0' selected 0b111 : '1' selected
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_2** = 0.

SHUTOFF_POL_2	
Description	Channel 2: Configure if the selected shutoff input signal used as shutoff output signal is inverted or not.
Loop	–
Bit Range	[19 : 19]

SHUTOFF_POL_2	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 23, 23) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b0 : inverter disabled 0b1 : inverter enabled
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_2** = 0.

UPD_MODE_2	
Description	Channel 2: Control the update mode of the internal SHUTOFF_SYNC_2 signal.
Loop	–
Bit Range	[21 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 23, 23) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : internal signal *SHUTOFF_SYNC_2* is cleared; a set is not possible 0b01 : signal *PSU_SHUT_OFF* [2:2] sets internal signal *SHUTOFF_SYNC_2* ; internal signal *SHUTOFF_SYNC_2* is cleared depending to bit field **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_2** 0b10 : signal *PSU_SHUT_OFF* [2:2] sets internal signal *SHUTOFF_SYNC_2* ; internal signal *SHUTOFF_SYNC_2* is cleared by signal *DTM_PREV_IN0_REDGE* or *DTM_PREV_IN0_FEDGE* or *DTM_IN_REDGE* of channel 0 or *DTM_IN_FEDGE* of channel 0 defined by bit field **CDTM[i]_DTM[d]_CTRL.DTM_SEL** 0b11 : signal *PSU_SHUT_OFF* [2:2]=1 sets internal signal *SHUTOFF_SYNC_2* ; signal *PSU_SHUT_OFF* [2:2]=0 clears internal signal *SHUTOFF_SYNC_2*
	Note: The reset of *SHUTOFF_SYNC_2* has lower priority than the set. **Note:** This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_2** = 0.

SHUT_OFF_RST_2	
Description	Channel 2: Clear of internal signal SHUTOFF_SYNC_2 if selected as control source.
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False

SHUT_OFF_RST_2	
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 23, 23) == 0
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No action
W-Coding	0 : No action 1 : shutoff reset (clear of internal signal *SHUTOFF_SYNC_2* enabled)
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_2** = 0.

WR_EN_2	
Description	Channel 2: Write enable of Bitfields
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no action
W-Coding	0 : writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_2** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_2** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_2** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_2** disabled 1 : writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_2** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_2** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_2** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_2** enabled
	Writing a 0b1 enables the writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_2** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_2** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_2** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_2** . **Note:** Write enable bit field always read as zero, because no storage element implemented.

SHUTOFF_SEL_3	
Description	Channel 3: Select input signal to be used as shutoff signal.
Loop	–
Bit Range	[26 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

SHUTOFF_SEL_3	
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 31, 31) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : *TIM_CH_IN0* selected 0b001 : *TIM_CH_IN1* selected 0b010 : *DTM_AUX_IN* [2:2] of DTM instance d (*DTM_AUX_IN* [0:0] of DTM instance d-1) selected 0b011 : *DTM_AUX_IN* [3:3] of DTM instance d (*DTM_AUX_IN* [1:1] of DTM instance d-1) selected 0b100 : *DTM_AUX_IN* [0:0] of DTM instance d selected 0b101 : *DTM_AUX_IN* [1:1] of DTM instance d selected 0b110 : '0' selected 0b111 : '1' selected
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_3** = 0.

SHUTOFF_POL_3	
Description	Channel 3: Configure if the selected shutoff input signal used as shutoff output signal is inverted or not.
Loop	-
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 31, 31) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b0 : inverter disabled 0b1 : inverter enabled
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_3** = 0.

UPD_MODE_3	
Description	Channel 3: Control the update mode of the internal SHUTOFF_SYNC_3 signal.
Loop	-
Bit Range	[29 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 31, 31) == 0

UPD_MODE_3	
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : internal signal *SHUTOFF_SYNC_3* is cleared; a set is not possible 0b01 : signal *PSU_SHUT_OFF* [3:3] sets internal signal *SHUTOFF_SYNC_3* ; internal signal *SHUTOFF_SYNC_3* is cleared depending to bit field **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_3** 0b10 : signal *PSU_SHUT_OFF* [3:3] sets internal signal *SHUTOFF_SYNC_3* ; internal signal *SHUTOFF_SYNC_3* is cleared by signal *DTM_PREV_IN0_REDGE* or *DTM_PREV_IN0_FEDGE* or *DTM_IN_REDGE* of channel 0 or *DTM_IN_FEDGE* of channel 0 defined by bit field **CDTM[i]_DTM[d]_CTRL.DTM_SEL** 0b11 : signal *PSU_SHUT_OFF* [3:3] =1 sets internal signal *SHUTOFF_SYNC_3* ; signal *PSU_SHUT_OFF* [3:3] =0 clears internal signal *SHUTOFF_SYNC_3*
	Note: The reset of *SHUTOFF_SYNC_3* has lower priority than the set. **Note:** This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_3** = 0.

SHUT_OFF_RST_3	
Description	Channel 3: Clear of internal signal SHUTOFF_SYNC_3 if selected as control source.
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, 31, 31) == 0
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No action
W-Coding	0 : No action 1 : shutoff reset (clear of internal signal *SHUTOFF_SYNC_3* enabled)
	Note: This bit field is write protected by bit **CDTM[i]_DTM[d]_CTRL2.WR_EN_3** = 0.

WR_EN_3	
Description	Channel 3: Write enable of Bitfields
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–

WR_EN_3	
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no action
W-Coding	0 : writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_3** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_3** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_3** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_3** disabled 1 : writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_3** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_3** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_3** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_3** enabled
	Writing a 0b1 enables the writing of the bit fields **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_SEL_3** , **CDTM[i]_DTM[d]_CTRL2.SHUTOFF_POL_3** , **CDTM[i]_DTM[d]_CTRL2.UPD_MODE_3** , **CDTM[i]_DTM[d]_CTRL2.SHUT_OFF_RST_3** . **Note:** Write enable bit field always read as zero, because no storage element implemented.

16.10.10 CDTM[i]_DTM[d]_CH[x]_DTV_SR

Description	CDTM[i]_DTM[d] channel [x] dead time shadow values
Loop	$i = \{n : 0 <= n <= \text{NCDTM} - 1\}; d = \text{CDTM}[i]; x = \{n : 0 <= n <= 3\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{CDTM}[i]\_\text{DTM}[d]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	CDTM[i]_DTM[d]_CH[x]_DTV_SR
Address	$0x20000 * i + 0x40 * d + 0x4 * x + 0x4430$
C-Name	GTM.CLS[i].CDTM.DTM[d].CH_DTV_SR[x]

Interface: MCS[i]

Name	CDTM[i]_DTM[d]_CH[x]_DTV_SR
Address	$0x40 * d + 0x4 * x + 0x4430$
C-Name	

RELRISE_SR	
Description	Shadow value for rising edge dead time
Loop	-
Bit Range	$[12 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

RELRISE_UPD_FE0RE1	
Description	Control if falling edge or rising edge triggers update of CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE
Loop	–
Bit Range	[14 : 14]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : update of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** triggered by falling edge 1 : update of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** triggered by rising edge

RELRISE_UPD_EN	
Description	Control bit to enable update of CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : update mechanism of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** disabled 1 : update mechanism of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** enabled

RELFALL_SR	
Description	Shadow value for falling edge dead time
Loop	–
Bit Range	[28 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

RELFALL_UPD_FE0RE1	
Description	Control if falling edge or rising edge triggers update of CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : update of **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** triggered by falling edge 1 : update of **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** triggered by rising edge

RELFALL_UPD_EN	
Description	Control bit to enable update of CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : update mechanism of **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** disabled 1 : update mechanism of **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** enabled

16.11 DTM Signal Description

16.11.1 DTM Signals

Loop	–
Condition	–
Format	–

CLK_RES	
Description	clock resolution signal in use in DTM module
Loop	–
Condition	–
Signal Type	INT
Assignment	–

COUT	
Description	signal used for multiple output signal combination feature; bit x of signal vector is output of channel x to be able to combine it with further signals of channel x+1
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CH_OUT_I	
Description	signal used for multiple output signal combination feature; input signal off a DTM channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CH_OUT_O	
Description	signal used for multiple output signal combination feature; output signal off a DTM channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

PSU_SHIFT	
Description	signal of phase shift unit to be used inside DTM channels; signal PSU_SHIFT[x:x] used by corresponding channel [x]
Loop	–
Condition	–
Signal Type	INT
Assignment	–

IN_EDGE	
Description	signal of each channel [x] to trigger a reload of counter BLK_DOWN_CNT or trigger an update of a register from a shadow register
Loop	–
Condition	–
Signal Type	INT
Assignment	–

DTM_IN_PREV	
Description	DTM channel input signal from a previous channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

DTM_IN_PREV_HRES[x]	
Description	DTM channel dtm_in high resolution signal from a previous channel
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–

DTM_IN_PREV_HRES[x]	
Signal Type	ARRAY[4]
Assignment	–

DTM_IN_FEDGE	
Description	signal of each channel [x]; only for channel [0] the signal is routed to DTM_IN0_FEDGE
Loop	–
Condition	–
Signal Type	–
Assignment	–

DTM_IN_REDGE	
Description	signal of each channel [x]; only for channel [0] the signal is routed to DTM_IN0_REDGE
Loop	–
Condition	–
Signal Type	–
Assignment	–

DTM_CH_F_IN	
Description	input signal of channel [x] used with cross channel dead time feature
Loop	–
Condition	–
Signal Type	–
Assignment	–

DTM_CH_F_OUT	
Description	output signal of channel [x] used with cross channel dead time feature
Loop	–
Condition	–
Signal Type	–
Assignment	–

DTM_IN_HRES	
Description	DTM channel dtm_in high resolution signal from (A)TOM / TIO
Loop	–
Condition	–
Signal Type	INT
Assignment	–

IN[x]_SIG	
Description	signal of channel [x] after the first two multiplexers on which the edge detection for signal IN_EDGE is done
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

EDGE_TRIGG_[x]	
Description	signal of channel [x] which shows if the selected input source has an edge

EDGE_TRIGG_[x]	
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

CII[x]_MUX	
Description	signal of channel [x] which is the multiplexer output which is in front of the XOR and AND gates for the combinational feature of the channel
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

IN[x]_DLY1	
Description	selected input source signal of channel [x] one clock cycle delayed; used for edge detection
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

IN[x]_DLY2	
Description	selected input source signal of channel [x] two clock cycle delayed; used for edge detection
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

IN[x]_RISE_PULSE	
Description	signal of DTM channel [x] that a rising edge at the input occurs
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

IN[x]_FALL_PULSE	
Description	signal of DTM channel [x] that a falling edge at the input occurs
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

IN[x]_RISE	
Description	signal of DTM channel after a dead time insertion for a rising edge

IN[x]_RISE	
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

IN[x]_FALL	
Description	signal of DTM channel after a dead time insertion for a falling edge
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Signal Type	ARRAY[4]
Assignment	–

DTM_AUX_IN0	
Description	signal used for feature phase shift or to be (N)AND/(N)OR/X(N)OR combined with further channel signals
Loop	–
Condition	–
Signal Type	INT
Assignment	–

DTM_AUX_IN1	
Description	signal used for feature phase shift or to be (N)AND/(N)OR/X(N)OR combined with further channel signals
Loop	–
Condition	–
Signal Type	INT
Assignment	–

PSU_SHUT_OFF	
Description	output signal of phase shift unit used for feature shutoff of channel x
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SHUTOFF_SYNC_0	
Description	signal used by individual channel shutoff feature of channel 0
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SHUTOFF_SYNC_1	
Description	signal used by individual channel shutoff feature of channel 1
Loop	–
Condition	–
Signal Type	INT

SHUTOFF_SYNC_1	
Assignment	–

SHUTOFF_SYNC_2	
Description	signal used by individual channel shutoff feature of channel 2
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SHUTOFF_SYNC_3	
Description	signal used by individual channel shutoff feature of channel 3
Loop	–
Condition	–
Signal Type	INT
Assignment	–

BLK_DOWN_CNT	
Description	output signal of blanking window counter DOWN COUNTER
Loop	–
Condition	–
Signal Type	–
Assignment	–

DT_DOWN_CNT	
Description	output signal of dead time counter DT_DOWN_CNT
Loop	–
Condition	–
Signal Type	–
Assignment	–

16.12 DTM Port Description

16.12.1 DTM AEI Infrastructure Interface

Loop	–
Condition	–
Format	–

AEI_DTM_SEL	
Description	AEI module select
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–

AEI_DTM_SEL	
Operational Reset	GTM_RESET
Reset Value	–

DTM_AEI_W1R0	
Description	AEI address
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_AEI_ADDR	
Description	AEI access: write = 1, read = 0
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	3 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_AEI_WDATA	
Description	AEI write data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	31 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_AEI_RDATA	
Description	AEI read data
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR

DTM_AEI_RDATA	
Port Direction	OUT
Port Range	31 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_AEI_READY	
Description	AEI ready: access completed
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_AEI_STATUS	
Description	AEI status
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

16.12.2 DTM Signal Interface

Loop	–
Condition	–
Format	–

DTM_IN	
Description	DTM input signal from (A)TOM / TIO
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	3 DOWNTO 0
Operational Clock	CLS[i]_CLK

DTM_IN	
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_IN_T	
Description	DTM input signal from (A)TOM / TIO
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	3 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_CH_IN0	
Description	DTM input signal from TIM
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_CH_IN1	
Description	DTM input signal from TIM
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_AUX_IN	
Description	DTM input signal from external
Loop	–
Condition	–
Logical Name	–

DTM_AUX_IN	
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	3 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_OUT0	
Description	DTM output 0 signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_OUT1	
Description	DTM output 1 signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

16.12.3 DTM Signal HRES Interface

Loop	–
Condition	–
Format	–

DTM_IN_HRES[x]	
Description	DTM channel x DTM_IN high resolution signal from (A)TOM / TIO
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN

DTM_IN_HRES[x]	
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_IN_T_HRES[x]	
Description	DTM channel x DTM_IN_T high resolution signal from (A)TOM / TIO
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_OUT0_HRES_CLK	
Description	DTM DTM_OUT0 high resolution output high resolution clock signals
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_OUT0_HRES_[x]	
Description	DTM channel [x] DTM_OUT0 high resolution output signal
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_OUT1_HRES_CLK	
Description	DTM DTM_OUT1 high resolution output high resolution clock signals

DTM_OUT1_HRES_CLK	
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	3 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_OUT1_HRES_[x]	
Description	DTM channel x DTM_OUT1 high resolution output signal
Loop	$x = \{n : 0 <= n <= 3\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

16.12.4 DTM / DTM Signal Interface

Loop	–
Condition	–
Format	–

DTM_COUT3_I	
Description	Signal between DTM instances; driven by previous instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_COUT3_O	
Description	Signal between DTM instances; used by following instance
Loop	–
Condition	–

DTM_COUT3_O	
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_PSU_IN	
Description	Signal between DTM instances; driven by previous instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_PSU_OUT	
Description	Signal between DTM instances; used by following instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_PREV_IN0_REDGE	
Description	Signal chain through all DTM instances; driven by previous instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_PREV_IN0_FEDGE	
Description	Signal chain through all DTM instances; driven by previous instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_IN0_REDGE	
Description	Signal chain through all DTM instances; used by following instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DTM_IN0_FEDGE	
Description	Signal chain through all DTM instances; used by following instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

17 Multi Channel Sequencer (MCS)

17.1 Overview

The Multi Channel Sequencer (MCS) is a generic data processing module that is connected to the ARU. One of its major applications is to calculate complex output sequences that may depend on the time base values of the TBU and are processed in combination with the ATOM submodule. Other applications can use the MCS to perform extended data processing of input data resulting from the TIM submodule. Moreover, some applications may process data provided by the CPU within the MCS submodule, and the calculated results are sent to the outputs using the ATOM submodules.

Table *47 "Generic Design Parameters"* summarizes all available generic design parameters of the MCS hardware structure.

Indices and their range as used inside this chapter are:

- i:={0, 1,..., NMCS−1} instance index of cluster / MCS module
- h:={0, 1,..., NHBP−1} index of hardware breakpoint logic blocks
- x:={0, 1,..., 7} channel index of a MCS−channel
- y:={0, 1,..., 7} index for general purpose register
- a: index of address range protectors

Table 47 Generic Design Parameters

Design Parameter	Description
W	Word width of the data path
T	Number of available MCS−channels
RDW	RAM data width of connected RAM
RAW	RAM address width used by the MCS for addressing memory
USR	Use second RAM port (0 − one RAM port available, 1 − two RAM ports available)
BAW	Bus Master Address Width
BDW	Bus Master Data Width
URIP	Use RAM input pipeline registers (0 − no register, 1 − use register)
UROP	Use RAM output pipeline registers (0 − no register , 1 − use register)
UDP	Use Decoder Pipeline register (0 − no register , 1 − use register)
UAP	Use ALU Pipeline register (0 − no register , 1 − use register)
NPS	Total number of pipeline stages (with NPS = 3 + URIP + UROP + UDP + UAP)

All MCS instances in the GTM use the values T=8, W=24, RDW=32, USR=1, BAW = 14, BDW = 32, URIP = 1, UROP = 1, UDP = 1, UAP = 1, and NPS = 7.
The parameter RAW can vary between the available MCS instances. RAW depends on the scalar variable NMCFG and the vector variable MCS_MAW. MCS_MAW defines the memory address width of the physically connected large RAM blocks for the individual MCS instances and NMCFG defines if an MCFG module is implemented (NMCFG is 1) or not implemented (NMCFG is 0). The dependency is defined as follows: If NMCFG is 0 the value RAW of the i−th MCS instance can be determined as RAW = MCS_MAW[i], whereas MCS_MAW[i] is the i−th element of vector MCS_MAW. If NMCFG is 1 the value RAW of the i−th MCS instance can be determined as RAW = MCS_MAW[i] + 1 and further, if NMCFG is 1 all vector elements MCS_MAW[i] must be equal.

17.2 Architecture

Figure 135 MCS Architecture

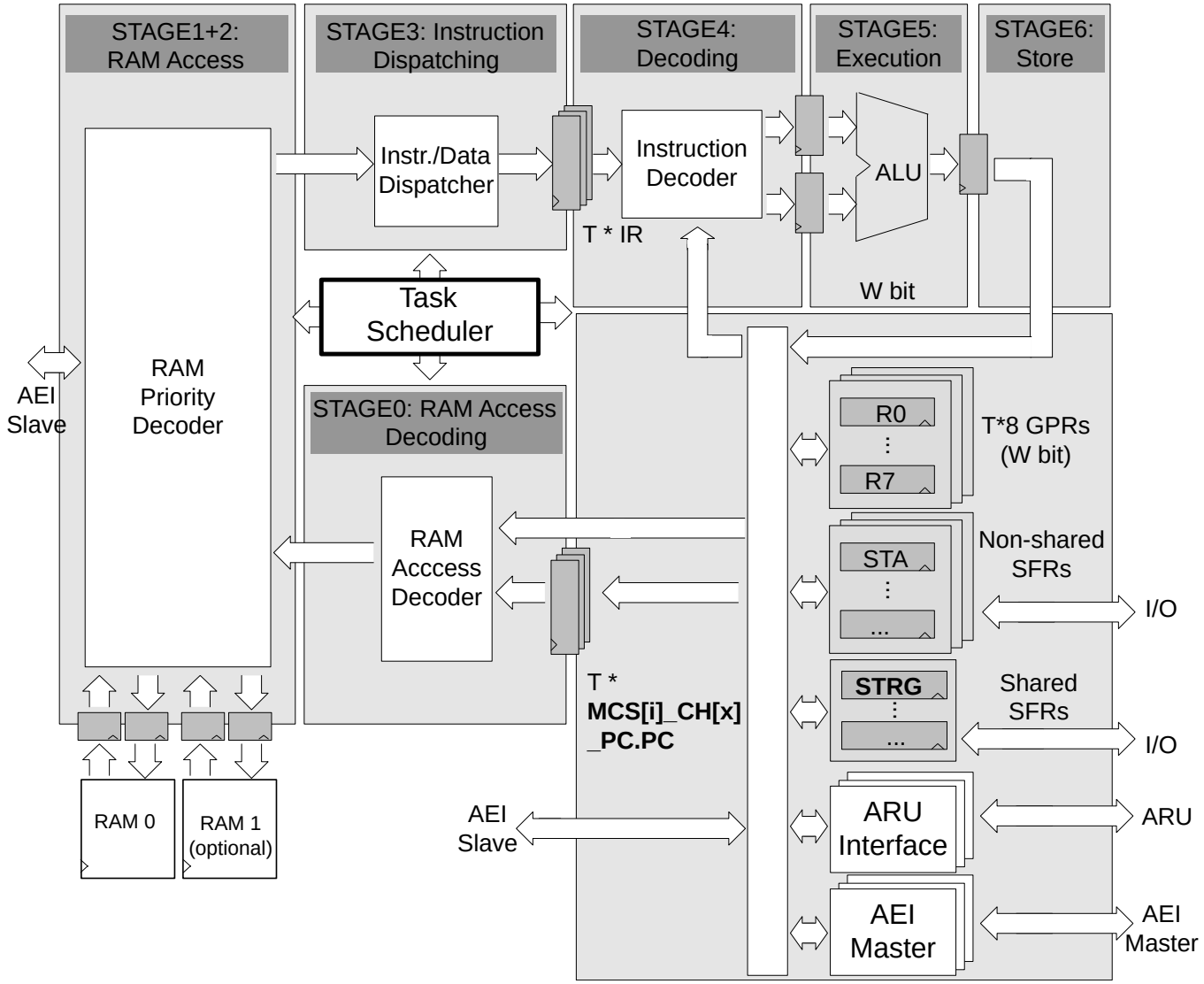


Figure *135 "MCS Architecture"* gives an overview of the MCS architecture assuming that all pipeline registers are implemented.

The data path of the MCS is shared by T so-called MCS-channels, whereas each MCS-channel executes a dedicated micro-program that is stored inside the RAM connected to the MCS module.

The connected RAM may contain arbitrary sized code and data sections that are accessible by all MCS-channels and an externally connected master via the microcontroller slave interface. More details about the RAM can be found in section *17.4 "Memory Organization"* .

An MCS-channel can also be considered as an individual thread of a processor that is scheduled to be the commonly used data path at a specific point in time. The execution of the different MCS-channels on the different pipeline stages is controlled by a central hardware related task scheduler, which enables immediate task switches in parallel to the program execution. Details about the task scheduler and the available scheduling algorithms can be found in section *17.3 "Scheduling"* .

Typically, if data has to be exchanged between different MCS-channels and/or the CPU, the connected RAM, which is accessible by all MCS-channels and the CPU, can be used.

Besides the commonly used data path, each MCS-channel has

- A set of eight General Purpose Registers (GPRs), each W bit wide,
- A set of non-shared Special Function Registers (SFRs) that are only accessible within a dedicated MCS-channel,
- A set of shared SFRs that are accessible by all MCS-channels,
- A channel specific instruction register (IR),
- A channel specific program counter register (**MCS[i]_CH[x]_PC.PC**),
- A dedicated ARU interface for communication with other ARU connected modules,

- An AEI Bus Master Interface for controlling other GTM submodules.

Generally, the GPRs of an MCS-channel x are only accessible by its corresponding MCS-channel x. However, the MCS provides a configuration that allows an MCS-channel x to access the GPRs of its successor MCS-channel x+1. This feature can be used to enlarge the number of registers for a specific MCS-channel x and/or to exchange data between neighboring channels.

For safety reasons, the register **MCS[i]_REG_PROT** can be used to define write protections for the neighboring registers of the individual MCS-channels.

In order to enable synchronization between different MCS-channels and/or the CPU, the MCS provides a common 24-bit wide trigger register that can be accessed as a shared SFR by all MCS-channels located in the same module. Writing to **STRG** sets bits and writing to **CTRG** clears bits in the common trigger register. To enable triggering of MCS-channels by CPU, the CPU can set bits in the common trigger register by writing to **MCS[i]_STRG** and clear bits by writing to **MCS[i]_CTRG** .

Considering the architecture in the figure above and assuming that all available pipeline stages are implemented (the generic parameters URIP, UROP, UDP, and UAP are set to 1), the main actions of the different pipeline stages are as follows:

- Pipeline stage 0 performs a setup of address, input data, and control signals for the next RAM access of a specific MCS-channel.
- In pipeline stage 1 and 2, the actual RAM access of a specific MCS-channel is executed, assuming an external connection of a synchronous RAM with a latency of one clock cycle.
- Pipeline stage 3 performs pre-decoding and dispatching of instructions and data resulting from the RAM.
- In pipeline stage 4, the instructions are decoded and data from the registers are loaded.
- In pipeline stage 5, the instruction is executed meaning that arithmetic operations are applied.
- Finally in pipeline stage 6, the calculated results are stored in the registers.

If any of the pipeline registers is not implemented, the adjacent pipeline stages are merged and thus processed within the same clock cycle.

The RAM priority decoder arbitrates RAM accesses that are requested by the CPU via the microcontroller slave interface and by the active MCS-channel. If both, CPU and an MCS-channel request a memory access to the same memory module the MCS-channel is prioritized.

Since the internal registers of the MCS can be updated by different sources (MCS write access by various instructions, CPU write access via the microcontroller slave interface, MCS write access by neighboring channel) a write conflict occurs if more than one source wants to write to the same register. In this case the result of the register is unpredictable. However, the software should setup its application in a way that such conflicts do not occur.

One exception is the common trigger register, which may be written by multiple sources (different MCS-channels and CPU) in order to enable triggering of different MCS-channels. Typically, the software should setup its application in a manner that different sources should not write the same bits in the trigger register.

17.3 Scheduling

The MCS provides a hardware related task scheduler, which globally controls the execution of the tasks in the different pipeline stages. The task scheduler implements four different scheduling modes, that can be selected by the **MCS[i]_CTRL_STAT.SCD_MODE** bit field. Depending on the selected scheduling mode, the task scheduler is selecting a dedicated MCS-channel that will be executed in pipeline stage 0 in the next clock cycle. Additionally, MCS-channels that are already present in the pipeline are shifted to its successor pipeline stage, with each clock cycle. This means, that the execution time of an MCS-channel in a specific pipeline stage is always one clock cycle.

The MCS task scheduler may also schedule an empty cycle to pipeline stage 0, in order to grant a time slice to the CPU for accessing the connected RAM.

If the task scheduler assigns an MCS-channel to pipeline stage 0, but this channel does not access the RAM, the CPU can access the corresponding RAM, even if the scheduler did not reserve an empty clock cycle.

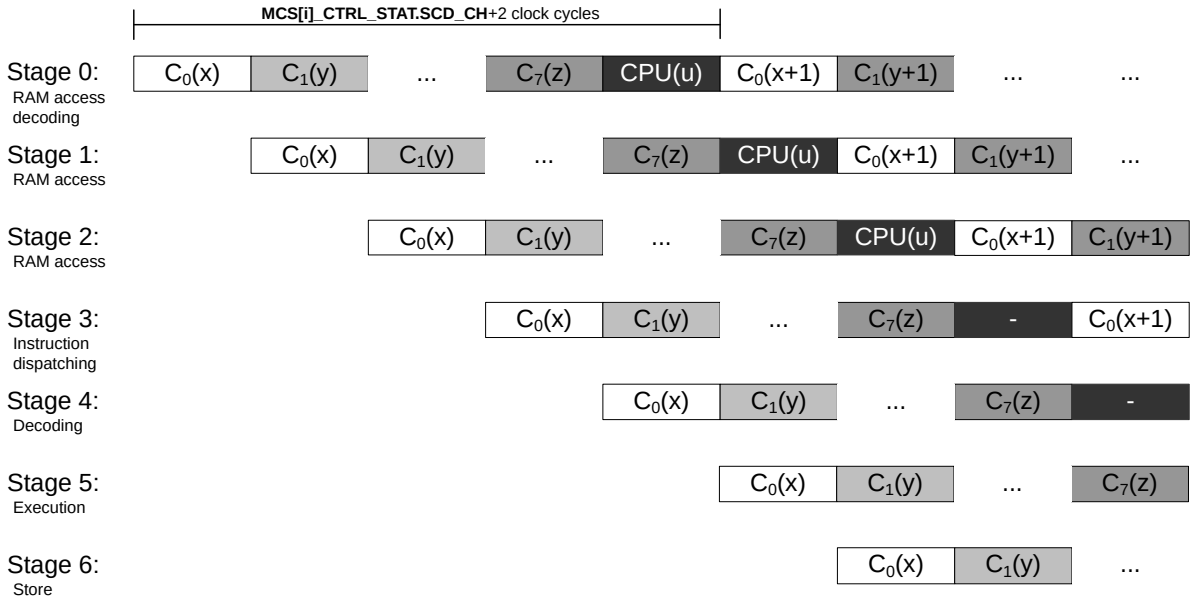
In the following, the available scheduling modes are described.

17.3.1 Round Robin Scheduling

The Round Robin Scheduling Mode implements the simplest scheduling algorithm. This algorithm schedules a predefined set of MCS-channels in the range [0; **MCS[i]_CTRL_STAT.SCD_CH**] in ascending order. After the last channel **MCS[i]_CTRL_STAT.SCD_CH** has been assigned to the pipeline, an empty cycle is scheduled in order to enable RAM access for the CPU. The parameter **MCS[i]_CTRL_STAT.SCD_CH** can be controlled by the register **MCS[i]_CTRL_STAT** . If the value of **MCS[i]_CTRL_STAT.SCD_CH** is greater than T-1, the scheduler assumes a value of T-1 for bit field **MCS[i]_CTRL_STAT.SCD_CH** .

Figure *136 "Timing of Round Robin Scheduling"* shows a timing example of the Round Robin Scheduling with T=8 MCS-channels (marked as C_0 to C_7) that are scheduled together with a CPU access to a pipeline width and NPS=7 stages. It is assumed that bit field **MCS[i]_CTRL_STAT.SCD_CH** is set to 7.

Figure 136 Timing of Round Robin Scheduling



The identifier $C_i(x)$ denotes that MCS-channel i is currently executing the instruction or data located in the memory at position x in the corresponding pipeline stage. The figure shows, which MCS-channel is activated in specific pipeline stage at a specific point in time.

Moreover, the figure shows that the Round Robin scheduling is always repeated after **MCS[i]_CTRL_STAT.SCD_CH** +2 clock cycles, which means that the time duration of an instruction cycle is **MCS[i]_CTRL_STAT.SCD_CH** +2 clock cycles. However, if the value **MCS[i]_CTRL_STAT.SCD_CH** + 2 is less than NPS, the duration of an instruction cycle is limited by the depth of the pipeline to NPS clock cycles. Thus the effective execution time of a single-cycle instruction is always MAX(**MCS[i]_CTRL_STAT.SCD_CH** +2, NPS) clock cycles, ignoring the latency of the pipeline.

If NPS is greater than T+1, NPS-T-1 additional empty cycles are inserted at the end of a round trip cycle. In this case the round trip time for the scheduler is determined by NPS, and thus the time duration for an instruction cycle is always NPS clock cycles.

The Round Robin scheduling algorithm has the characteristic that it fairly distributes all time slices to all MCS-channels and the CPU. This means, that the program execution time of a specific task is independent from the activity of any neighboring task or the CPU RAM access. Moreover, a program running in Round Robin mode never has to deal with data hazards. Therefore, a correct estimation of the actual program execution time is very easy. However, the Round Robin scheduling may waste clock cycles by scheduling MCS-channels that are not ready to execute an instruction (e.g. MCS-channel is disabled by CPU).

In order to keep the parallel execution of MCS channels in Round Robin Mode fully deterministic, which means that the execution time of an MCS channel never depends on the activity of another MCS channel, the following condition has to be met:
If the Modified Harvard Architecture is enabled (setting **MCS[i]_CTRL_STAT.EN_HVD** = 1), all MCS channels must use the same RAM module exclusively as code memory and the other RAM module exclusively as data memory. If this condition is violated, an intended parallel memory access of an MCS channel might be executed as a serial memory access, due to the activity on another MCS channel. This means that the Round Robin Mode does not have a determnistic timing behavior anymore. If the Modified Harvard Architecture is disabled (setting **MCS[i]_CTRL_STAT.EN_HVD** = 0), the behavior of the MCS is always deterministic in Round Robin mode, independently of the memory layout.

17.3.2 Accelerated Scheduling

In order to improve the computational performance, the accelerated scheduling mode provides two key features. Firstly, the scheduler only selects MCS-channels that are not suspended and thus can actually execute an instruction. Secondly, the scheduler applies instruction prefetching to minimize empty cycles in the pipeline. An MCS-channel is entering suspended state due to one of the following reasons:

- An MCS-channel is executing a read or write request to an ARU connected submodule (instruction ARD, AWR, ARDI, AWRI, NARD, NARDI).
- An MCS-channel is executing a read or write request at its bus master interface (instruction BRD, BWR, BRDI, BWRI).
- An MCS-channel waits on a register match event (e.g. instruction WURM), in order to wait on a desired register value (e.g. trigger event from another MCS-channel).
- An MCS-channel is disabled.

In the case of instruction prefetching, the scheduler will assign an MCS-channel C_p to pipeline stage 0, which is already present in another pipeline stage. This means, that the execution of the last instruction C_p located in the memory MEM(**MCS[i]_CH[x]_PC.PC** /4) is not yet finished completely, whereas **MCS[i]_CH[x]_PC.PC** is the current value of the program counter of MCS-channel C_p. Thus, the newly scheduled MCS-channel C_p will speculatively prefetch a successor instruction MEM(**MCS[i]_CH[x]_PC.PC** /4+PFO) respecting a prefetch offset PFO under the assumption that there will be no branch and no memory access in the program between the instructions MEM(**MCS[i]_CH[x]_PC.PC** /4) and MEM(**MCS[i]_CH[x]_PC.PC** /4+PFO). The prefetch offset PFO is determined by counting the number of already scheduled MCS-channels Cp in the pipeline. However, if the assumption fails, the pipeline will be flushed by replacing all MCS-channel C_p of the pipeline with an empty cycle, as soon as the instruction decoder detects a branch or a memory access. All other MCS-channels unequal to MCS-channel C_p within are not affected by the flushing action. The flushing action is always synchronized to the last pipeline stage NPS-1.

Besides the flushing conditions mentioned above, there are also other conditions that cause a flush of the pipeline for a specific MCS-channel. In the following, all possible flushing events are summarized:

- An MCS-channel is enabled.
- An MCS-channel is entering a suspended state.
- An MCS-channel is taking a conditional or unconditional branch (instruction JMP, JBS, JBC, CALL, RET, JMPI, JBSI, JBCI, CALLI).
- An MCS-channel accessing memory for data transfer (instruction MRD, MWR, MRDI, MWRI, MRDIO, MWRIO, MWRL, MWRIL, PUSH, POP).
- An MCS-channel is executing a read or write request at its bus master interface (instruction BRD, BWR, BRDI, BWRI).
- An MCS-channel is modifying the trigger register (write access to **CTRG** or **STRG**) and the same channel is reading back this register (read access to **CTRG** or **STRG**) while the delay between both accesses is less than UAP+UDP+1 clock cycles.

In general, each MCS-channel can accept instruction prefetching. However, there are some cases in which an upcoming flushing of the pipeline can be easily detected by the MCS hardware due to evaluation of internal states. Therefore, it is defined that an MCS-channel accepts instruction prefetching only under the following conditions:

- An MCS-channel is currently not in the second cycle of a two-cycle control flow instruction (instruction CALL, RET).
- An MCS-channel is currently not in the second cycle of a three-cycle memory access instruction (instruction MWRL, MWRIL).

The accelerated scheduling mode guarantees, that the time duration of an instruction cycle varies between 1 and T+1 cycles. Hence, a single--cycle instruction has an effective execution time between 1 to T+1 clock cycles, depending on the number of suspended MCS-channels and

the actual instruction sequence. The worst case execution time occurs if all channels are active and the CPU also accesses the RAM. The best case occurs e.g. if only one MCS-channel is enabled and the executed program sequence has only linear code without branches and memory access.

The algorithm of the accelerated scheduling mode first evaluates the state of all available MCS-channels as well as a CPU request to the RAMs and then it decides if a specific MCS-channel or an empty cycle is assigned to pipeline stage 0 in the next clock cycle.

Note:

The accelerated scheduling mode treats RAM access requests from the CPU in a similar manner as MCS-channels, which means that empty cycles for RAM requests are only inserted into the pipeline if there is an active RAM request from the CPU or no other task can be scheduled.

In order to fairly trade all available MCS-channels as well as CPU RAM requests and to guarantee a worst case execution time of T+1 clock cycles, an additional task prioritization scheme is applied that dynamically prioritizes all MCS-channels and a CPU memory access depending on the history of the scheduler's decisions. The algorithm of the accelerated scheduler mode is executed every clock cycle and it works in the following manner:

1. Try to find an MCS-channel C_r with highest priority that is not suspended and not already scheduled to the pipeline stages 0 to NPS-2. If C_r is found assign C_r to pipeline stage 0 and finish scheduling for current clock cycle.
2. Otherwise, try to find an MCS-channel C_p with highest priority that is not suspended and accepts instruction prefetching. If C_p is found assign C_p to pipeline stage 0 and finish scheduling for current clock cycle.
3. Otherwise, try to find an MCS-channel C_s with highest priority that is suspended and accepts instruction prefetching. If C_s is found assign C_s to pipeline stage 0 and finish scheduling for current clock cycle.
4. Otherwise, assign an empty cycle to pipeline stage 0 and finish scheduling for current clock cycle.

The underlying task prioritization scheme tracks the history of the scheduled MCS-channels in a list consisting of T+1 items. The list is initialized with all MCS-channels followed by a reserved time slot for the CPU RAM access. The position of an MCS-channel within this list implicitly defines the priority, while the back of this list holds the MCS-channel with highest priority. Whenever the scheduling algorithm described above has found an MCS-channel C_r or C_p to be scheduled in the next clock cycle, it removes this item from the list and put it to the front of the list. In order to fairly prioritize all MCS-channels, the algorithm also moves the item at the back of the list to the second position in the list, after the inserted scheduled front item. Since the list always contains all possible MCS-channels and with each clock cycle each nonscheduled item is moved at least one position towards the end of the list, it is obvious that each MCS-channel will have the highest priority not later than T+1 clock cycles.

Figure *137 "Timing of Accelerated Scheduling"* shows a timing example of the accelerated scheduling with NPS=7 pipeline stages.

Figure 137 Timing of Accelerated Scheduling

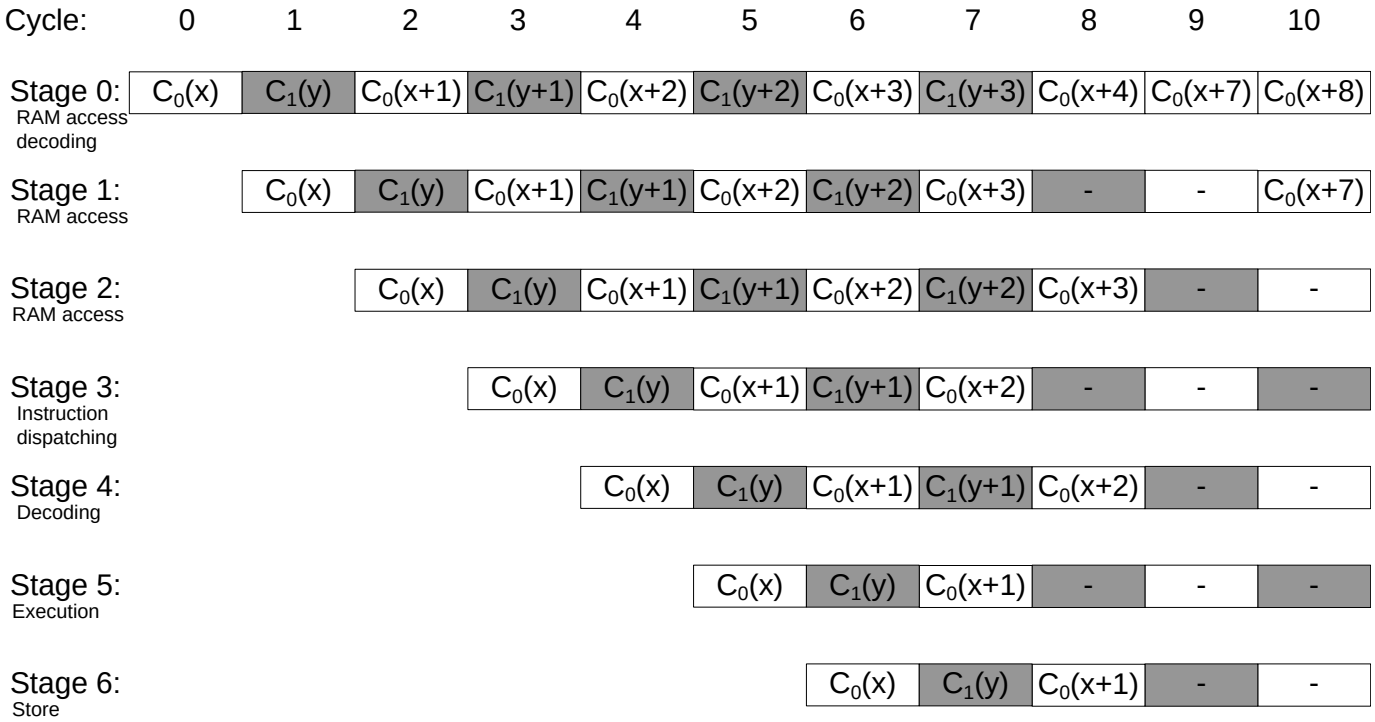


Figure 138 MCS Code Example for Accelerated Scheduling

MCS-channel 0

Memory Location	Instruction
x+0	ADDL R0, 7
x+1	JBC STA, Z, x+7
x+2	MOVL R2, 5

MCS-channel 1

Memory Location	Instruction
y+0	WURM STRG, R1, 0xFFFF
y+1	ADDL R2, 5
y+2	ADD R3, R2

The example assumes that initially MCS-channels 0 and 1 are enabled and the program for each MCS-channel is located in the RAM as shown in Figure *138 "MCS Code Example for Accelerated Scheduling"* . Since both channels are ready to run, the scheduler fairly selects the channels in an alternating order, as it can be obtained in stage 0 at the clock cycles before cycle 7. Since MCS-channel 1 is entering suspended state (to wait on a trigger bit) at cycle 7 in stage 6 with the instruction of memory location y, the scheduler will only select MCS-channel 0 in the following cycles by applying instruction prefetching. Moreover, by entering the suspended state of channel 1 at cycle 7, all other time slots that have been allocated to channel 1 are flushed from the pipeline. This flushing is denoted in Figure *137 "Timing of Accelerated Scheduling"* by the dashes of the grey shaded cycles.

The scheduler applies instruction prefetching during the whole sequence, due to the fact that the number of enabled channels is always less than the available number of pipeline stages NPS.

The actual state of pipeline in cycle 9 and 10 depends on conditional branch instruction of memory location x + 1 (cycle 8 stage 6). If the branch is not taken, all prefetched instructions of channel 0 are correct and the linear code execution of MCS-channel 0 is continued. However, if the branch to memory location x+7 is taken, as it is shown in the Figure, the scheduler will fetch the instruction C_0 (x+7) in cycle 9 at stage 0. Further, the prefetched instructions are no longer valid and therefore the reserved time slots for channel 0 are flushed, which is denoted by the dashes with white background in Figure *137 "Timing of Accelerated Scheduling"* .

17.3.3 Single Prioritization Scheduling

The Single Prioritization Scheduling mode is an extended variant of the Accelerated Scheduling mode, which additionally applies a task prioritization of a single MCS-channel. In this mode, the bit field **MCS[i]_CTRL_STAT.SCD_CH** is used to identify a dedicated MCS-channel that is always preferred during scheduling. This means, that the scheduler will assign preferred MCS-channel **MCS[i]_CTRL_STAT.SCD_CH** to pipeline stage 0, as long as this channel is not suspended. If the preferred MCS-channel is entering its suspended state, the scheduling algorithm switches to the accelerated scheduling as previously described in section *17.3.2 "Accelerated Scheduling"* . Whenever the MCS-channel **MCS[i]_CTRL_STAT.SCD_CH** is resuming from its suspended state, the scheduler switches back and assigns the channel **MCS[i]_CTRL_STAT.SCD_CH** to pipeline stage 0 until the next suspension event occurs. If the bit field **MCS[i]_CTRL_STAT.SCD_CH** contains the value T or higher, the task scheduler will always prioritize CPU access to the RAM. This means, whenever the task scheduler detects that the CPU wants to access an MCS-RAM, the scheduler will assign an empty cycle into pipeline stage 0. If the CPU does not access the RAM any more, it switches back to the accelerated mode, as described previously in section *17.3.2 "Accelerated Scheduling"* .

In consequence, the Single Prioritization Scheduling mode cannot guarantee a maximum time duration of an instruction cycle for the overall execution of all MCS-channels, since it strongly depends on the activity of the prioritized MCS-channel **MCS[i]_CTRL_STAT.SCD_CH** . However, the Single Prioritization Scheduling mode provides the fastest possible execution for MCS-channel **MCS[i]_CTRL_STAT.SCD_CH** . Moreover, during the time span, in which the prioritized MCS-channel **MCS[i]_CTRL_STAT.SCD_CH** is suspended, this mode guarantees a duration of 1 to T+1 clock cycles of an instruction cycle for all non-prioritized channels.

If an MCS program is being executed in Single Prioritization Scheduling mode, the CPU cannot access the MCS RAM from which the prioritized program code is fetched. In this case the CPU RAM access will be blocked until the prioritized MCS-channel is entering its suspended state. If data needs to be exchanged between CPU and a prioritized MCS-channel, either the general purpose registers or the opposite RAM module (if USR = 1) should be used.

17.3.4 Multiple Prioritization Scheduling

The Multiple Prioritization Scheduling mode is an extended variant of the Accelerated Scheduling mode, which additionally applies a task prioritization for multiple MCS-channels. In this mode, the bit field **MCS[i]_CTRL_STAT.SCD_CH** is used to identify a set of dedicated MCS-channels, which are always preferred during scheduling. The identifiers of the prioritized MCS-channels x with $x \in \{0, 1, \ldots,$ **MCS[i]_CTRL_STAT.SCD_CH** $\}$ and the non-prioritized channels x with $x \in \{($ **MCS[i]_CTRL_STAT.SCD_CH** $+1), 1, \ldots, (T-1)\}$. The individual priority for the set of prioritized MCS-channels is applied in descending order, which means that MCS-channel 0 has the highest priority, followed by MCS-channel 1, which has the second highest priority, and so on. The non-prioritized MCS-channels do not have any priority. A value of T-1 or higher for the bit field **MCS[i]_CTRL_STAT.SCD_CH** means that all T MCS-channels are prioritized MCS-channels.
With each clock cycle, the Multiple Prioritization Scheduling mode will assign the non-suspended MCS-channel with the highest priority from the set of prioritized MCS-channels to pipeline stage 0, as long as there are non-suspended prioritized MCS-channels available. If all prioritized MCS-channels are suspended, the scheduling algorithm switches to the accelerated scheduling as previously described in section *17.3.2 "Accelerated Scheduling"* and it schedules the non-prioritized channels. Whenever a prioritized MCS-channel is resuming from its suspended state, the scheduler switches back and applies the described prioritization scheme until the next suspension event occurs.

In consequence, the Multiple Prioritization Scheduling mode cannot guarantee a maximum time duration of an instruction cycle for the overall execution of all MCS-channels, since it strongly depends on the activity of the prioritized MCS-channels. However, the Multiple Prioritization Scheduling mode provides the fastest possible execution for prioritized MCS-channels. Moreover, during the time span, in which all prioritized MCS-channels are suspended, this mode guarantees a duration of 1 to T+1 clock cycles of an instruction cycle for all non-prioritized channels.

If an MCS program is being executed in Multiple Prioritization Scheduling mode, the CPU cannot access the MCS RAM from which the prioritized program code is fetched. In this case, the CPU RAM access will be blocked until the last prioritized MCS-channel is entering its suspended state. If data needs to be exchanged between CPU and a prioritized MCS-channel, either the general purpose registers or the opposite RAM module (if USR = 1) should be used.

17.4 Memory Organization

The MCS provides a configurable RAM interface, which either supports the connection of one (USR = 0) or two (USR = 1) RAM modules with an address width of RAW bits and a data width of RDW bits. Hence, the MCS module provides a byte wise address range from 0 to $2^{RAW+USR+2}-1$, although the memory is only word wise addressable.

If only one RAM port is used (USR = 0), the entire address range of the MCS only uses one memory page, which ranges from 0 to MP0-4, assuming a byte wise addressing scheme.

If two RAM ports are used (USR = 1), the entire address range of the MCS is divided into two seamless memory pages, whereas memory page 0 ranges from 0 to MP0-4 and memory page 1 ranges from MP0 to MP1-4, assuming a byte wise addressing scheme for all addresses.

The parameters MP0 and MP1 depend on the variables NMCFG, MCS_MAW and MCS_ERM as follows:
If no MCFG module is available (NMCFG = 0), the parameters MP0 and MP1 are fixed and can be determined as MP0 = $2^{MCS\_MAW[i]+2}$ and MP1 = $2^{MCS\_MAW[i]+2} + 2^{MCS\_MAW[i]-(1-MCS\_ERM[i])+2}$. The variables MCS_MAW[i] and MCS_ERM[i] represent the i-th vector element of the corresponding vector variable MCS_MAW and MCS_ERM, whereas the index i corresponds to the instance number of an MCS instance. MCS_MAW[i] defines the memory address width of the physically connected large RAM block that is connected to RAM port 0 of the i-th MCS. The parameter MCS_ERM[i] defines, if the RAM block that is associated to RAM port 1 of the i-th MCS instance has full size (MCS_ERM[i] is 1) or half size (MCS_ERM[i] is 0).

If an MCFG module is available (NMCFG=1), the actual values for the parameters MP0 and MP1 are configurable and provided by the module MCFG. Details about the configuration with module MCFG can be found in section *19 "Memory Configuration (MCFG)"* .

Since the MCS memory is accessible by the MCS itself and from outside via the microcontroller slave interface, the MCS needs to arbitrate between both. If both AEI and MCS try to access same RAM page at the same point in time, the implemented arbitration scheme always prefers the memory access from the MCS.

If AEI and MCS try to access different RAM pages at the same point in time, there is no need for arbitration and therefore both accesses can be executed in parallel.

If an ECC error occurs while an MCS-channel x reads data or an MCS-channel x executes an instruction that was fetched with an ECC error, the corresponding MCS-channel is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, and no other MCS registers or memory cells are updated.
==> DG: If the ECC error occurs during an instruction fetch, the debug trace interface does not signalize an instruction fetch. If the ECC error occurs during a data fetch, the debug trace interface does not signalize a data transfer but the instruction trace interface signalizes the fetched instruction.

The MCS always checks that all memory accesses are within a valid range and no overflow of the program counter occurs. If the RAM address of an instruction fetch or a data access of an MCS-channel x is greater than or equal to the parameter MP1, or if an instruction execution produces a program counter overflow, the corresponding MCS-channel is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, the actual write access to non-existing memory location is not applied, and no other MCS registers or memory cells are updated.
==> DG: If the memory overflow occurs during a data fetch, the debug trace interface signalizes the memory access instruction at its instruction trace interface, but it does not signalize any data transfer at its data trace interface. On the other hand, if the memory overflow occurs during an instruction fetch, the debug trace interface does not signalize any instruction at its instruction trace interface.

The Cluster Configuration Module (CCM) implements so-called address range protectors (ARPs), in order to protect several code and data sections of the MCS memory. If an MCS-channel x writes to such a protected memory region, the MCS-channel is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, the actual write access to the protected memory location is not applied, and no other MCS registers or memory cells are updated.
==> DG: If a write access to a protected memory region occurs, the debug trace interface signalizes the write access instruction at its instruction trace interface, but it does not signalize a data transfer at its data trace interface.

During a RAM initialization phase, initiated by a write access to bit field **MCS[i]_CTRL_STAT.RAM_RST** , the entire content of the RAM is initialized with zeros. During this phase any write access to the MCS memory is protected.

17.4.1 Modified Harvard Architecture

If an MCS provides two RAM ports (USR = 1), it implements a modified Harvard architecture which means that an MCS program can trade off between flexibility for code and data sections and performance improvements with respect to memory accesses.

In general, the MCS supports arbitrarily sized code and data section which can be assigned to both memory pages. However, if code and data sections are located on the same memory page, a memory access instruction will require 2 instruction cycles with serial RAM accesses. During the first instruction cycle the instruction is fetched and decoded and during the second instruction cycle the actual memory access is executed. On the other hand, if code and data sections are located on different memory pages, a memory access instruction will only require 1 instruction cycle with a parallel RAM access, since instruction fetch, decoding and memory access can be executed within the same instruction cycle. An MCS program can always handle both variants of memory accesses.

If a program is structured in a way that all code and data sections are located on different memory pages and the program is executed in Round Robin scheduling mode the MCS program achieves a true speedup of factor two for memory accesses compared to a program structure using a single memory page for code and data section. This is due to the fact that the Round Robin mode does not have to deal with data hazards. If any other scheduling mode is selected, data hazards might occur which means that the speedup of memory can be limited.

The parallel memory accesses provided by the proposed modified Harvard architecture can be enabled or disabled by the bit field **MCS[i]_CTRL_STAT.EN_HVD** . If bit field **MCS[i]_CTRL_STAT.EN_HVD** is cleared, all memory accesses are executed as a 2 cycle serial memory accesses. In this case, the actual assignment of code and data sections to the different memory pages does not have any impact on the program execution time.

17.5 AEI Bus Master Interface

The MCS module provides an AEI bus master interface, which enables communication with externally connected modules. The data width of this interface is BDW bit and the address width is BAW bit leading to a maximum byte wise address ranging from 0 to $2^{BAW+2}-1$. The bus master interface is shared among all available MCS-channels meaning that each MCS-channel may initiate a read or write access on the bus but only one channel can be served at a specific point in time.

However, the AEI bus master interface guarantees, that a bus access is always completed within one instruction cycle and bus access of different MCS-channels do not modify the latency of each other. The only exceptions are bus accesses to RAM modules (e.g. DPLL RAM) either from AEI bus master or CPU via the microcontroller slave interface. Since bus accesses to RAM modules cannot be completed within a single clock cycle, additional wait cycles have to be inserted into the bus protocol leading to the fact that the MCS-channel that is accessing the RAM is entering a suspended state. Moreover, if an MCS-channel is accessing a RAM module, the latency of a bus access in another MCS-channel can also be modified even if the neighboring channel is accessing only a configuration register.

The AEI bus master interface of an MCS module is connected to the AEI bus system in order to control the submodules of the GTM within MCS. However, it is not possible to access the entire GTM by a single MCS module. The i-th MCS instance can only access the GTM submodules that are located within the i-th cluster of the GTM. Details about the clusters and its available modules can be found in section *3.1 "Overview"* . Additionally, the address map for accessing GTM with the AEI bus master interface of an MCS differs from the address map for an externally connected mircrocontroller slave interface. The actual addresses for accessing registers and memory locations with the AEI bus master interface of an MCS is defined by the MCS interface description of the appropriate module specification.
Since the submodules of the GTM can be accessed by the CPU and the AEI bus master of an MCS, the GTM-IP provides an additional arbitration scheme to manage parallel accesses from both master interfaces. If CPU and an MCS want to access a GTM submodule of the same cluster, the arbiter will grant the access to the MCS. However, if the CPU and all the MCS instances want to access GTM submodules of different clusters, the accesses can be executed in parallel.

The AEI bus master interface can be controlled by the MCS instructions BRD, BRDI, BWR, and BWRI. These instructions are described in section *17.9 "Instruction Set"* .

If the bit field **MCS[i]_CTRL_STAT.HLT_AEIM_ERR** is set and if an MCS-channel x performs an invalid bus master access to a GTM submodule, which returns an AEI status value unequal to zero, the MCS-channel x is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, and no other MCS registers or memory cells are updated. On the other hand, if bit field **MCS[i]_CTRL_STAT.HLT_AEIM_ERR** is cleared the AEI status information is ignored within the MCS. Nevertheless, the status register **CCM[i]_AEIM_STA** is still getting updated.

17.6 AXI Bus Master Interface

The GTM provides an AXI bus master interface which allows to initiate bus transactions externally to the GTM. Details about the connection of this bus interface can be found in the specification of the microcontroller.
The GTM integrates a central AXI transaction generator module that is controlling and arbitrating the entire AXI master communication of the GTM. This module provides up to 16 communication slots to a cluster of the GTM which implements an MCS instance. In order to control the common AXI master functionality and the functionality of the dedicated AXI communication slots the AEI bus master interface of the MCS provides direct access to the central AXI transaction generator module. This means that the AXI bus master interface is controlled with the standard AEI bus master instructions (instruction BRD, BRDI, BWR, and BWRI). A detailed explanation about how to use AXI bus master interface can be found in section *4 "AXI Master"* .

17.7 MCS Software Reset of Channels

Writing via the configuration interface the value 0b1 to the bit field **MCS[i]_RESET.RST[x]** , will immediately set the content of the following registers of MCS-channel x to its initial hardware reset state.

MCS[i]_CH[x]_CTRL
MCS[i]_CH[x]_PC
MCS[i]_CH[x]_R[y]
MCS[i]_CH[x]_ACB
MCS[i]_CH[x]_MHB
MCS[i]_CH[x]_IRQ_NOTIFY
MCS[i]_CH[x]_IRQ_EN
MCS[i]_CH[x]_IRQ_MODE
MCS[i]_CH[x]_EIRQ_EN
MCS[i]_CAT.CAT[x]
MCS[i]_CWT.CWT[x]
MCS[i]_ERR.ERR[x]
MCS[i]_HBP[h]_STATUS.HALT_CH[x]

Atomic reset of multiple channels is possible by writing a 1 on each associated bit. E.g: Writing 0x0F to register **MCS[0]_RESET** will reset the MCS0 channel 0,1,2,3.

17.8 Debugging

In order to ease the debugging for more complex applications, the MCS architecture provides NHBP (with NHBP=2) configurable hardware breakpoints.
These hardware break points can be used by debugger tools for the realization of standard debugging features like single stepping of program sequences or setting various program or data break points. The dedicated IRQ signals of the hardware break points can also be used to trigger watch points in the microcontroller system.

Figure 139 Hardware Break Point Logic

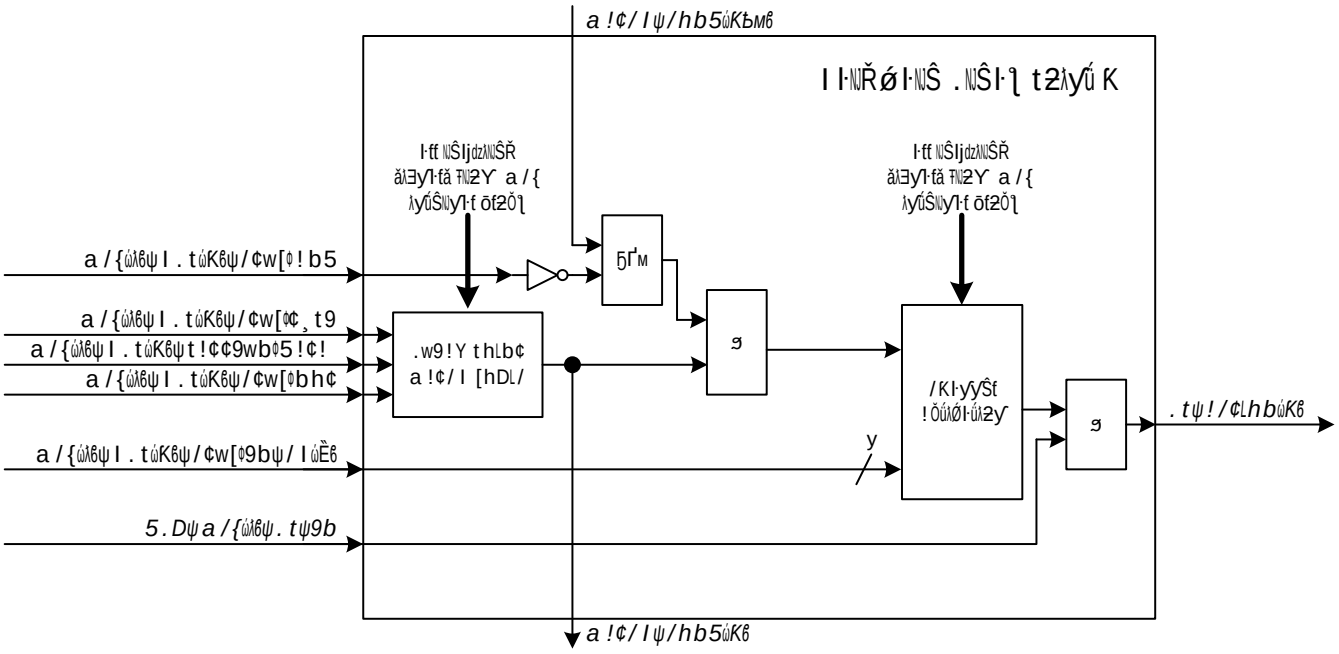


Figure *139 "Hardware Break Point Logic"* shows a block diagram of a hardware break point.
The h-th hardware break point is configured by the registers **MCS[i]_HBP[h]_CTRL** and **MCS[i]_HBP[h]_PATTERN** (with h = 0 to NHBP−1 and i = 0 to NMCS − 1).
The bit field **MCS[i]_HBP[h]_CTRL.TYPE** configures the actual type of break point. The following basic types are available:

- INSTR – setting up an instruction break point at a specific address range. The match condition *MATCH_COND* [h:h] of the h-th break point is true, if an instruction from memory location z (z is a word address) with z >= **MCS[i]_HBP[h]_PATTERN.DATA** [15:2] and z <= **MCS[i]_HBP[h]_PATTERN.DATA** [31:18] is entering the last pipeline stage for execution but before the actual instruction is storing any results to the registers. Hence, if such a break point issues a halt, the associated program counter is also not incremented and it still holds the address z of the current break point location. If the bit field **MCS[i]_HBP[h]_CTRL.NOT** is set, the matching range condition is negated, which means that the address match condition is true if z < **MCS[i]_HBP[h]_PATTERN.DATA** [15:2] or z > **MCS[i]_HBP[h]_PATTERN.DATA** [31:18].
- DADR – setting up a data break point at a specific address range. The match condition *MATCH_COND* [h:h] of the h-th break point is true, if data is read and/or written from/to memory location z (z is a word address) with z >= **MCS[i]_HBP[h]_PATTERN.DATA** [15:2] and z <= **MCS[i]_HBP[h]_PATTERN.DATA** [31:18] and after the memory access instruction is fully executed meaning that the program counter is pointing to the instruction that is following the memory access instruction. Again, if the bit field **MCS[i]_HBP[h]_CTRL.NOT** is set, the matching range condition is negated, which means that the address match condition is true if z < **MCS[i]_HBP[h]_PATTERN.DATA** [15:2] or z > **MCS[i]_HBP[h]_PATTERN.DATA** [31:18].
- DPAT – setting up a data break point at a specific data pattern. The match condition *MATCH_COND* [h:h] of the h-th break point is true, if data that is read and/or written equals to the value defined in the bit field **MCS[i]_HBP[h]_PATTERN.DATA** and after the memory access instruction is fully executed meaning that the program counter is pointing to the instruction that is following the memory access instruction. If the bit field **MCS[i]_HBP[h]_CTRL.NOT** is set, the pattern match condition is negated leading to a match condition with value true on any read and/or write access unequal to **MCS[i]_HBP[h]_PATTERN.DATA** .

As it can be obtained from Figure *139 "Hardware Break Point Logic"* , the resulting match condition *MATCH_COND* [h:h] is additionally processed until it finally leads to a break point action event *BP_ACTION* [h:h]. The logical AND conjunction between *MATCH_COND* [h:h] and *MATCH_COND* [h+1:h+1] is applied if bit field **MCS[i]_HBP[h]_CTRL.AND** is set and both break points are data break points. Further, the bit field **MCS[i]_HBP[h]_CTRL.EN_CH[x]** with 0 <= x < T must enable a break point for a specific MCS-channel x and the global enable signal *DBG_MCS[i]_BP_EN* must also be enabled in order to fire the break point action *BP_ACTION* [h:h] of the h-th hardware break point. Please note that the signal *DBG_MCS[i]_BP_EN* is controlled externally by the microcontroller. Moreover, any falling edge of this signal will resume all halted break points of the i-th MCS.

If the logical AND conjunction of the first breakpoint (h=0) needs to be applied, all the break point enable bit fields of the second break point (h=1) shall be disabled (**MCS[i]_HBP[1]_CTRL.EN_CH[x]** = 0 with 0 <= x <= 7) in order to ensure that only the first break point (h=0) is triggered.

The bit field **MCS[i]_HBP[h]_CTRL.SCOPE** defines the actual action that is executed if the *BP_ACTION* [h:h] is set to true. The following configurations for bit field **MCS[i]_HBP[h]_CTRL.SCOPE** are available:

- NO_HLT – the value true at *BP_ACTION* [h:h] only triggers the associated break point interrupt, which can be controlled with registers **MCS[i]_HBP[h]_IRQ_EN** , **MCS[i]_HBP[h]_IRQ_NOTIFY** , **MCS[i]_HBP[h]_IRQ_FORCINT** , and **MCS[i]_HBP[h]_IRQ_MODE** according to the functionality described in section *3.12 "GTM-IP Interrupt Concept"* . The execution of the program sequence is neither halted nor modified in any way. This configuration can be used for triggering watch points in a microcontroller system.
- HLT_CH – an MCS-channel x with **MCS[i]_HBP[h]_CTRL.EN_CH[x]** set to true and *MATCH_COND* [h:h] set to true triggers its dedicated break point interrupt and further it halts the MCS-channel x. The other MCS-channel just keeps on running. The halted MCS-channel is signalized in the break point status register **MCS[i]_HBP[h]_STATUS** by setting the corresponding bit field **MCS[i]_HBP[h]_STATUS.HALT_CH[x]** to true. Moreover, the bit fields **STA.EN** and **MCS[i]_CH[x]_CTRL.EN** are still set to true in order to signalize that an MCS-channel is not explicitly disabled by configuration interface or from the MCS itself. The actual behavior of the halt depends on the type of break point as follows: If **MCS[i]_HBP[h]_CTRL.TYPE** is INSTR the break point type is configured as instruction break point, which means that the MCS-channel x is halted if the corresponding instruction reaches the last pipeline stage. However, the instruction does not update any results in the registers (e.g. program counter or GPRs) which ensures that all the registers reflect the correct state of the executed program sequence before the break point is reached. A write access with value true to the bit field **MCS[i]_HBP[h]_STATUS.HALT_CH[x]** will resume the MCS-channel x from its break point by enabling the channel again starting with the execution of the non-finished instruction from the break point but without halting the MCS-channel on the current instruction. Any debugger related modification of a register that is referenced by the instruction at the break point will be respected by the instruction after the resume. If **MCS[i]_HBP[h]_CTRL.TYPE** is DADR or DPAT, the break point type is configured as a data break point which means that the MCS-channel x is halted, if the corresponding instruction is fully executed meaning that the results of the instruction are updated in the registers and/or in the memory and the program counter is incremented. A write access with value true to the bit field **MCS[i]_HBP[h]_STATUS.HALT_CH[x]** will resume the MCS-channel x from its break point by enabling the channel again starting with instruction that follows immediately the break point.
- HLT_SYS – an MCS-channel x with **MCS[i]_HBP[h]_CTRL.EN_CH[x]** set to true and *MATCH_COND* [h:h] set to true triggers its dedicated break point interrupt, it halts the MCS-channel x (similar to HLT_CH described previously), and further it issues a system halt. System halt means that the GTM halt logic is activated and the halt is also signalized to the microcontroller. Depending on the actual integration of the silicon vendor, the system halt can be extended in a way that CPUs and/or other peripheral modules of the microcontroller are halted synchronously with an MCS break point. The activation of the GTM halt logic is switching off system clock of all GTM modules, but the registers and RAM modules that are accessible via the bus configuration interface can still be read and/or modified. The GTM halt logic is deactivated if the halted MCS break points which were configured with a **MCS[i]_HBP[h]_CTRL.SCOPE** set to HLT_SYS are resumed by a write access to its corresponding **MCS[i]_HBP[h]_STATUS.HALT_CH[x]** with value true.

17.9 Instruction Set

This section describes the entire instruction set of the MCS submodule. First, a brief overview over all available instructions is given and a detailed description of each instruction can be found in sections *17.9.1 "MOVL Instruction"* and the following sections.

In general, each instruction is RDW bit wide but the duration of each instruction varies between several instruction cycles. As already described in section *17.3 "Scheduling"* , the number of required clock cycles for an instruction cycle can be fixed or variable, depending on the selected scheduling mode. In the case of the Round Robin Scheduling, the duration is fixed with T+1 clock cycles, in the case of the Accelerated Scheduling the duration is variable in the range between 1 and T+1 clock cycles, and in all other Scheduling modes the duration is also variable and may even be more than T+1 clock cycles, depending on the application.

Before the available instructions are described, some commonly used terms, abbreviations and expressions are introduced:

OREG: The operation register set OREG:={R0, R1, ..., R7} ∪ { **STA** , **ACB** , **CTRG** , **STRG** , **TBU_TS0** , **TBU_TS1** , **TBU_TS2** , **MHB** } include all MCS accessible internal channel specific GPRs {R0, R1, ..., R7} and the sub set { **STA** , **ACB** , **CTRG** , **STRG** , **TBU_TS0** , **TBU_TS1** , **TBU_TS2** , **MHB** } of SFRs.

XOREG: The extended operation register set XOREG:=OREG ∪ {RS0, RS1, ..., RS7} ∪ { **GMI0** , **GMI1** , **DSTA** , **DSTAX** , **AXIMI** , **MSINT** , **TIOI** , **TIOSL** } extends the operation registers set OREG by the GPRs of the succeeding MCS-channel {RS0, RS1, ..., RS7} and the SFRs { **GMI0** , **GMI1** , **DSTA** , **DSTAX** , **AXIMI** , **MSINT** , **TIOI** , **TIOSL** }.

WXREG: The extended wait instruction operation register set WXREG:=OREG ∪ { **GMI0** , **GMI1** , **DSTA** , **DSTAX** , **AXIMI** , **MSINT** , **TIOI** , **TIOSL** } extends the operation registers set OREG by the SFRs { **GMI0** , **GMI1** , **DSTA** , **DSTAX** , **AXIMI** , **MSINT** , **TIOI** , **TIOSL** }.

AREG: The ARU register set AREG:={R0, R1, R2, ..., R7, ZERO} includes all registers that can be written by incoming ARU transfers (ARD, ARDI, NARD, and NARDI instructions). These registers include all eight general purpose registers. The dummy register ZERO may be used to discard an incoming 24-bit ARU word.

GREG: The general purpose register set GREG:={R0, R1, R2, ..., R7} includes all channel specific GPRs without GPRs of neighboring channels.

BAREG: The base address register set BAREG:={R0, R1, ..., R7} ∪ {RS0, RS1, ..., RS7} includes all available GPRs.

In the following, the register sets OREG, XOREG, GREG, WXREG, BAREG and AREG are referred by the instructions. Typically, an operation announces W data bits. Whenever, a register of a register set implements less than W bits, it is assumed that these register bits only define the LSBs of an operation. The missing MSBs are always read and written as zeros.

WLIT: The set WLIT:={0, 1,..., $2^{W}-1$} is a W bit wide literal value used for encoding immediate operands.

ALIT: The set ALIT:={0, 1,..., $2^{14}-1$} is 14-bit wide unsigned literal value used for encoding word wise memory addresses.

AOLIT: The set AOLIT:={-2^{13},..., −1, 0, 1,..., $2^{13}-1$} is 14−bit wide signed literal value used for encoding relative word wise memory address offsets.

ARDLIT: The set ARDLIT:={0, 1,..., $2^{9}-1$} is a 9−bit literal used for ARU read addresses.

AWRLIT: The set AWRLIT:={0, 1,..., 23} is used as ARU write indexes, selecting one of the 24 ARU write addresses.

BALIT: The set BALIT:={0, 1,..., $2^{BAW}-1$} is a BAW bit wide literal used for encoding bus master addresses.

SFTLIT: The set SFTLIT:={0, 1,..., W} is used as literal value for shift instructions.

BWSLIT: The set BWSLIT:={1, 2,..., W} is used as literal value for multiplication and division instructions.

BITLIT: The set BITLIT:={0, 1,..., 15} is a 4−bit literal used for bit indexing.

XBITLIT: The set XBITLIT:={0, 1,..., W−1} is a literal used for bit indexing of register bits.

MSKLIT: The set MSKLIT:={0, 1,..., $2^{16}-1$} is a 16−bit literal used for bit−masking.

BIT SELECTION: The expression VAR[i] represents the i−th bit of a variable VAR.

BIT RANGE SELECTION: The expression VAR[m:n] represents the bit slice of variable VAR that is ranging from bit n to bit m.

MEMORY ADDRESSING: The expression MEM(X) represents the RDW bit wide value at location x (x ∈ ALIT) of the memory. The expression MEM(x)[m:n] represents the bit slice ranging from bit n to m of the RDW bit wide word at memory location x.

ARU ADDRESSING: In the case of ARU reading, the expression ARU(x) represents the 2*W+5 bit wide ARU word of ARU channel at read address x (x ∈ ARDLIT). In the case of ARU writing, the expression ARU(x) represents a 2*W+5 bit wide ARU word that is written to an ARU channel indexed by the index x (x ∈ AWRLIT). The index x selects a single ARU write channel from the pool of the ARU write channels allocated to the MCS submodule. An MCS submodule has 24 dedicated ARU write channels, indexed by values 0 to 23. The expression ARU(x)[m:n] represents the bit slice ranging from bit n to m of the 2*W+5 bit wide ARU word.

BUS MASTER ADDRESSING: In the case of reading/writing from the bus master interface, the expression BUS(x) represents the BDW bit wide data word that is read/written at address x (x ∈ BALIT), whereas x is a word address. The expression BUS(x)[m:n] represents the bit slice ranging from bit n to m of the BDW bit wide data word at the bus.

The individual instructions are decoded by evaluating the bits '0' and '1' at its expected positions, as mentioned in field encoding of the individual instructions below, whereas the leftmost bit indicates the most significant bit and the rightmost bit indicated the lowest significant bit. If the instruction decoder detects an invalid combination of these bits, the corresponding MCS−channel x is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, and no other MCS registers or memory cells are updated. Bit positions marked as '−' are not relevant for the corresponding instruction and they are ignored completely during instruction decoding. The bit position 'a_i', 'b_i', and 'c_i' are used for binary encoding of the instruction arguments A, B, and C. The decimal values that can be determined by adding up the products a_i*2^i, b_i*2^i, and c_i*2^i are associated with instruction's arguments A, B, and C, respectively. The actual decimal values encode the elements of the referred sets defined above. However, if A and B refer to a register set, the calculated decimal values of arguments A and B correspond to the internal register addresses mentioned in table *48 "Internal Register Addresses"* .
==> DG: The debug trace interface does not signalize an instruction fetch at its instruction trace interface.

If a register has an active write protection for an MCS−channel x (configured by register **MCS[i]_REG_PROT**) and an MCS instruction wants to write to such a write protected register, the corresponding MCS−channel is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, and no other MCS registers or memory cells are updated. This behavior is not explicitly mentioned in the instruction descriptions below but it is implemented for all instructions.
==> DG: The debug trace interface signalizes an instruction fetch at its instruction trace interface.

17.9.1 MOVL Instruction

Syntax : MOVL A, C

Encoding : 0001a_3 a_2 a_1 a_0 c_{23} c_{22} c_{21} c_{20} c_{19} c_{18} c_{17} c_{16} c_{15} c_{14} c_{13} c_{12} c_{11} c_{10} c_9 c_8 c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0

Operation : A ← C

Status : Z

Duration : 1 instruction cycle

Description : Transfer literal value C (C ∈ WLIT) to register A (A ∈ OREG).

The zero bit **STA.Z** is set, if the transferred value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.2 MOV Instruction

Syntax : MOV A, B

Encoding : 1010a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0000–a$_4$–b$_4$----------

Operation : A ← B

Status : Z

Duration : 1 instruction cycle

Description : Transfer register B (B ∈ XOREG) to register A (A ∈ XOREG).

The zero bit **STA.Z** is set, if the transferred value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.3 MRD Instruction

Syntax : MRD A, C

Encoding : 1010a$_3$a$_2$a$_1$a$_0$----0001c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$--

Operation : A ← MEM(C)[W–1:0];

MHB ← MEM(C)[RDW–1:W]

Status : Z

Duration : 2 instruction cycles (serial access) but not faster than 1+NPS clock cycles or 1 instruction cycle (parallel access).

Description : Transfer the lower W bits of memory content at location C (C ∈ ALIT) to register A (A ∈ OREG).

The upper RDW–W bits of the memory content at location C are transferred to the MHB register.

The zero bit **STA.Z** is set, if the lower W bits of the transferred value are zero, otherwise the zero bit is cleared.

If the MHB register is selected as destination register A (A ∈ OREG), the bits 0 to RDW–W–1 of the referred memory location are transferred to MHB.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.4 MWR Instruction

Syntax : MWR A, C

Encoding : 1010a$_3$a$_2$a$_1$a$_0$----0010c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$--

Operation : MEM(C)[W–1:0] ← A;

MEM(C)[RDW–1:W] ← MHB

Status : –

Duration : 2 instruction cycles (serial access) but not faster than 1+NPS clock cycles or 1 instruction cycle (parallel access).

Description : Transfer W bit value of register A (A ∈ OREG) together with the MHB register to the memory at location C (C ∈ ALIT).

The W bit value of register A is stored in the LSBs (bit 0 to W–1) of the memory location.

The MHB register is stored in bits W to RDW–W–1 of the referred memory location.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.5 MWRL Instruction

Syntax : MWRL A, C

Encoding : 1010a$_3$a$_2$a$_1$a$_0$----0111c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$--

Operation : MEM(C)[W–1:0] ← A

Status : –

Duration : 3 instruction cycles but not faster than 1+2*NPS clock cycles due to pipeline flushing. No parallel memory access possible.

Description : Transfer W bit value of register A (A ∈ OREG) to memory at location C (C ∈ ALIT).

The W bit value of register A is stored in the LSBs (bit 0 to W−1) of the memory location and the bits W to RDW−W are left unchanged.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

The instruction executes a read modify write operation of the referred memory location.

17.9.6 MRDI Instruction

Syntax : MRDI A, B [, C]

Encoding : 1010a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0011c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$--

Operation : A ← MEM(((B>>2) + C) AND 0x3FFFFF)[W−1:0]

MHB ← MEM(((B>>2) + C) AND 0x3FFFFF)[RDW−1:W]

Status : Z

Duration : 2 instruction cycles (serial access) but not faster than 1+NPS clock cycles or 1 instruction cycle (parallel access).

Description : Transfer the bits 0 to W−1 of a memory location to register A (A ∈ OREG) using indirect addressing.

The upper RDW−W bits of this memory location are transferred to MHB register.

The memory location where to read from, depends on register B (B ∈ GREG) and literal C (C ∈ AOLIT) and it is defined as (((B>>2) + C) AND 0x3FFFFF), where AND is a bitwise AND conjunction and the value of B is treated as a unsigned value.

If the optional operand C is not available in the assembler syntax, the MCS assembler tool generates code with a default value of 0 for operand C.

The zero bit **STA.Z** is set, if the transferred bits 0 to W−1 are zero, otherwise the zero bit is cleared.

If the MHB register is selected as destination register A (A ∈ OREG), the bits 0 to RDW−W−1 of the referred memory location are transferred to MHB.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.7 MRDIO Instruction

Syntax : MRDIO A, B

Encoding : 1010a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$1101−a$_4$−b$_4$----------

Operation : A ← MEM(((B>>2) + (R5>>2)) AND 0x3FFFFF)[W−1:0]

MHB ← MEM(((B>>2) + (R5>>2)) AND 0x3FFFFF)[RDW−1:W]

Status : Z

Duration : 2 instruction cycles (serial access) but not faster than 1+NPS clock cycles or 1 instruction cycle (parallel access).

Description : Transfer the bits 0 to W−1 of a memory location to register A (A ∈ XOREG) using indirect addressing with offset calculation.

The upper RDW−W bits of this memory location are transferred to MHB register.

The memory location where to read from depends on register B (B ∈ BAREG) and register R5 and it is defined as (((B>>2) + (R5>>2)) AND 0x3FFFFF), where AND is a bitwise AND conjunction and the values of B and R5 are treated as unsigned values.

The zero bit **STA.Z** is set, if the transferred bits 0 to W−1 are zero, otherwise the zero bit is cleared.

If the MHB register is selected as destination register A, the bits 0 to RDW−W−1 of the referred memory location are transferred to MHB.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.8 MWRI Instruction

Syntax : MWRI A, B [, C]

Encoding : 1010a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0100c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$--

Operation : MEM(((B>>2) + C) AND 0x3FFFFF)[W-1:0] ← A;

MEM(((B>>2) + C) AND 0x3FFFFF)[RDW-1:W] ← MHB

Status : -

Duration : 2 instruction cycles (serial access) but not faster than 1+NPS clock cycles or 1 instruction cycle (parallel access).

Description : Transfer value of register A (A ∈ OREG) to the LSBs 0 to W-1 of a memory location using indirect addressing.

The MHB register is moved to the bits W to RDW-1 at the same memory location.

If the optional operand C is not available in the assembler syntax, the MCS assembler generates code with a default value of 0 for operand C.

The memory location where to write to, depends on register B (B ∈ GREG) and literal C (C ∈ AOLIT) and it is defined as (((B>>2) + C) AND 0x3FFFFF), where AND is a bitwise AND conjunction and the value of B is treated as a unsigned value.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.9 MWRIO Instruction

Syntax : MWRIO A, B

Encoding : 1010a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$1110-a$_4$-b$_4$----------

Operation : MEM(((B>>2)+ (R5>>2)) AND 0x3FFFFF)[W-1:0] ← A;

MEM(((B>>2) + (R5>>2)) AND 0x3FFFFF)[RDW-1:W] ← MHB

Status : -

Duration : 2 instruction cycles (serial access) but not faster than 1+NPS clock cycles or 1 instruction cycle (parallel access).

Description : Transfer value of register A (A ∈ XOREG) to the LSBs 0 to W-1 of a memory location using indirect addressing with offset calculation.

The MHB register is moved to the bits W to RDW-1 at the same memory location.

The memory location where to write to, depends on register B (B ∈ BAREG) and register R5 and it is defined as (((B>>2) + (R5>>2)) AND 0x3FFFFF), where AND is a bitwise AND conjunction and the values of B and R5 are treated as unsigned values.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.10 MWRIL Instruction

Syntax : MWRIL A, B

Encoding : 1010a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$1000----------

Operation : MEM(B>>2)[W-1:0] ← A;

Status : -

Duration : 3 instruction cycles but not faster than 1+2*NPS clock cycles due to pipeline flushing. No parallel memory access possible.

Description : Transfer W bit value of A (A ∈ OREG) to memory using indirect addressing.

The memory location where to write to is defined by register B (B ∈ GREG).

The W bit value is stored in the LSBs (bit 0 to W-1) of the memory location and the bits W to RDW-1 are left unchanged.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

The instruction executes a read modify write operation of the referred memory location.

17.9.11 POP Instruction

Syntax : POP A

Encoding : 1010a$_3$a$_2$a$_1$a$_0$----0101----------

Operation : A ← MEM(R7>>2)[W-1:0];
MHB ← MEM(R7>>2)[RDW-1:W];
R7 ← R7 - 4;

Status : Z

Duration : 2 instruction cycles (serial access) but not faster than 1+NPS clock cycles or 1 instruction cycle (parallel access).

Description : Transfer the LSBs (bit 0 to W-1) from the top of stack to register A (A ∈ OREG \\ {R7}), followed by decrementing the stack pointer register R7 with the value 4.

The upper bits W to RDW-1 from the top of the stack are transferred to register MHB.

If the MHB register is selected as destination register A (A ∈ OREG \\ {R7}), the bits 0 to RDW-W-1 from the top of the stack are transferred to MHB.

The memory location for the top of the stack is defined by (R7>>2).

The zero bit **STA.Z** is set, if the lower W bit of the transferred value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.12 PUSH Instruction

Syntax : PUSH A

Encoding : 1010a$_3$a$_2$a$_1$a$_0$----0110----------

Operation : R7 ← R7 + 4;
MEM(R7>>2)[W-1:0] ← A;
MEM(R7>>2)[RDW-1:W] ← MHB

Status : -

Duration : 2 instruction cycles (serial access) but not faster than 1+NPS clock cycles or 1 instruction cycle (parallel access).

Description : Increment the stack pointer register R7 with the value 4, followed by transferring a W bit value of operand A (A ∈ OREG \\ {R7}) together with a MHB register to the new top of the stack. The W bit value of A is stored in the bits 0 to W-1 of the memory location.

The content of the MHB register is stored in the bit W to RDW-1 of the memory location.

The memory location for the top of the stack is defined by (R7>>2).

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.13 ARD Instruction

Syntax : ARD A, B, C

Encoding : 1011a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0000-------c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A ← ARU(C)[W-1:0];
B ← ARU(C)[2*W-1:W];
ACB ← ARU(C)[4+2*W:2*W]

Status : CAT, SAT

Duration : Suspends current MCS-channel until ARU transfer finished.

Description : Perform a blocking read access to the ARU and transfer both W bit values received at the ARU port to the registers A and B (A ∈ AREG, B ∈ AREG), whereas A holds the lower W bit ARU word and B holds the upper W bit ARU word.

If A and B refer to the same register, only the upper W bit ARU word is stored and the lower W bit ARU word is discarded.

If any transferred W bit value from the ARU should not be stored in a register, the dummy register ZERO ∈ AREG can be selected in A or B to discard the corresponding ARU data. The binary encoding of the address for the dummy register ZERO can be chosen by an arbitrary value within the range 8 to 15.

The received ARU control bits are stored in the register ACB.

The literal C (C ∈ ARDLIT) defines the ARU address where to read from.

At the beginning of the instruction execution the **STA.CAT** bit is always cleared.
After the execution of the instruction the **STA.SAT** flag is updated in order to show if the transfer was successful (**STA.SAT** = 1) or if the transfer failed (**STA.SAT** = 0) due to a cancellation by the CPU.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.14 ARDI Instruction

Syntax : ARDI A, B

Encoding : 1011a_3 a_2 a_1 a_0 b_3 b_2 b_1 b_0 0100----------

Operation : A ← ARU(R6[8:0])[W-1:0];
B ← ARU(R6[8:0])[2*W-1:W];
ACB ← ARU(R6[8:0])[4+2*W:2*W]

Status : CAT, SAT

Duration : Suspends current MCS-channel until ARU transfer finished.

Description : Perform a blocking read access to the ARU and transfer both W bit values received at the ARU port to the registers A and B (A ∈ AREG, B ∈ AREG), whereas A holds the lower W bit ARU word and B holds the upper W bit ARU word.

If A and B refer to the same register, only the upper W bit ARU word is stored and the lower W bit ARU word is discarded.

If any transferred W bit value from the ARU should not be stored in a register, the dummy register ZERO ∈ AREG can be selected in A or B to discard the corresponding ARU data. The binary encoding of the address for the dummy register ZERO can be chosen by an arbitrary value within the range 8 to 15 .

The received ARU control bits are stored in the register ACB.

The read address is obtained from the bits 0 to 8 of the channels register R6.

At the beginning of the instruction execution the **STA.CAT** bit is always cleared.
After the execution of the instruction the **STA.SAT** flag is updated in order to show if the transfer was successful (**STA.SAT** = 1) or if the transfer failed (**STA.SAT** = 0) due to a cancellation by the CPU.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.15 AWR Instruction

Syntax : AWR A, B, C

Encoding : 1011a_3 a_2 a_1 a_0 b_3 b_2 b_1 b_0 0001----------c_4 c_3 c_2 c_1 c_0

Operation : ARU(C)[W-1:0] ← A;
ARU(C)[2*W-1:W] ← B;
ARU(C)[4+2*W:2*W] ← ACB;

Status : CAT, SAT

Duration : Suspends current MCS-channel until ARU transfer finished.

Description : Perform a blocking write access to the ARU and transfer two W bit values to the ARU port using the registers A and B (A ∈ OREG, B ∈ OREG), whereas A holds the lower W bit ARU word and B holds the upper W bit ARU word.

The ARU control bits are taken from the register ACB.

The literal C (C ∈ AWRLIT) defines an index into the pool of ARU write addresses that are used for writing data. This index is mapped to an ARU write address as shown in column "MCS write index" of table " ARU Write Addresses" in *93 "ARU Write Address Overview"* .

Each MCS submodule has a pool of several write addresses that can be shared between all MCS-channels arbitrarily.

At the beginning of the instruction execution the **STA.CAT** bit is always cleared.
After the execution of the instruction the **STA.SAT** flag is updated in order to show if the transfer was successful (**STA.SAT** = 1) or if the transfer failed (**STA.SAT** = 0) due to a cancellation by the CPU.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

BOSCH

17.9.16 AWRI Instruction

Syntax : AWRI A, B

Encoding : 1011a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0101----------

Operation : ARU(R6[4:0])[W-1:0] ← A;
ARU(R6[4:0])[2*W-1:W] ← B;
ARU(R6[4:0])[4+2*W:2*W] ← ACB;

Status : CAT, SAT

Duration : Suspends current MCS-channel until ARU transfer finished.

Description : Perform a blocking write access to the ARU and transfer two W bit values to the ARU port using the registers A and B (A ∈ OREG, B ∈ OREG), whereas A holds the lower W bit ARU word and B holds the upper W bit ARU word.

The ARU control bits are taken from the register ACB.

The bits 0 to 4 of the register R6 define an index into the pool of ARU write addresses that are used for writing data. This index is mapped to an ARU write address as shown in column "MCS write index" of table "ARU Write Addresses" in *93 "ARU Write Address Overview"* .

Each MCS submodule has a pool of several write addresses that can be shared between all MCS-channels arbitrarily.

At the beginning of the instruction execution the **STA.CAT** bit is always cleared.
After the execution of the instruction the **STA.SAT** flag is updated in order to show if the transfer was successful (**STA.SAT** = 1) or if the transfer failed (**STA.SAT** = 0) due to a cancellation by the CPU.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.17 NARD Instruction

Syntax : NARD A, B, C

Encoding : 1011a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0010-------c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A ← ARU(C)[W-1:0];
B ← ARU(C)[2*W:W];
ACB ← ARU(C)[4+2*W:2*W]

Status : SAT

Duration : Suspends current MCS-channel until the ARU is selecting the MCS-channel.

Description : Perform a non-blocking read access to the ARU trying to transfer both W bit values received at the ARU port to the registers A and B (A ∈ AREG, B ∈ AREG), whereas A holds the lower W bit ARU word, B holds the upper W bit ARU word, and the ACB register holds the received ARU control bits. The literal C (C ∈ ARDLIT) defines the ARU address, where to read from.

Non-blocking ARU read access means that the instruction is suspending the MCS-channel until the ARU scheduler is selecting the requesting MCS-channel. If the transfer finished successfully, the bit SAT of the register **STA** is set and the transferred values are stored in the registers A, B, and ACB. If the transfer failed due to missing data at the requested source, the bit SAT of the register **STA** is cleared and registers A, B, and ACB are not changed.

If A and B refer to the same register, only the upper W bit ARU word is stored and the lower W bit ARU word is discarded.

If any transferred W bit value from the ARU should not be stored in a register, the dummy register ZERO ∈ AREG can be selected in A or B to discard the corresponding ARU data. The binary encoding of the address for the dummy register ZERO can be chosen by an arbitrary value within the range 8 to 15.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.18 NARDI Instruction

Syntax : NARDI A, B

Encoding : 1011a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0011----------

Operation : A ← ARU(R6[8:0])[W-1:0];
B ← ARU(R6[8:0])[2*W-1:W];
ACB ← ARU(R6[8:0])[4+2*W:2*W]

Status : SAT

Duration : Suspends current MCS-channel until the ARU is selecting the MCS-channel.

Description : Perform a non-blocking read access to the ARU trying to transfer both W bit values received at the ARU port to the registers A and B (A ∈ AREG, B ∈ AREG), whereas A holds the lower W bit ARU word, B holds the upper W bit ARU word, and the ACB register holds the received ARU control bits. The read address is obtained from the bits 0 to 8 of the channels register R6.

Non-blocking ARU read access means that the instruction is suspending the MCS-channel until the ARU scheduler is selecting the requesting MCS-channel. If the transfer finished successfully, the bit SAT of the register **STA** is set and the transferred values are stored in the registers A, B, and ACB. If the transfer failed due to missing data at the requested source, the bit SAT of the register **STA** is cleared and registers A, B, and ACB are not changed.

If A and B refer to the same register, only the upper W bit ARU word is stored and the lower 24-bit ARU word is discarded.

If any transferred W bit value from the ARU should not be stored in a register, the dummy register ZERO ∈ AREG can be selected in A or B to discard the corresponding ARU data. The binary encoding of the address for the dummy register ZERO can be chosen by an arbitrary value within the range [15:8].

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.19 BRD Instruction

Syntax : BRD A, C

Encoding : 1010-a_2 a_1 a_0 ----1001c $_{13}$ c $_{12}$ c $_{11}$ c $_{10}$ c $_9$ c $_8$ c $_7$ c $_6$ c $_5$ c $_4$ c $_3$ c $_2$ c $_1$ c $_0$ --

Operation : A ← BUS(C)[W-1:0];
MHB ← BUS(C)[BDW-1:W]

Status : -

Duration : Suspends current MCS-channel if addressed slave inserts at least one wait cycle (e.g. accessing a RAM module) otherwise 1 instruction cycle.

Description : Initiate a read access at the bus master interface using the address C (C ∈ BALIT) and transfer the lower W bits of the received data to register A (A ∈ GREG).

The upper BDW-W bits of the received data are transferred to the MHB register.

If the delay between the BRD instruction and its successor instruction is one or two system clock cycles (e.g. in accelerated scheduling mode) and the successor instruction is reading data resulting from the BRD instruction, a data hazard occurs in the pipeline resulting in a pipeline flush. This means, if very fast program execution is required (e.g. only one task is activated in accelerated scheduling mode) a program sequence like BRD R1, 0x0288; ADD R3, R1; (9 clock cycles) can be accelerated by reformulating the sequence as BRD R1, 0x0288; NOP; NOP; ADD R3, R1; (4 clock cycles).

Since the MHB register is always transferred via AEI bus master, it also figures out another data dependency, which can cause a data hazard resulting in a pipeline flush. Therefore, a sequence like BRD R1, 0x0288; BWR R3, 0x304 (9 clock cycles) could also be optimized by the sequence BRD R1, 0x0288; NOP; NOP; BWR R3, 0x304 (4 clock cycles).

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.20 BWR Instruction

Syntax : BWR A, C

Encoding : 1010-a_2 a_1 a_0 ----1010c $_{13}$ c $_{12}$ c $_{11}$ c $_{10}$ c $_9$ c $_8$ c $_7$ c $_6$ c $_5$ c $_4$ c $_3$ c $_2$ c $_1$ c $_0$ --

Operation : BUS(C)[W-1:0] ← A;
BUS(C)[BDW-1:W] ← MHB

Status : -

Duration : Suspends current MCS-channel if addressed slave inserts at least one wait cycle (e.g. accessing a RAM module) otherwise 1 instruction cycle.

Description : Initiate a write access at the bus master interface using the address C (C ∈ BALIT) and transfer the content of register A (A ∈ GREG) to the bits 0 to W-1 of the bus.

The content of the MHB register is transferred to the bits W to BDW-W-1 of the bus.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.21 BRDI Instruction

Syntax : BRDI A, B

Encoding : 1010-a_2 a_1 a_0 -b_2 b_1 b_0 1011----------

Operation : A ← BUS(B[BAW+1:2])[W−1:0];
MHB ← BUS(B[BAW+1:2])[BDW−1:W]

Status : −

Duration : Suspends current MCS−channel if addressed slave inserts at least one wait cycle (e.g. accessing a RAM module) otherwise 1 instruction cycle.

Description : Initiate a read access at the bus master interface using indirect addressing and transfer the lower W bits of the received data to register A (A ∈ GREG).

The upper BDW−W bits of the received data are transferred to the MHB register.

The address for the transfer is identified by the bits 2 to BAW+1 of register B (B ∈ GREG).

If the delay between the BRDI instruction and its successor instruction is one or two system clock cycles (e.g. in accelerated scheduling mode) and the successor instruction is reading data resulting from the BRDI instruction, a data hazard occurs in the pipeline resulting in a pipeline flush. This means, if very fast program execution is required (e.g. only one task is activated in accelerated scheduling mode) a program sequence like BRDI R1, R6; ADD R3, R1; (9 clock cycles) can be accelerated by reformulating the sequence as BRDI R1, R6; NOP; NOP; ADD R3, R1; (4 clock cycles).

Since the MHB register is always transferred via AEI bus master, it also figures out another data dependency, which can cause a data hazard resulting in a pipeline flush. Therefore, a sequence like BRDI R1, R2; BWRI R3, R4 (9 clock cycles) could also be optimized by the sequence BRDI R1, R2; NOP; NOP; BWRI R3, R4 (4 clock cycles).

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.22 BWRI Instruction

Syntax : BWRI A, B

Encoding : 1010−a_2 a_1 a_0 −b_2 b_1 b_0 1100−−−−−−−−−−−−−−−−

Operation : BUS(B[BAW+1:2])[W−1:0] ← A;
BUS(B[BAW+1:2])[BDW−1:W] ← MHB

Status : −

Duration : Suspends current MCS−channel if the addressed slave inserts at least one wait cycle (e.g. accessing a RAM module) otherwise 1 instruction cycle.

Description : Initiate a write access at the bus master interface using the indirect addressing and transfer the content of register A (A ∈ GREG) to the bits 0 to W−1 of the bus.

The content of the MHB register is transferred to the bits W to BDW−W−1 of the bus.

The address for the transfer is identified by the bits 2 to BAW+1 of register B (B ∈ GREG).

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.23 ADDL Instruction

Syntax : ADDL A, C

Encoding : 0010a_3 a_2 a_1 a_0 c_{23} c_{22} c_{21} c_{20} c_{19} c_{18} c_{17} c_{16} c_{15} c_{14} c_{13} c_{12} c_{11} c_{10} c_9 c_8 c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0

Operation : A ← A + C

Status : Z, CY, N, V

Duration : 1 instruction cycle

Description : Perform addition operation of a register A (A ∈ OREG) with a W bit literal value C (C ∈ WLIT) and store the result in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit **STA.CY** is set, if an unsigned overflow/underflow occurred during addition, otherwise the bit is cleared. An unsigned overflow has occurred when the result of the operation cannot be represented in the interval [0; 2^W −1], assuming that both operands A and C are unsigned values within the interval [0; 2^W −1].

The overflow bit **STA.V** is set, if a signed overflow/underflow occurred during addition, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [−2^{W-1} ; 2^{W-1} −1], assuming that both operands A and C are signed values within the interval [−2^{W-1} ; 2^{W-1} −1].

The negative bit **STA.N** equals the MSB of the operation result, in order to determine if a calculated signed result is negative (**STA.N** =1) or positive (**STA.N** =0), assuming that no overflow/underflow occurred.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.24 ADD Instruction

Syntax : ADD A, B

Encoding : 1100a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0000–a$_4$–b$_4$----------

Operation : A ← A + B

Status : Z, CY, N, V

Duration : 1 instruction cycle

Description : Perform addition operation of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG). The result is stored in the register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit **STA.CY** is set, if an unsigned overflow occurred during addition, otherwise the bit is cleared. An unsigned overflow has occurred when the result of the operation cannot be represented in the interval [0; 2^W-1], assuming that both operands A and B are unsigned values within the interval [0; 2^W-1].

The overflow bit **STA.V** is set, if a signed overflow/underflow occurred during addition, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; $2^{W-1}-1$], assuming that both operands A and B are signed values within the interval [-2^{W-1}; $2^{W-1}-1$].

The negative bit **STA.N** equals the MSB of the operation result, in order to determine if a calculated signed result is negative (**STA.N** =1) or positive (**STA.N** =0), assuming that no overflow/underflow occurred.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.25 ADDC Instruction

Syntax : ADDC A, B

Encoding : 1101a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0110–a$_4$–b$_4$----------

Operation : A ← A + B + CY

Status : Z, CY, N, V

Duration : 1 instruction cycle

Description : Perform addition operation of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG) and the carry flag. The result is stored in the register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit **STA.CY** is set, if an unsigned overflow occurred during addition, otherwise the bit is cleared. An unsigned overflow has occurred when the result of the operation cannot be represented in the interval [0; 2^W-1], assuming that both operands A and B are unsigned values within the interval [0; 2^W-1].

The overflow bit **STA.V** is set, if a signed overflow/underflow occurred during addition, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; $2^{W-1}-1$], assuming that both operands A and B are signed values within the interval [-2^{W-1}; $2^{W-1}-1$].

The negative bit **STA.N** equals the MSB of the operation result, in order to determine if a calculated signed result is negative (**STA.N** =1) or positive (**STA.N** =0), assuming that no overflow/underflow occurred.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.26 SUBL Instruction

Syntax : SUBL A, C

Encoding : 0011a$_3$a$_2$a$_1$a$_0$c$_{23}$c$_{22}$c$_{21}$c$_{20}$c$_{19}$c$_{18}$c$_{17}$c$_{16}$c$_{15}$c$_{14}$c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A ← A − C

Status : Z, CY, N, V

Duration : 1 instruction cycle

Description : Perform subtraction operation of a register A (A ∈ OREG) with a W bit literal value C (C ∈ WLIT). The result is stored in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit **STA.CY** is set, if an unsigned underflow occurred during subtraction, otherwise the bit is cleared. An unsigned underflow has occurred when the result of the operation cannot be represented in the interval [0; $2^{W}-1$], assuming that both operands A and C are unsigned values within the interval [0; $2^{W}-1$].

The overflow bit **STA.V** is set, if a signed overflow/underflow occurred during subtraction, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; $2^{W-1}-1$], assuming that both operands A and C are signed values within the interval [-2^{W-1}; $2^{W-1}-1$].

The negative bit **STA.N** equals the MSB of the operation result, in order to determine if a calculated signed result is negative (**STA.N** =1) or positive (**STA.N** =0), assuming that no overflow/underflow occurred.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.27 SUB Instruction

Syntax : SUB A, B

Encoding : 1100a_3 a_2 a_1 a_0 b_3 b_2 b_1 b_0 0001$-a_4$ $-b_4$ ----------

Operation : A ← A − B

Status : Z, CY, N, V

Duration : 1 instruction cycle

Description : Perform subtraction operation of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG). The result is stored in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit **STA.CY** is set, if an unsigned underflow occurred during subtraction, otherwise the bit is cleared. An unsigned underflow has occurred when the result of the operation cannot be represented in the interval [0; $2^{W}-1$], assuming that both operands A and B are unsigned values within the interval [0; $2^{W}-1$].

The overflow bit **STA.V** is set, if a signed overflow/underflow occurred during subtraction, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; $2^{W-1}-1$], assuming that both operands A and B are signed values within the interval [-2^{W-1}; $2^{W-1}-1$].

The negative bit **STA.N** equals the MSB of the operation result, in order to determine if a calculated signed result is negative (**STA.N** =1) or positive (**STA.N** =0), assuming that no overflow/underflow occurred.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.28 SUBC Instruction

Syntax : SUBC A, B

Encoding : 1101a_3 a_2 a_1 a_0 b_3 b_2 b_1 b_0 0111$-a_4$ $-b_4$ ----------

Operation : A ← A − B − CY

Status : Z, CY, N, V

Duration : 1 instruction cycle

Description : Perform subtraction operation of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG) and the carry flag. The result is stored in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit **STA.CY** is set, if an unsigned underflow occurred during subtraction, otherwise the bit is cleared. An unsigned underflow has occurred when the result of the operation cannot be represented in the interval [0; $2^{W}-1$], assuming that both operands A and B are unsigned values within the interval [0; $2^{W}-1$].

The overflow bit **STA.V** is set, if a signed overflow/underflow occurred during subtraction, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval [-2^{W-1}; $2^{W-1}-1$], assuming that both operands A and B are signed values within the interval [-2^{W-1}; $2^{W-1}-1$].

The negative bit **STA.N** equals the MSB of the operation result, in order to determine if a calculated signed result is negative (**STA.N** =1) or positive (**STA.N** =0), assuming that no overflow/underflow occurred.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.29 NEG Instruction

Syntax : NEG A, B

Encoding : 1100a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0010-a$_4$-b$_4$----------

Operation : A ← − B

Status : Z, N, V

Duration : 1 instruction cycle

Description : Perform negation operation (2's Complement) with an operand B (B ∈ XOREG) and store the result in a register A (A ∈ XOREG).

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The overflow bit **STA.V** is set, if a signed overflow/underflow occurred during subtraction, otherwise the bit is cleared. A signed overflow/underflow has occurred when the result of the operation cannot be represented in the interval $[-2^{W-1}; 2^{W-1}-1]$, assuming that both operands A and B are signed values within the interval $[-2^{W-1}; 2^{W-1}-1]$.

The negative bit **STA.N** equals the MSB of the operation result, in order to determine if a calculated signed result is negative (**STA.N** =1) or positive (**STA.N** =0), assuming that no overflow/underflow occurred.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.30 ANDL Instruction

Syntax : ANDL A, C

Encoding : 0100a$_3$a$_2$a$_1$a$_0$c$_{23}$c$_{22}$c$_{21}$c$_{20}$c$_{19}$c$_{18}$c$_{17}$c$_{16}$c$_{15}$c$_{14}$c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A ← A AND C

Status : Z

Duration : 1 instruction cycle

Description : Perform bitwise AND conjunction of a register A (A ∈ OREG) with a W bit literal value C (C ∈ WLIT) and store the result in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.31 AND Instruction

Syntax : AND A, B

Encoding : 1100a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0011-a$_4$-b$_4$----------

Operation : A ← A AND B

Status : Z

Duration : 1 instruction cycle

Description : Perform bitwise AND conjunction of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG) and store the result in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.32 ORL Instruction

Syntax : ORL A, C

Encoding : 0101a$_3$a$_2$a$_1$a$_0$c$_{23}$c$_{22}$c$_{21}$c$_{20}$c$_{19}$c$_{18}$c$_{17}$c$_{16}$c$_{15}$c$_{14}$c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A ← A OR C

Status : Z

Duration : 1 instruction cycle

Description : Perform bitwise OR conjunction of a register A (A ∈ OREG) with a W bit literal value C (C ∈ WLIT) and store the result in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.33 OR Instruction

Syntax : OR A, B

Encoding : 1100a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0100-a$_4$-b$_4$----------

Operation : A ← A OR B

Status : Z

Duration : 1 instruction cycle

Description : Perform bitwise OR conjunction of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG) and store the result in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.34 XORL Instruction

Syntax : XORL A, C

Encoding : 0110a$_3$a$_2$a$_1$a$_0$c$_{23}$c$_{22}$c$_{21}$c$_{20}$c$_{19}$c$_{18}$c$_{17}$c$_{16}$c$_{15}$c$_{14}$c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A ← A XOR C

Status : Z

Duration : 1 instruction cycle

Description : Perform bitwise XOR conjunction of a register A (A ∈ OREG) with a W bit literal value C (C ∈ WLIT) and store the result in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.35 XOR Instruction

Syntax : XOR A, B

Encoding : 1100a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0101-a$_4$-b$_4$----------

Operation : A ← A XOR B

Status : Z

Duration : 1 instruction cycle

Description : Perform bitwise XOR conjunction of a register A (A ∈ XOREG) with an operand B (B ∈ XOREG) and store the result in register A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.36 SHR Instruction

Syntax : SHR A, C

Encoding : 1100a$_3$a$_2$a$_1$a$_0$----0110-a$_4$---------c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A ← A >> C

Status : Z, CY

Duration : 1 instruction cycle

Description : Perform right shift operation C (C ∈ SFTLIT) times of register A (A ∈ XOREG). The MSBs that are shifted into A are cleared.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit **STA.CY** is updated to the last LSB that is shifted out of the register. If the shift value C is 0 the carry bit **STA.CY** is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.37 SHL Instruction

Syntax : SHL A, C

Encoding : 1100a $_3$a $_2$a $_1$a $_0$----0111-a $_4$---------c $_4$c $_3$c $_2$c $_1$c $_0$

Operation : A ← A << C

Status : Z, CY

Duration : 1 instruction cycle

Description : Perform left shift operation C (C ∈ SFTLIT) times of register A (A ∈ XOREG). The LSBs that are shifted into A are cleared.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The carry bit **STA.CY** is updated to the previous MSB that is shifted out of the register. If the register A contains less than W bits or if C is 0 the carry bit **STA.CY** is always cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.38 ASRU Instruction

Syntax : ASRU A, B

Encoding : 1101a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$0100-a $_4$-b $_4$----------

Operation : A ← A >> B

Status : Z

Duration : 1 instruction cycle

Description : Perform arithmetic unsigned right shift operation, which means that the unsigned operand of register A (A ∈ XOREG) is right shifted B times (B ∈ XOREG). Operand B is also an unsigned type. The MSBs that are shifted into A are cleared.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.39 ASRS Instruction

Syntax : ASRS A, B

Encoding : 1101a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$0101-a $_4$-b $_4$----------

Operation : A ← A >> B

Status : Z

Duration : 1 instruction cycle

Description : Perform arithmetic signed right shift operation, which means that the signed operand of register A (A ∈ XOREG) is right shifted B times (B ∈ XOREG). Operand B is an unsigned type. The operation also performs a sign extension, which means that value of the MSBs that are shifted into A are determined by the MSB of the original operand A.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.40 ASL Instruction

Syntax : ASL A, B

Encoding : 1101a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$0011-a $_4$-b $_4$----------

Operation : A ← A << B

Status : Z, CY, V

Duration : 1 instruction cycle

Description : Perform arithmetic left shift operation for signed and unsigned numbers, which means that the operand of register A (A ∈ XOREG) is left shifted B times (B ∈ XOREG). Operand B is always an unsigned type.

The carry bit **STA.CY** is set, if an unsigned overflow occurred during shifting, otherwise the bit is cleared. An unsigned overflow has occurred if the calculated result A*2 ^{B} cannot be represented in the interval [0; 2 ^{W} -1], assuming that both operands A and B are unsigned values within the interval [0; 2 ^{W} -1].

The overflow bit **STA.V** is set, if a signed overflow/underflow occurred during shifting, otherwise the bit is cleared. A signed overflow/underflow has occurred when the calculated result A*2 ^{B} cannot be represented in the interval [-2 $^{W-1}$; 2 $^{W-1}$ -1], assuming that signed operand A is within the interval [-2 $^{W-1}$; 2 $^{W-1}$ -1] and the unsigned operand B is within the interval [0; 2 $^{W-1}$ -1].

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.41 MULU Instruction

Syntax : MULU A, B[, C]

Encoding : 1100a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$1000-a $_4$-b $_4$-------c $_4$c $_3$c $_2$c $_1$c $_0$

Operation : [[R4,] A] ← A[(C-1):0] * B[(C-1):0]

Status : Z

Duration : 1 instruction cycle

Description : Perform an unsigned multiplication operation of an operand A (A ∈ XOREG if C is less than or equal to W/2; A ∈ XOREG \\ {R4} if C is greater than W/2) with an operand B (B ∈ XOREG). The multiplication is only performed with the bits 0 to C-1 (C ∈ BWSLIT) of both operands A and B and the bits C to W-1 are ignored. If C is less than or equal to W/2, the product of the multiplication is stored in register A and register R4 is left unchanged. If C is greater than W/2, the bits 0 to W-1 are stored in A and the bits W to 2*C-1 are stored in R4. The results stored in the registers are always zero extended to W bits.
If the optional operand C is not specified in the assembler code, the MCS assembler generates code with a default value of W for operand C.
The zero bit **STA.Z** is set, if the calculated product is zero, otherwise the zero bit is cleared.

If the delay between the MULU instruction and its successor instruction is one system clock cycles (e.g. in accelerated scheduling mode) and the successor instruction is either a WURM, WURMX, WURCX, WUCE, BRDI, BWR, or BWRI instruction that is accessing the multiplication result as argument a data hazard in the pipeline occurs resulting in a pipeline flush.
This means, if very fast program execution is required (e.g. only one task is activated in accelerated scheduling mode) a program sequence like MULU R1, R2; BWRI R1, R3; (9 clock cycles) can be accelerated by reformulating the sequence as MULU R1, R2; NOP; BWRI R1, R3; (3 clock cycles).

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.42 MULS Instruction

Syntax : MULS A, B[, C]

Encoding : 1100a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$1001-a $_4$-b $_4$-------c $_4$c $_3$c $_2$c $_1$c $_0$

Operation : [[R4,] A] ← A[(C-1):0] * B[(C-1):0]

Status : Z, N

Duration : 1 instruction cycle

Description : Perform a signed multiplication operation of an operand A (A ∈ XOREG if C is less than or equal to W/2; A ∈ XOREG \\ {R4} if C is greater than W/2) with an operand B (B ∈ XOREG). The multiplication is only performed with the bits 0 to C-1 (C ∈ BWSLIT) of both operands A and B, in which bit C-1 is used as sign bit (-2^(C-1)) and the bits C to W-1 are ignored. If C is less than or equal to W/2, the product of the multiplication is stored in register A and register R4 is left unchanged. If C is greater than W/2, the bits 0 to W-1 are stored in A and the bits W to 2*C-1 are stored in R4. The results stored in the registers are always sign extended to W bits.
If the optional operand C is not specified in the assembler code, the MCS assembler generates code with a default value of W for operand

C.
The zero bit **STA.Z** is set, if the calculated product is zero, otherwise the zero bit is cleared.
The negative bit **STA.N** equals the MSB of the operation result, in order to determine if a calculated signed result is negative (N=1) or positive (N=0).

If the delay between the MULS instruction and its successor instruction is one system clock cycles (e.g. in accelerated scheduling mode) and the successor instruction is either a WURM, WURMX, WURCX, WUCE, BRDI, BWR, or BWRI instruction that is accessing the multiplication result as argument a data hazard in the pipeline occurs resulting in a pipeline flush.
This means, if very fast program execution is required (e.g. only one task is activated in accelerated scheduling mode) a program sequence like MULS R1, R2; BWRI R1, R3; (9 clock cycles) can be accelerated by reformulating the sequence as MULS R1, R2; NOP; BWRI R1, R3; (3 clock cycles).

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.43 DIVU Instruction

Syntax : DIVU A, B[, C]

Encoding : 1100a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$1010–a$_4$–b$_4$–––––––c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : R4 ← A[(C–1):0] – B[(C–1):0] * ⌊A[(C–1):0] / B[(C–1):0]⌋;
A ← ⌊A[(C–1):0] / B[(C–1):0]⌋

Status : CY, Z, ERR

Duration : C instruction cycles but not faster than C+NPS–1 clock cycles due to pipeline flushing.

Description : Perform an unsigned division operation of operand A (A ∈ ((GREG ∪ {RS0, ... , RS7} ∪ {ACB, MHB}) \\ {R4, B})) divided by operand B (B ∈ ((GREG ∪ {RS0, ... , RS7} ∪ {ACB, MHB}) \\ {R4, A})). The division is only performed with the bits 0 to C–1 (C ∈ BWSLIT) of the operands and the remaining bits C to W–1 are ignored. This means that the dynamic range of A and B is defined in the interval [0; 2^C –1]. The integral part of the quotient is stored in the register A and the remainder of the division is stored in register R4. The resulting quotient A and remainder R4 are always zero extended to W bits.
If the optional operand C is not specified in the assembler code, the MCS assembler generates code with a default value of W for operand C.
The zero bit **STA.Z** is set, if the calculated quotient in A is zero, otherwise the zero bit is cleared.
The carry bit **STA.CY** is set, if the calculated remainder in R4 is not zero, otherwise the carry bit is cleared.

If the bits 0 to C–1 of operand B are zero or C is not an element of BWSLIT, the corresponding MCS–channel is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, and no other MCS registers or memory cells are updated.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.44 DIVS Instruction

Syntax : DIVS A, B[, C]

Encoding : 1100a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$1011–a$_4$–b$_4$–––––––c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : R4 ← A[(C–1):0] – B[(C–1):0] * ⌊A[(C–1):0] / B[(C–1):0]⌋;
A ← ⌊A[(C–1):0] / B[(C–1):0]⌋

Status : CY, Z, N, V, ERR

Duration : C + 4 instruction cycles but not faster than C+3+NPS clock cycles due to pipeline flushing.

Description : Perform a signed division operation of operand A (A ∈ ((GREG ∪ {RS0, ... , RS7} ∪ {ACB, MHB}) \\ {R4, B})) divided by operand B (B ∈ ((GREG ∪ {RS0, ... , RS7} ∪ {ACB, MHB}) \\ {R4, A})). The division is only performed with the bits 0 to C–1 (C ∈ BWSLIT) of both operands, in which bit C–1 is used as sign bit (–2^{C-1}) and the bits C to W–1 are ignored. This means that the dynamic range of A[(C–1):0] and B[(C–1):0] is defined in the interval [–2^{C-1}; 2^{C-1} –1]. The integral part of the quotient is stored in the register A and the remainder of the division is stored in register R4. The resulting quotient A and remainder R4 are always sign extended to W bits. The integral part of the quotient is always truncated towards 0. The sign of the remainder is always the same sign as the dividend A[(C–1):0]. The absolute value of the remained is always less than the divisor B[(C–1):0].
If the optional operand C is not specified in the assembler code, the MCS assembler generates code with a default value of W for operand C.
The zero bit **STA.Z** is set, if the calculated quotient in A is zero, otherwise the zero bit is cleared.
The carry bit **STA.CY** is set, if the calculated remainder in R4 is not zero, otherwise the carry bit is cleared.
The overflow bit **STA.V** is set, if the calculated quotient in A cannot be represented in the interval [–2^{W-1}; 2^{W-1} –1], otherwise the overflow bit is cleared.
The negative bit **STA.N** equals the MSB of the quotient, in order to determine if a calculated signed result is negative (**STA.N** =1) or positive (**STA.N** =0).

If the bits 0 to C–1 of operand B are zero or C is not an element of BWSLIT, the corresponding MCS–channel is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, and no other MCS registers or memory cells are updated.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.45 MINU Instruction

Syntax : MINU A, B

Encoding : 1100a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$1100–a $_4$–b $_4$----------

Operation : A ← MIN(A, B)

Status : Z

Duration : 1 instruction cycle

Description : Determine the minimum of an unsigned operand A (A ∈ XOREG) and an unsigned operand B (B ∈ XOREG). If A is less than or equal to B, A is left unchanged. Otherwise, if A is greater than B, the operand B is moved to A.
The zero bit **STA.Z** is set, if the calculated result of A is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.46 MINS Instruction

Syntax : MINS A, B

Encoding : 1100a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$1101–a $_4$–b $_4$----------

Operation : A ← MIN(A, B)

Status : Z

Duration : 1 instruction cycle

Description : Determine the minimum of a signed operand A (A ∈ XOREG) and a signed operand B (B ∈ XOREG). If A is less than or equal to B, A is left unchanged. Otherwise, if A is greater than B, the operand B is moved to A.
The zero bit **STA.Z** is set, if the calculated result of A is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.47 MAXU Instruction

Syntax : MAXU A, B

Encoding : 1100a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$1110–a $_4$–b $_4$----------

Operation : A ← MAX(A, B)

Status : Z

Duration : 1 instruction cycle

Description : Determine the maximum of an unsigned operand A (A ∈ XOREG) and an unsigned operand B (B ∈ XOREG). If A is greater than or equal to B, A is left unchanged. Otherwise, if A is less than B, the operand B is moved to A.
The zero bit **STA.Z** is set, if the calculated result of A is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.48 MAXS Instruction

Syntax : MAXS A, B

Encoding : 1100a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$1111–a $_4$–b $_4$----------

Operation : A ← MAX(A, B)

Status : Z

Duration : 1 instruction cycle

Description : Determine the maximum of a signed operand A (A ∈ XOREG) and a signed operand B (B ∈ XOREG). If A is greater than or equal to B, A is left unchanged. Otherwise, if A is less than B, the operand B is moved to A.
The zero bit **STA.Z** is set, if the calculated result of A is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.49 ATUL Instruction

Syntax : ATUL A, C

Encoding : 0111a$_3$a$_2$a$_1$a$_0$c$_{23}$c$_{22}$c$_{21}$c$_{20}$c$_{19}$c$_{18}$c$_{17}$c$_{16}$c$_{15}$c$_{14}$c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A − C

Status : Z, CY

Duration : 1 instruction cycle

Description : Arithmetic test with an unsigned operand A (A ∈ OREG) and an unsigned W bit literal value C (C ∈ WLIT).

The carry bit **STA.CY** is set if unsigned operand A is less than unsigned literal C.

Otherwise, the carry bit CY of status register STA is cleared if unsigned operand A is greater than or equal to unsigned literal C.

The zero bit **STA.Z** is set, if A equal to C.

Otherwise, the zero bit **STA.Z** is cleared, if A is unequal to C.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.50 ATU Instruction

Syntax : ATU A, B

Encoding : 1101a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0000−a$_4$−b$_4$----------

Operation : A − B

Status : Z, CY

Duration : 1 instruction cycle

Description : Arithmetic Test with an unsigned operand A (A ∈ XOREG) and an unsigned operand B (B ∈ XOREG).

The carry bit **STA.CY** is set if unsigned operand A is less than unsigned operand B.

Otherwise, the carry bit **STA.CY** is cleared if unsigned operand A is greater than or equal to unsigned operand B.

The zero bit **STA.Z** is set, if A equals to B.

Otherwise, the zero bit **STA.Z** is cleared, if A is unequal to B.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.51 ATSL Instruction

Syntax : ATSL A, C

Encoding : 1000a$_3$a$_2$a$_1$a$_0$c$_{23}$c$_{22}$c$_{21}$c$_{20}$c$_{19}$c$_{18}$c$_{17}$c$_{16}$c$_{15}$c$_{14}$c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$

Operation : A − C

Status : Z, CY

Duration : 1 instruction cycle

Description : Arithmetic Test with a signed operand A (A ∈ OREG) and a signed W bit literal value C (C ∈ WLIT).

The carry bit **STA.CY** is set if signed operand A is less than signed literal C.

Otherwise, the carry bit **STA.CY** is cleared if signed operand A is greater than or equal to signed literal C.

The zero bit **STA.Z** is set, if A equals to C.

Otherwise, the zero bit **STA.Z** is cleared, if A is unequal to C.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.52 ATS Instruction

Syntax : ATS A, B

Encoding : 1101a $_{3}$a $_{2}$a $_{1}$a $_{0}$b $_{3}$b $_{2}$b $_{1}$b $_{0}$0001−a $_{4}$−b $_{4}$----------

Operation : A − B

Status : Z, CY

Duration : 1 instruction cycle

Description : Arithmetic Test with a signed operand A (A ∈ XOREG) and a signed operand B (B ∈ XOREG).

The carry bit **STA.CY** is set if signed operand A is less than signed operand B.

Otherwise, the carry bit **STA.CY** is cleared if signed operand A is greater than or equal to signed operand B.

The zero bit **STA.Z** is set, if A equals to B.

Otherwise, the zero bit **STA.Z** is cleared, if A is unequal to B.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.53 BTL Instruction

Syntax : BTL A, C

Encoding : 1001a $_{3}$a $_{2}$a $_{1}$a $_{0}$c $_{23}$c $_{22}$c $_{21}$c $_{20}$c $_{19}$c $_{18}$c $_{17}$c $_{16}$c $_{15}$c $_{14}$c $_{13}$c $_{12}$c $_{11}$c $_{10}$c $_{9}$c $_{8}$c $_{7}$c $_{6}$c $_{5}$c $_{4}$c $_{3}$c $_{2}$c $_{1}$c $_{0}$

Operation : A AND C

Status : Z

Duration : 1 instruction cycle

Description : Bit test of an operand A (A ∈ OREG) with a W bit literal bit mask C (C ∈ WLIT).

The bit test is performed by applying a bitwise logical AND operation with operand A and the bit mask C without storing the result.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.54 BT Instruction

Syntax : BT A, B

Encoding : 1101a $_{3}$a $_{2}$a $_{1}$a $_{0}$b $_{3}$b $_{2}$b $_{1}$b $_{0}$0010−a $_{4}$−b $_{4}$----------

Operation : A AND B

Status : Z

Duration : 1 instruction cycle

Description : Bit test of an operand A (A ∈ XOREG) with an operand B (B ∈ XOREG), whereas usually one of the operands is a register holding a bit mask.

The bit test is performed by applying a bitwise logical AND operation with register A and register B without storing the result.

The zero bit **STA.Z** is set, if the calculated value is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.55 SETB Instruction

Syntax : SETB A, B

Encoding : 1011a $_{3}$a $_{2}$a $_{1}$a $_{0}$b $_{3}$b $_{2}$b $_{1}$b $_{0}$0110−a $_{4}$−b $_{4}$----------

Operation : A[B[4:0]] ←1

Status : Z

Duration : 1 instruction cycle

Description : Set the B[4:0]−th bit (B ∈ XOREG) of an operand A (A ∈ XOREG) to true. Only the bits 0 to 4 of operand B are used as bit index of operand A and the other bits of B are ignored. If the value B[4:0] is greater than or equal to W, the operation of SETB does not modify operand A but the status flag Z is updated.

The instruction SETB performs an implicit read−modify−write operation meaning that the entire content of A is read first and after manipulating the desired bit the entire content of A is written back. This implementation may cause undesired results in special function registers and therefore it should be used carefully.

The zero bit **STA.Z** is set if the modified value of A is zero, otherwise the bit **STA.Z** is cleared. The **STA.Z** bit is set e.g. if the B[4:0]−th bit of A is read−only, its value is zero and all other bits of A are cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.56 CLRB Instruction

Syntax : CLRB A, B

Encoding : 1011a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$0111−a $_4$−b $_4$----------

Operation : A[B[4:0]] ←0

Status : Z

Duration : 1 instruction cycle

Description : Clear the B[4:0]−th bit (B ∈ XOREG) of an operand A (A ∈ XOREG). Only the bits 0 to 4 of operand B are used as bit index of operand A and the other bits of B are ignored. If the value B[4:0] is greater than or equal to W, the operation of CLRB does not modify operand A but the status flag Z is updated.

The instruction CLRB performs an implicit read−modify−write operation meaning that the entire content of A is read first and after manipulating the desired bit the entire content of A is written back. This implementation may cause undesired results in special function registers and therefore it should be used carefully.

The zero bit **STA.Z** is set if the modified value of A is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.57 XCHB Instruction

Syntax : XCHB A, B

Encoding : 1010a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$1111−a $_4$−b $_4$----------

Operation : A[B[4:0]] ↔CY

Status : Z, CY

Duration : 1 instruction cycle

Description : Exchange the B[4:0]−th bit (B ∈ XOREG) of an operand A (A ∈ XOREG) with the **STA.CY** bit in the status register. Only the bits 0 to 4 of operand B are used as bit index of operand A and the other bits of B are ignored. If the value B[4:0] is greater than or equal to W, the operation of XCHB does not modify operand A but the status flag Z is updated and the bit CY is cleared.

The instruction XCHB performs an implicit read−modify−write operation meaning that the entire content of A is read first and after manipulating the desired bit the entire content of A is written back. This implementation may cause undesired results in special function registers and therefore it should be used carefully.

The zero bit **STA.Z** is set if the modified value of A is zero, otherwise the zero bit is cleared.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.58 JMP Instruction

Syntax : JMP C

Encoding : 1110--------0000c $_{13}$c $_{12}$c $_{11}$c $_{10}$c $_9$c $_8$c $_7$c $_6$c $_5$c $_4$c $_3$c $_2$c $_1$c $_0$−−

Operation : MCS[i]_CH[x]_PC.PC ← C << 2

Status : −

Duration : 1 instruction cycle but not faster than NPS clock cycles due to pipeline flushing.

Description : Execute unconditional jump to the memory location C (C ∈ ALIT).

The program counter **MCS[i]_CH[x]_PC.PC** is loaded with literal C.

17.9.59 JBS Instruction

Syntax : JBS A, B, C

Encoding : 1110a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0001c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$--

Operation : MCS[i]_CH[x]_PC.PC ← C << 2 if A[B] is set
MCS[i]_CH[x]_PC.PC ← MCS[i]_CH[x]_PC.PC + 4 if A[B] is clear

Status : -

Duration : 1 instruction cycle but if the jump is executed, it is not faster than NPS clock cycles due to pipeline flushing.

Description : Execute conditional jump to the memory location C (C ∈ ALIT).

The program counter **MCS[i]_CH[x]_PC.PC** is loaded with literal C if the bit at position B (B ∈ BITLIT) of operand A (A ∈ OREG) is set.

Otherwise, if the bit is cleared the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.60 JBC Instruction

Syntax : JBC A, B, C

Encoding : 1110a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0010c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$--

Operation : MCS[i]_CH[x]_PC.PC ← C << 2 if A[B] is clear
MCS[i]_CH[x]_PC.PC ← MCS[i]_CH[x]_PC.PC + 4 if A[B] is set

Status : -

Duration : 1 instruction cycle but if the jump is executed it is not faster than NPS clock cycles due to pipeline flushing.

Description : Execute conditional jump to the memory location C (C ∈ ALIT).

The program counter **MCS[i]_CH[x]_PC.PC** is loaded with literal C if the bit at position B (B ∈ BITLIT) of operand A (A ∈ OREG) is cleared.

Otherwise, if the bit is set the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.61 CALL Instruction

Syntax : CALL C

Encoding : 1110--------0011c$_{13}$c$_{12}$c$_{11}$c$_{10}$c$_9$c$_8$c$_7$c$_6$c$_5$c$_4$c$_3$c$_2$c$_1$c$_0$--

Operation : R7 ← R7 + 4;
MEM(R7>>2) ← MCS[i]_CH[x]_PC.PC + 4;
MCS[i]_CH[x]_PC.PC ← C << 2

Status : -

Duration : 2 instruction cycles (serial access) but not faster than 2*NPS clock cycles or 1 instruction cycle (parallel access).

Description : Call subprogram at memory location C (C ∈ ALIT).

The stack pointer register R7 is incremented by the value 4.

The memory location for the top of the stack is defined by (R7>>2).

After the stack pointer is incremented, the value **MCS[i]_CH[x]_PC.PC** + 4 is transferred to the top of the stack.

The program counter **MCS[i]_CH[x]_PC.PC** is loaded with value (C << 2).

If **MCS[i]_CH[x]_PC.PC** + 4 generates an program counter overflow, the corresponding MCS-channel is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, the stackpointer is not updated and no value is written to the top of the stack and the program counter is not updated with the target branch address.

17.9.62 RET Instruction

Syntax : RET

Encoding : 1110--------0100----------------

Operation : MCS[i]_CH[x]_PC.PC ← MEM(R7>>2) AND 0xFFFFFC;
R7 ← R7 – 4

Duration : 2 instruction cycles (serial access) but not faster than 2*NPS clock cycles or 1 instruction cycle (parallel access).

Description : Return from subprogram.

The program counter **MCS[i]_CH[x]_PC.PC** is loaded with current value on the top of the stack.

Finally, the stack pointer register R7 is decremented by the value 4.

The memory location for the top of the stack is defined by (R7>>2).

17.9.63 JMPI Instruction

Syntax : JMPI

Encoding : 1110--------0101----------------

Operation : MCS[i]_CH[x]_PC.PC ← R6 AND 0xFFFFFC

Status : –

Duration : 1 instruction cycle but not faster than NPS clock cycles due to pipeline flushing.

Description : Execute indirect unconditional jump to the memory location provided in register R6.

The program counter **MCS[i]_CH[x]_PC.PC** is defined as (R6 AND 0xFFFFFC), where AND is a bitwise AND conjunction and the value of R6 is treated as an unsigned value.

17.9.64 JBSI Instruction

Syntax : JBSI A, B

Encoding : 1110a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$0110–a $_4$–b $_4$----------

Operation : MCS[i]_CH[x]_PC.PC ← R6 AND 0xFFFFFC if A[B] is set
MCS[i]_CH[x]_PC.PC ← MCS[i]_CH[x]_PC.PC + 4 if A[B] is clear

Status : –

Duration : 1 instruction cycle but if the jump is executed it is not faster than NPS clock cycles due to pipeline flushing.

Description : Execute indirect conditional jump to the memory location provided in register R6.

The program counter **MCS[i]_CH[x]_PC.PC** is defined as (R6 AND 0xFFFFFC), if the bit at position B (B ∈ XBITLIT) of operand A (A ∈ XOREG) is set, where AND is a bitwise AND conjunction and the value of R6 is treated as an unsigned value.

Otherwise, if the bit at position B of operand A is cleared, the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.65 JBCI Instruction

Syntax : JBCI A, B

Encoding : 1110a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$0111–a $_4$–b $_4$----------

Operation : MCS[i]_CH[x]_PC.PC ← R6 AND 0xFFFFFC if A[B] is clear
MCS[i]_CH[x]_PC.PC ← MCS[i]_CH[x]_PC.PC + 4 if A[B] is set

Status : –

Duration : 1 instruction cycle but if the jump is executed it is not faster than NPS clock cycles due to pipeline flushing.

Description : Execute indirect conditional jump to the memory location provided in register R6.

The program counter **MCS[i]_CH[x]_PC.PC** is defined as (R6 AND 0xFFFFFC), if the bit at position B (B ∈ XBITLIT) of operand A (A ∈ XOREG) is cleared, where AND is a bitwise AND conjunction and the value of R6 is treated as an unsigned value.

Otherwise, if the bit at position B of operand A is set, the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.9.66 CALLI Instruction

Syntax : CALLI

Encoding : 1110--------1000----------------

Operation : R7 ← R7 + 4;
MEM(R7>>2) ← MCS[i]_CH[x]_PC.PC + 4;
MCS[i]_CH[x]_PC.PC ← R6 AND 0xFFFFFC

Status : -

Duration : 2 instruction cycles (serial access) but not faster than 2*NPS clock cycles or 1 instruction cycle (parallel access).

Description : Call subprogram indirectly, where the register R6 is identifying the target memory location.

The stack pointer register R7 is incremented by the value 4.

The memory location for the top of the stack is defined by (R7>>2).

After the stack pointer is incremented, the value **MCS[i]_CH[x]_PC.PC** + 4 is transferred to the top of the stack.

The program counter **MCS[i]_CH[x]_PC.PC** is loaded with (R6 AND 0xFFFFFC), where AND is a bitwise AND conjunction and the value of R6 is treated as an unsigned value.

If **MCS[i]_CH[x]_PC.PC** + 4 generates an program counter overflow, the corresponding MCS-channel is disabled, the bit fields **STA.ERR** and **MCS[i]_ERR.ERR[x]** are set, the interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is triggered, the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is updated, the stackpointer is not updated and no value is written to the top of the stack and the program counter is not updated with the target branch address.

17.9.67 WURM Instruction

Syntax : WURM A, B, C

Encoding : 1111a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$0000c $_{15}$c $_{14}$c $_{13}$c $_{12}$c $_{11}$c $_{10}$c $_9$c $_8$c $_7$c $_6$c $_5$c $_4$c $_3$c $_2$c $_1$c $_0$

Operation : Wait until register match.

Status : CWT

Duration : Suspends current MCS-channel. If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode, the worst case latency for reactivating a prioritized MCS-channel is 2+NPS clock cycles. This is the delay between the match event of the corresponding WURM instruction and the beginning of the following MCS instruction.

Description : Suspend current MCS-channel until the following register match condition occurs:
A = (B AND MASK),
whereas A ∈ OREG, B ∈ OREG, AND is a bitwise AND operation with bitmask MASK. The bits [23:16] of MASK are set to true and the bits [15:0] are copied from the instructions literal C ∈ MSKLIT. If the match condition evaluates to true, the suspended MCS-channel is resumed and the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4 meaning that the MCS-channel continues its program. However, if the match condition is true at the beginning of the instruction execution, the instruction does not suspend the channel and the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.
At the beginning of the instruction execution the **STA.CWT** bit is always cleared. After the execution of the instruction the **STA.CWT** bit is updated in order to show if the instruction finished successfully (**STA.CWT** = 0) or it was canceled by the CPU (**STA.CWT** = 1).

This instruction can be used to wait for one or more trigger events generated by other MCS-channels or the CPU. In this case register B is the trigger register **STRG** , A is a general purpose register holding the bits with the trigger condition to wait for and C is the bitmask that enables trigger bits of interest. The trigger bits can be set by other MCS-channels with a write access (e.g. using a MOVL instruction) to the **STRG** register or the CPU with a write access to the **MCS[i]_STRG** register. The trigger bits are not cleared automatically by hardware after resuming an MCS-channel, but they have to be cleared explicitly with a write access to the register **CTRG** by the MCS-channel or with a write access to the register **MCS[i]_CTRG** by the CPU.

Note:
More than one channel can wait for the same trigger bit to continue.

17.9.68 WURMX Instruction

Syntax : WURMX A, B

Encoding : 1111a $_3$a $_2$a $_1$a $_0$b $_3$b $_2$b $_1$b $_0$0001---b $_4$----------

Operation : Wait until extended register match.

Status : CWT

Duration : Suspends current MCS-channel. If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode, the worst case latency for reactivating a prioritized MCS-channel is 2+NPS clock cycles. This is the delay between the match event of the corresponding WURMX instruction and the beginning of the following MCS instruction.

Description : Suspend current MCS-channel until the following register match condition occurs:
A = B AND R6,
whereas A ∈ OREG, B ∈ WXREG, and AND is a bitwise AND operation. If the match condition evaluates to true, the suspended MCS-channel is resumed and the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4 meaning that the MCS-channel continues its program. However, if the match condition is true at the beginning of the instruction execution, the instruction does not suspend the channel and the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.
At the beginning of the instruction execution the CWT bit in the register STA is always cleared. After the execution of the instruction the CWT bit is updated in order to show if the instruction finished successfully (CWT = 0) or it was canceled by the CPU (CWT = 1).

17.9.69 WURCX Instruction

Syntax : WURCX A, B

Encoding : 1111a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0010---b$_4$----------

Operation : Wait until extended register change.

Status : CWT

Duration : Suspends current MCS-channel. If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode, the worst case latency for reactivating a prioritized MCS-channel is 2+NPS clock cycles. This is the delay between the match event of the corresponding WURCX instruction and the beginning of the following MCS instruction.

Description : Suspend current MCS-channel until the following register change condition occurs:
A != B AND R6,
whereas A ∈ OREG, B ∈ WXREG, and AND is a bitwise AND operation. If the change condition evaluates to true, the suspended MCS-channel is resumed and the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4 meaning that the MCS-channel continues its program. However, if the change condition is true at the beginning of the instruction execution, the instruction does not suspend the channel and the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.
At the beginning of the instruction execution the CWT bit in the register STA is always cleared. After the execution of the instruction the CWT bit is updated in order to show if the instruction finished successfully (CWT = 0) or it was canceled by the CPU (CWT = 1).

The WURCX instruction can be used for observation of volatile registers (e.g. register **DSTAX**) in order to react on status signal changes.

17.9.70 WUCE Instruction

Syntax : WUCE A, B

Encoding : 1111a$_3$a$_2$a$_1$a$_0$b$_3$b$_2$b$_1$b$_0$0011----------

Operation : Wait until cyclic event.

Status : CWT

Duration : Suspends current MCS-channel. If the MCS is configured in Single Prioritization or Multiple Prioritization Scheduling Mode, the worst case latency for reactivating a prioritized MCS-channel is 2+NPS clock cycles. This is the delay between the match event of the corresponding WUCE instruction and the beginning of the following MCS instruction.

Description : Suspend current MCS-channel until a cyclic event compare matches. The meaning of a cyclic event is described in section *3.11.1 "Cyclic Event Compare"* . The WUCE instruction can be used to synchronize an MCS program to a cyclic event generated by a TBU channel. If the event is in the future, the MCS-channel suspends until the event occurs. If the event is in the past, the WUCE instruction is finished immediately.

In order to setup a WUCE instruction correctly, the counting direction of the TBU channel has to be considered. If the TBU channel is counting forward (incrementing), the operand A (A ∈ OREG) must refer the compare value and operand B (B ∈ OREG) must refer the desired TBU counter register (e.g. **TBU_TS0**). On the other hand, if the TBU channel is counting backward (decrementing), the operand A refers to the desired TBU counter register and operand B refers to the compare value.
If the comparison condition evaluates to true, the suspended MCS-channel is resumed and the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4 meaning that the MCS-channel continues its program. However, if the condition is true at the beginning of the instruction execution, the instruction does not suspend the channel and the program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.
At the beginning of the instruction execution the **STA.CWT** is always cleared. After the execution of the instruction the **STA.CWT** bit is updated in order to show if the instruction finished successfully (**STA.CWT** = 0) or it was canceled by the CPU (**STA.CWT** = 1).

17.9.71 NOP Instruction

Syntax : NOP

Encoding : 0000----------

Operation : –

Status : –

Duration : 1 instruction cycle

Description : No operation is performed.

The program counter **MCS[i]_CH[x]_PC.PC** is incremented by the value 4.

17.10 MCS Internal Registers Description

This section describes the MCS internal registers that can be directly addressed with the MCS instruction set. Many of the registers can also be addressed by the CPU but with another Register Label (for details see section *17.11 "MCS Configuration Registers Description"*). Some of the internal registers are also shared between neighboring MCS–channels. The addresses of these internal registers can be obtained from table *48 "Internal Register Addresses"* .

Table 48 Internal Register Addresses

Register Mnemonic	Address
R[y]	0x0+y
RS[y]	0x10+y
STA	0x8
ACB	0x9
CTRG	0xA
STRG	0xB
TBU_TS0	0xC
TBU_TS1	0xD
TBU_TS2	0xE
MHB	0xF
GMI0	0x18
GMI1	0x19
DSTA	0x1A
DSTAX	0x1B
AXIMI	0x1C
MSINT	0x1D
TIOI	0x1E
TIOSL	0x1F
ZERO	0xC

17.10.1 R[y]

Description	General purpose register [y]
Loop	$y = \{n : 0 <= n <= 7\}$
Condition	
View Condition	
Register Reference	*17.11.3 "MCS[i]_CH[x]_R[y]"*

Interface: MCS_INSTR[i]

Name	R[y]
Address	$0x0 + y$
C–Name	

DATA	
Description	Data field of general purpose register

DATA	
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–

Note:
Register R4 is also used as destination register of upper multiplication result from instructions MULU and MULS.

Note:
Register R5 is also used as offset register for the instructions MRDIO and MWRIO.

Note:
Register R6 is also used as a mask register for the instruction WURMX and WURCX.

Note:
Register R6 is also used as address destination register for the instructions JMPI, JBSI, JBCI, and CALLI.

Note:
Register R6 used also as index/address register for indirect ARU addressing instructions.

Note:
Register R7 is also used as stack pointer register, if stack operations are used in the MCS micro program.

17.10.2 RS[y]

Description	Refer to succeeding channel register R[y]
Loop	$y = \{n : 0 <= n <= 7\}$
Condition	
View Condition	
Register Reference	*17.11.3 "MCS[i]_CH[x]_R[y]"*

Interface: MCS_INSTR[i]

Name	RS[y]
Address	$0x10 + y$
C–Name	

DATA	
Description	data field of general purpose register
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

DATA	
Initial value	0
Protect Enable Cond	–
Reset group	–

Note:
The register **RS[y]** physically refers to the internal general purpose register **R[y]** of the succeeding MCS–channel. The successor of MCS–channel T–1 is MCS–channel 0.

17.10.3 STA

Description	Control and status register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	

Interface: MCS_INSTR[i]

Name	STA
Address	$0x8$
C–Name	

EN	
Description	Enable current MCS–channel
Loop	–
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
RW–Coding	0 : Disable current MCS–channel 1 : Enable current MCS–channel

IRQ	
Description	Trigger IRQ
Loop	–
Bit Range	$[1:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

IRQ	
Protect Enable Cond	–
Reset group	–
R-Coding	0 : No interrupt pending in MCS-channel x 1 : Interrupt is pending in MCS-channel x
W-Coding	0 : No action 1 : Trigger interrupt
	Note: An MCS-channel triggers an IRQ by writing value 1 to bit IRQ. Writing a value 0 to this bit does not cancel the IRQ, and thus has no effect. **Note:** This bit field physically refers to bit 0 of the register **MCS[i]_CH[x]_IRQ_NOTIFY** . **Note:** The IRQ bit can only be cleared by CPU, by writing a 1 to the corresponding **MCS[i]_CH[x]_IRQ_NOTIFY** register (see chapter **MCS[i]_CH[x]_IRQ_NOTIFY**). **Note:** An MCS-channel can read the IRQ bit in order to determine the current state of the IRQ handling. The MCS-channel reads a value 1 if an IRQ was released but not cleared by CPU. If an MCS-channel reads a value 0 no IRQ was released or it has been cleared by CPU. **Note:** If NPS > 5 and an MCS program triggers the IRQ (e.g. by MOVL STA, 0x2) the actual interrupt event is delayed by NPS-5 clock cycles, which means that an immediate read of the interrupt notify flag (e.g. by MOV R2, STA) may signalize the state of the IRQ bit before the trigger.

ERR	
Description	Set Error Signal
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : No Error occurred 1 : Error occurred
W-Coding	0 : No action 1 : Set Error bit
	The ERR bit of an MCS-channel reflects an Error status that may be caused by one of the following conditions: ▽

ERR	
	△ ▶ MCS-channel sets the **STA.ERR** bit by software (e.g. with instruction MOVL STA, 0x4) ▶ ECC RAM Error occurred while accessing the connected RAM (also disables the MCS-channel by clearing bit **STA.EN** and the first error occurred updates bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID**) ▶ Decoding an instruction with an invalid opcode (also disables the MCS-channel by clearing bit **STA.EN** and the first error occurred updates bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID**) ▶ A memory address range overflow occurred (also disables the MCS-channel by clearing bit **STA.EN** and the first error occurred updates bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID**) ▶ Division by zero resulting from a DIVU or DIVS instruction (also disables the MCS-channel by clearing bit **STA.EN** and the first error occurred updates bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID**). ▶ MCS-channel wants to write to a GPR that is write protected by register **MCS[i]_REG_PROT** (also disables the MCS-channel by clearing bit **STA.EN** and the first error occurred updates bit field **MCS[i]_CTRL_STAT.-ERR_SRC_ID**). ▶ MCS-channel wants to write to a protected memory range defined by an address range protector (ARP) of the submodule CCM (also disables the MCS-channel by clearing bit **STA.EN** and the first error occurred updates bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID**). ▶ MCS-channel performs an invalid AEI bus master access while the bit field **MCS[i]_CTRL_STAT.HLT_AEI-M_ERR** is set (also disables the MCS-channel by clearing bit **STA.EN** and the first error occurred updates bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID**) **Note:** If the **STA.ERR** bit is set due to a memory address range overflow any read or write access to the RAM is blocked. **Note:** If the GTM includes a MON submodule, the error is signaled via port *MCS_ERR* to the MON module. **Note:** An MCS-channel can set the error bit by writing value 1 to bit **STA.ERR** . Writing a value 0 to this bit does not cancel the error signal, and thus has no effect. In addition, writing a value 1 to **STA.ERR** always triggers the **STA.ERR** interrupt, independently from the current state of the error signal. **Note:** The ERR bit can only be cleared by CPU, by writing a 1 to the **MCS[i]_ERR** register (see chapter **MCS[i]_E-RR**). **Note:** An MCS-channel can read the ERR bit in order to determine the current state of the error signal. The MCS--channel reads a value 1 if an error occurred previously, but not cleared by CPU. If an MCS-channel reads a value 0 no error was set or it has been cleared by CPU.

MCA	
Description	MON Activity signaling from MCS-channel
Loop	-
Bit Range	$[3:3]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-
W-Coding	0 : No action. 1 : Activity is signaled to submodule MON via port *MCS_CH_MCA* .

MCA	
	Note: When this bit is set the corresponding MCS-channel bit field in the MON submodule's register **MON_ACTIVITY_MCS[j]** is set (see chapter **MON_ACTIVITY_MCS[j]**).

CY	
Description	Carry bit
Loop	-
Bit Range	[4 : 4]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-
R-Coding	0 : Carry bit is cleared 1 : Carry bit is set
	The carry bit is updated by several arithmetic and logic instructions. In arithmetic operations, the carry bit indicates an unsigned under/overflow.

Z	
Description	Zero bit
Loop	-
Bit Range	[5 : 5]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-
R-Coding	0 : Zero bit is cleared 1 : Zero bit is set
	The zero bit is updated by several arithmetic, logic and data transfer instructions to indicate a result of zero.

V	
Description	Overflow bit
Loop	-
Bit Range	[6 : 6]
Access Type	R
Volatile	True

V	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : Overflow bit is cleared 1 : Overflow bit is set
	The overflow bit is updated by arithmetic instructions in order to indicate a signed under/overflow.

N	
Description	Negative bit
Loop	–
Bit Range	[7 : 7]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : Negative bit is cleared 1 : Negative bit is set
	The negative bit is updated by arithmetic instructions in order to indicate a negative result.

CAT	
Description	Cancel ARU transfer bit
Loop	–
Bit Range	[8 : 8]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : No cancellation request for ARU transfer 1 : CPU requested cancellation for ARU transfer

CAT	
	Note: This bit is always cleared at the beginning of the execution of a blocking ARU instruction. A cancellation request can be applied by a write request to register **MCS[i]_CAT** . If the MCS program needs to detect a cancellation request it should evaluate the bit **STA.CAT** immediately after the cancelled instruction.

CWT	
Description	Cancel WURM instruction bit
Loop	–
Bit Range	[9 : 9]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : No cancellation request for waiting instruction 1 : CPU requested cancellation for waiting instruction
	Note: This bit is always cleared at the beginning of the execution of a WURM, WURMX, or WURCX instruction. A cancellation request can be applied by a write request to register **MCS[i]_CWT** . If the MCS program needs to detect a cancellation request it should evaluate the bit **STA.CWT** immediately after the cancelled instruction.

SAT	
Description	Successful ARU transfer bit
Loop	–
Bit Range	[10 : 10]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : Latest ARU data transfer failed 1 : Latest ARU data transfer finished successfully
	Note: This bit is always updated after the execution of ARU instructions in order to show if the ARU data transfer was successful or not. In the case of non–blocking ARU instructions (NARD, NARDI) a cleared **STA.SAT** flag signalizes that the data source has no data available and in the case of blocking ARU instructions (ARD, ARDI, AWR, AWRI) a cleared **STA.SAT** flag signalizes that the data transfer was canceled by the CPU.

Attention:

Writing to bits of the register **STA** with instructions that do implicitly a read–modify–write operation (e.g. "ANDL **STA** , 0xFFFFFE", "OR **STA** , **R[0]** ", or "SETB **STA** , **R[0]** ") is dangerous, since writing back the possibly modified content of the read access (which reflects status

information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instructions MOVL).

17.10.4 ACB

Description	ARU control bit register
Loop	
Condition	
View Condition	
Register Reference	*17.11.4 "MCS[i]_CH[x]_ACB"*

Interface: MCS_INSTR[i]

Name	ACB
Address	$0x9$
C-Name	

ACB[k]	
Description	ARU Control bit [k]
Loop	$k = \{n : 0 <= n <= 4\}$
Bit Range	$[k : k]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-
	Note: This bit is updated by each ARU read access and its value is sent to ARU by each ARU write access on bit 48 of the ARU word.

17.10.5 CTRG

Description	Clear trigger bit register
Loop	
Condition	
View Condition	
Register Reference	*17.11.13 "MCS[i]_CTRG"*

Interface: MCS_INSTR[i]

Name	CTRG
Address	$0xA$
C-Name	

TRG[m]	
Description	Trigger bit [m]

TRG[m]	
Loop	$m = \{n : 0 <= n <= 7\}$
Bit Range	$[m : m]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	MCS[i]_CTRL_STAT.EN_TIM_FOUT == 0 0 : Trigger bit is 0 1 : Trigger bit is 1
R-Coding	MCS[i]_CTRL_STAT.EN_TIM_FOUT == 1 0 : Input signal *F_OUT* [m:m] of TIM instance i channel 0 is low 1 : Input signal *F_OUT* [m:m] of TIM instance i channel 0 is high
W-Coding	0 : No action 1 : Clear trigger bit

TRG[n]	
Description	Trigger bit [n].
Loop	$n = \{n : 8 <= n <= 15\}$
Bit Range	$[n : n]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	MCS[i]_CTRL_STAT.EN_TIM_FOUT == 0 0 : Trigger bit is 0 1 : Trigger bit is 1
R-Coding	MCS[i]_CTRL_STAT.EN_TIM_FOUT == 1 0 : Input signal *F_OUT* [n-8:n-8] of TIM instance i+1 channel 0 is low 1 : Input signal *F_OUT* [n-8:n-8] of TIM instance i+1 channel 0 is high
W-Coding	0 : No action 1 : Clear trigger bit

TRG[o]	
Description	Trigger bit [o]
Loop	$o = \{n : 16 <= n <= 23\}$
Bit Range	$[o : o]$

TRG[o]	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : Trigger bit is 0 1 : Trigger bit is 1
W–Coding	0 : No action 1 : Clear trigger bit

Note:

The trigger bits TRGx are accessible by all MCS–channels as well as the CPU. Setting a trigger bit can be performed with the **STRG** register, in the case of an MCS–channel or the **MCS[i]_STRG** register in the case of the CPU. Clearing a trigger bit can be performed with the **CTRG** register, in the case of an MCS–channel or the **MCS[i]_CTRG** register in the case of the CPU. Trigger bits can be used for signalizing specific events to MCS–channels or the CPU. An MCS–channel suspended with a WURM instruction can be resumed by setting the appropriate trigger bit.

Note:

Besides setting the trigger bits with register **STRG** / **MCS[i]_STRG** , the k–th trigger bit TRGk (with k < 16) can also be set by the external capture event that is enabled by the k–th bit of register **CCM[i]_EXT_CAP_EN** . If bit k is disabled, the k–th trigger bit TRGk can only be set by MCS or CPU.

Note:

The result of a read access to this register differs in dependency of the bit field **MCS[i]_CTRL_STAT.EN_TIM_FOUT** .

Attention:

Writing to bits of the register **CTRG** with instructions that do implicitly a read–modify–write operation (e.g. "OR **CTRG** , R0" or "SETB **CTRG** , R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instructions MOVL).

17.10.6 STRG

Description	Set trigger bit register
Loop	
Condition	
View Condition	
Register Reference	*17.11.14 "MCS[i]_STRG"*

Interface: MCS_INSTR[i]

Name	STRG
Address	$0xB$
C–Name	

TRG[k]	
Description	Trigger bit [k]
Loop	$k = \{n : 0 <= n <= 23\}$
Bit Range	$[k : k]$
Access Type	RW
Volatile	True

TRG[k]	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : Trigger is cleared 1 : Trigger is set
W–Coding	0 : No action 1 : Set trigger bit

Note:

The trigger bits TRGx are accessible by all MCS–channels as well as the CPU. Setting a trigger bit can be performed with the **STRG** register, in the case of an MCS–channel or the **MCS[i]_STRG** register in the case of the CPU. Clearing a trigger bit can be performed with the **CTRG** register, in the case of an MCS–channel or the **MCS[i]_CTRG** register in the case of the CPU. Trigger bits can be used for signalizing specific events to MCS–channels or the CPU. An MCS–channel suspended with a WURM instruction can be resumed by setting the appropriate trigger bit.

Note:

Besides setting the trigger bits with register **STRG** / **MCS[i]_STRG** , the k–th trigger bit TRGk (with k < 16) can also be set by the external capture event that is enabled by the k–th bit of register **CCM[i]_EXT_CAP_EN** . If bit k is disabled, the k–th trigger bit TRGk can only be set by MCS or CPU.

Attention:

Writing to bits of the register **STRG** with instructions that do implicitly a read–modify–write operation (e.g. "OR **STRG** , R0" or "SETB **STRG** , R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instructions MOVL).

17.10.7 TBU_TS0

Description	TBU timestamp TBU_CH0_BASE register; input port CCM[i]_TBU_TS0
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	TBU_TS0
Address	$0xC$
C–Name	

TS	
Description	Current TBU time stamp 0
Loop	–
Bit Range	$[23 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

TS	
Initial value	0
Protect Enable Cond	-
Reset group	-

17.10.8 TBU_TS1

Description	TBU timestamp TBU_CH[1]_BASE register; input port CCM[i]_TBU_TS1
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	TBU_TS1
Address	$0xD$
C-Name	

TS	
Description	Current TBU time stamp 1
Loop	-
Bit Range	$[23 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-

17.10.9 TBU_TS2

Description	TBU timestamp TBU_CH[2]_BASE register; input port CCM[i]_TBU_TS2
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	TBU_TS2
Address	$0xE$

C-Name	

TS	
Description	Current TBU time stamp 2
Loop	-
Bit Range	$[23:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-

17.10.10 MHB

Description	Memory high bits register
Loop	
Condition	
View Condition	
Register Reference	*17.11.5 "MCS[i]_CH[x]_MHB"*

Interface: MCS_INSTR[i]

Name	MHB
Address	$0xF$
C-Name	

DATA	
Description	High Byte of a memory transfer
Loop	-
Bit Range	$[7:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-

17.10.11 GMI0

Description	GTM module interrupt 0 register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$

Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	GMI0
Address	$0x18$
C-Name	

TIM_CH[x]_IRQ	
Description	IRQ signal MCS_TIM[i]_IRQ[x:x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$i < \mathrm{NTIM}$
Initial value	0
Protect Enable Cond	-
Reset group	-
R-Coding	0 : IRQ signal *MCS_TIM[i]_IRQ* [x:x] is cleared 1 : IRQ signal *MCS_TIM[i]_IRQ* [x:x] is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

TOM_CH_2X[x]_IRQ	
Description	IRQ signal MCS_TOM[i]_IRQ[x:x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$i < \mathrm{NTOM}$
Initial value	0
Protect Enable Cond	-
Reset group	-
R-Coding	0 : IRQ signal *MCS_TOM[i]_IRQ* [x:x] is cleared 1 : IRQ signal *MCS_TOM[i]_IRQ* [x:x] is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

ATOM_CH[x]_IRQ	
Description	IRQ signal MCS_ATOM[i]_IRQ[x:x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$i < \mathrm{NATOM}$
Initial value	0
Protect Enable Cond	-
Reset group	-
R-Coding	0 : IRQ signal *MCS_ATOM[i]_IRQ* [x:x] is cleared 1 : IRQ signal *MCS_ATOM[i]_IRQ* [x:x] is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

Attention:

Writing to bits of the register **GMI0** with instructions that do implicitly a read-modify-write operation (e.g. "OR **GMI0** , R0" or "SETB **GMI0** , R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instructions MOVL).

Note:

A write access with value 1 to the bit fields of register GMI0 normally generates a single cycle pulse on its HW_clear signals. Multiple write accesses with value 1 to the same bit field can widen the pulse to more than one clock cycle, which means that newly incoming interrupts at the same time will be lost.

17.10.12 GMI1

Description	GTM module interrupt 1 register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	GMI1
Address	$0x19$
C-Name	

MCS_IP1_CH[x]_IRQ	
Description	IRQ signal MCS_NMCS[i]_IRQ[x:x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True

MCS_IP1_CH[x]_IRQ	
Multithread	False
ModifiedWriteValue	–
Condition	$i + 1 < \mathrm{NMCS}$
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : IRQ signal *MCS_NMCS[i]_IRQ* [x:x] is cleared 1 : IRQ signal *MCS_NMCS[i]_IRQ* [x:x] is set
W–Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

TTA_IP1_CH[x]_IRQ	
Description	IRQ signal MCS_TIMATOM[i]_IRQ[x:x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$i + 1 < \max(\max(\mathrm{NTIM}, \mathrm{NTOM}), \mathrm{NATOM})$
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : IRQ signal *MCS_TIMATOM[i]_IRQ* [x:x] is cleared 1 : IRQ signal *MCS_TIMATOM[i]_IRQ* [x:x] is set
W–Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

MCS0_CH[x]_IRQ	
Description	Channel IRQ of MCS0
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : IRQ signal *MCS_MCS[0]_IRQ* [x:x] is cleared 1 : IRQ signal *MCS_MCS[0]_IRQ* [x:x] is set

MCS0_CH[x]_IRQ	
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

Attention:

Writing to bits of the register **GMI1** with instructions that do implicitly a read-modify-write operation (e.g. "OR **GMI1** , R0" or "SETB **GMI1** , R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instructions MOVL).

Note:

A write access with value 1 to the bit fields of register GMI1 normally generates a single cycle pulse on its HW_clear signals. Multiple write accesses with value 1 to the same bit field can widen the pulse to more than one clock cycle, which means that newly incoming interrupts at the same time will be lost.

17.10.13 DSTA

Description	DPLL status register
Loop	$i = \{n : 0 <= n <= 0\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	DSTA
Address	$0x1A$
C-Name	

STA_T	
Description	Status Trigger FSM
Loop	-
Bit Range	$[7 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-
	Actual status of DPLL Trigger FSM. The description of the FSM states can be found in chapter **DPLL_STA** .

STA_S	
Description	Status State FSM
Loop	-
Bit Range	$[15 : 8]$
Access Type	R

STA_S	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
	Actual status of DPLL State FSM. The description of the FSM states can be found in chapter **DPLL_STA** .

TASI	
Description	Trigger Active Slope Interrupt
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal, Signal *DPLL_TASI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_TASI* of DPLL is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

TISI	
Description	Trigger Inactive Slope Interrupt
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal, Signal *DPLL_TISI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_TISI* of DPLL is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

SASI	
Description	State Active Slope Interrupt
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : IRQ signal, Signal *DPLL_SASI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_SASI* of DPLL is set
W–Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

SISI	
Description	State Inactive Slope Interrupt
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : IRQ signal, Signal *DPLL_SISI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_SISI* of DPLL is set
W–Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

CDTI	
Description	Calculation of trigger duration interrupt
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–

CDTI	
R-Coding	0 : IRQ signal, Signal *DPLL_CDTI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_CDTI* of DPLL is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

Note:
This register is only implemented in MCS instance 0. In other MCS instances, a read access always returns 0 and a write access is always ignored.

Attention:
Writing to bits of the register **DSTA** with instructions that do implicitly a read-modify-write operation (e.g. "OR **DSTA** , R0" or "SETB **DSTA** , R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instructions MOVL).

17.10.14 DSTAX

Description	DPLL extended status register
Loop	$i = \{n : 0 <= n <= 0\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	DSTAX
Address	$0x1B$
C-Name	

STA_FLAG_T	
Description	DPLL status trigger flag
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	-
R-Coding	0 : The binary coded DPLL state (**DPLL_STA.STA_T**) for *TRIGGER* did not leave the state defined in **DPLL_STA_MASK.STA_NOTIFY_T** 1 : The binary coded DPLL state (**DPLL_STA.STA_T**) for *TRIGGER* left the state defined in **DPLL_STA_MASK.STA_NOTIFY_T**
W-Coding	0 : No action 1 : Clear flag

STA_FLAG_T	
	DPLL status trigger flag as described in bit field definition **DPLL_STA_FLAG.STA_FLAG_T** (see chapter **DPLL_STA_FLAG**).

STA_FLAG_S	
Description	DPLL status state flag
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : The binary coded DPLL state (**DPLL_STA.STA_S**) for *STATE* did not leave the state defined in **DPLL_STA_MASK.STA_NOTIFY_S** 1 : The binary coded DPLL state (**DPLL_STA.STA_S**) for *STATE* left the state defined in **DPLL_STA_MASK.STA_NOTIFY_S**
W-Coding	0 : No action 1 : Clear flag
	DPLL status state flag as described in bit field definition **DPLL_STA_FLAG.STA_FLAG_S** (see chapter **DPLL_STA_FLAG**).

INC_CNT1_FLAG	
Description	DPLL INC_CNT1 Flag
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : The counter value for *SUB_INC1* (**DPLL_INC_CNT1.INC_CNT1**) did not leave the state in which it is equal to the value stored in **DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY** 1 : The counter value for *SUB_INC1* (**DPLL_INC_CNT1.INC_CNT1**) left the state in which it is equal to the value stored in **DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY**
W-Coding	0 : No action 1 : Clear flag
	DPLL status state flag as described in bit field definition **DPLL_STA_FLAG.INC_CNT1_FLAG** (see chapter **DPLL_STA_FLAG**).

INC_CNT2_FLAG	
Description	DPLL INC_CNT2 Flag
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : The counter value for *SUB_INC2* (**DPLL_INC_CNT2.INC_CNT2**) did not leave the state in which it is equal to the value stored in **DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY** 1 : The counter value for *SUB_INC2* (**DPLL_INC_CNT2.INC_CNT2**) left the state in which it is equal to the value stored in **DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY**
W-Coding	0 : No action 1 : Clear flag
	DPLL status state flag as described in bit field definition **DPLL_STA_FLAG.INC_CNT2_FLAG** (see chapter **D-PLL_STA_FLAG**).

TASI	
Description	Trigger Active Slope Interrupt
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal, Signal *DPLL_TASI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_TASI* of DPLL is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

TISI	
Description	Trigger Inactive Slope Interrupt
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–

TISI	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal, Signal *DPLL_TISI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_TISI* of DPLL is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

SASI	
Description	State Active Slope Interrupt
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal, Signal *DPLL_SASI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_SASI* of DPLL is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

SISI	
Description	State Inactive Slope Interrupt
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal, Signal *DPLL_SISI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_SISI* of DPLL is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

CDTI	
Description	Calculation of trigger duration interrupt

CDTI	
Loop	–
Bit Range	$[20:20]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal, Signal *DPLL_CDTI* of DPLL is cleared 1 : IRQ signal, Signal *DPLL_CDTI* of DPLL is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

Note:
This register is only implemented in MCS instance 0. In other MCS instances, a read access always returns 0 and a write access is always ignored.

Attention:
Writing to bits of the register **DSTAX** with instructions that do implicitly a read-modify-write operation (e.g. "OR **DSTAX** , R0" or "SETB **DSTAX** , R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instructions MOVL).

17.10.15 AXIMI

Description	AXI Master interrupt register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	$i < \text{NAXIM}$
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	AXIMI
Address	$0x1C$
C-Name	

AEIM_AXIM_IRQ[x]	
Description	IRQ signal of AXI bus master slot [x]
Loop	$x = \{n : 0 <= n <= \text{AXIM} - 1\}$
Bit Range	$[x:x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–

AEIM_AXIM_IRQ[x]	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : IRQ signal *AEIM_AXIM_IRQ* [x:x] is cleared 1 : IRQ signal *AEIM_AXIM_IRQ* [x:x] is set
W–Coding	0 : No Action 1 : Trigger signal *AEIM_AXIM_IRQ_CLEAR* [x:x]

Attention:

Writing to bits of the register **AXIMI** with instructions that do implicitly a read–modify–write operation (e.g. "OR **AXIMI** , R0" or "SETB **AXIMI** , R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instructions MOVL).

17.10.16 MSINT

Description	MCS shared interrupt register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	MSINT
Address	$0x1D$
C–Name	

S_IRQ[k]	
Description	Shared IRQ signal of current MCS instance
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[k : k]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	–
R–Coding	0 : IRQ signal *S_IRQ* [k:k] is cleared 1 : IRQ signal *S_IRQ* [k:k] is set
W–Coding	0 : No Action 1 : Trigger IRQ signal *S_IRQ* [k:k]

The interrupt signals of register MSINT are shared among all available MCS-channels. An MCS-channel can trigger an individual IRQ by setting the corresponding bit and the state of an IRQ can be obtained by reading the corresponding bit. Clearing of an MCS shared interrupt signal can be performed by a write access to the register **MCS[i]_SINT_IRQ_NOTIFY** .

Note:

In order to use the shared interrupt signals for the detection of events from external peripheral modules, the shared interrupts can also be cleared or set from outside of the GTM.

Attention:

Writing to bits of the register **MSINT** with instructions that do implicitly a read-modify-write operation (e.g. "OR **MSINT** , R0" or "SETB **MSINT** , R0") is dangerous, since writing back the possibly modified content of the read access (which reflects status information) may cause undesirable results. A secure way for writing to bits of this register is to use instructions that do not read the content (e.g. use instruction MOV).

17.10.17 TIOI

Description	TIO interrupt register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	TIOI
Address	$0x1E$
C-Name	

G0_CH[c]_IRQ	
Description	IRQ signal MCS_TIO[i]_G[0]_IRQ[c:c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c : c]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NTIO\_CH8} > 0$
Initial value	0
Protect Enable Cond	-
Reset group	-
R-Coding	0 : IRQ signal *MCS_TIO[i]_G[0]_IRQ* [c:c] is cleared 1 : IRQ signal *MCS_TIO[i]_G[0]_IRQ* [c:c] is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

G1_CH[c]_IRQ	
Description	IRQ signal MCS_TIO[i]_G[1]_IRQ[c:c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[8 + c : 8 + c]$

G1_CH[c]_IRQ	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIO\_CH8} > 1$
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal *MCS_TIO[i]_G[1]_IRQ* [c:c] is cleared 1 : IRQ signal *MCS_TIO[i]_G[1]_IRQ* [c:c] is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

G2_CH[c]_IRQ	
Description	IRQ signal MCS_TIO[i]_G[2]_IRQ[c:c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[16 + c : 16 + c]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIO\_CH8} > 2$
Initial value	0
Protect Enable Cond	–
Reset group	–
R-Coding	0 : IRQ signal *MCS_TIO[i]_G[2]_IRQ* [c:c] is cleared 1 : IRQ signal *MCS_TIO[i]_G[2]_IRQ* [c:c] is set
W-Coding	0 : No Action 1 : Trigger *HW_CLEAR* signal on the connected IRQs

17.10.18 TIOSL

Description	TIO signal level register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: MCS_INSTR[i]

Name	TIOSL
Address	$0x1F$

C-Name	

G0_CH[c]_S	
Description	Signal Level TIO_G[0]_S_OUT[c:c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c : c]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NTIO\_CH8} > 0$
Initial value	$\mathrm{TIO\_OUT\_RST}$
Protect Enable Cond	-
Reset group	-
R-Coding	0 : Signal *TIO_G[0]_S_OUT* of instance i channel c is low 1 : Signal *TIO_G[0]_S_OUT* of instance i channel c is high

G1_CH[c]_S	
Description	Signal Level TIO_G[1]_S_OUT[c:c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[8 + c : 8 + c]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NTIO\_CH8} > 1$
Initial value	$\mathrm{TIO\_OUT\_RST}$
Protect Enable Cond	-
Reset group	-
R-Coding	0 : Signal *TIO_G[1]_S_OUT* of instance i channel c is low 1 : Signal *TIO_G[1]_S_OUT* of instance i channel c is high

G2_CH[c]_S	
Description	Signal Level TIO_G[2]_S_OUT[c:c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[16 + c : 16 + c]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NTIO\_CH8} > 2$

G2_CH[c]_S	
Initial value	TIO_OUT_RST
Protect Enable Cond	–
Reset group	–
R-Coding	0 : Signal *TIO_G[2]_S_OUT* of instance i channel c is low 1 : Signal *TIO_G[2]_S_OUT* of instance i channel c is high

Note:
This register reflects information of TIO sampling registers *27.7.1 "TIO[i]_S"*

17.11 MCS Configuration Registers Description

The MCS configuration registers of the MCS module are accessible by the AEI bus interface. Some of these registers physically refer to MCS Internal registers that are mapped to the AEI. Details can be found in the individual register descriptions.

17.11.1 MCS[i]_CH[x]_CTRL

Description	MCS[i] channel x control register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_CTRL
Address	$0x20000 * i + 0x100 * x + 0x2020$
C-Name	GTM.CLS[i].MCS.CH[x].CTRL

Interface: MCS[i]

Name	MCS[i]_CH[x]_CTRL
Address	$0x100 * x + 0x2020$
C-Name	

EN	
Description	Enable/Disable Request of MCS-channel x
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	0
Protect Enable Cond	MCS[i]_CTRL_STAT.RAM_RST == 1
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync

EN	
R-Coding	0 : MCS-channel x is disabled 1 : MCS-channel x is enabled
W-Coding	0 : Request for disabling MCS-channel x 1 : Request for enabling MCS-channel x
	Note: Enabling or disabling of an MCS-channel is synchronized to the ending of an instruction and thus it may take several clock cycles, e.g. active memory transfers or pending WURM instruction have to be finished before disabling the MCS-channel. The current internal state of a channel can be obtained by reading the bit EN. **Note:** To disable an MCS-channel reliably the **STA.EN** bit should be cleared followed by setting the **STA.CAT** and **STA.CWT** bit in order to cancel any pending waiting or ARU instructions. **Note:** The EN bit is write protected during RAM reset phase.

IRQ	
Description	Interrupt state
Loop	-
Bit Range	[1 : 1]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : No interrupt pending in MCS-channel x 1 : Interrupt is pending in MCS-channel x
	Note: This bit is read only and it physically refers to the internal IRQ state.

ERR	
Description	Error state
Loop	-
Bit Range	[2 : 2]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

ERR	
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : No error signal pending in MCS-channel x 1 : Error signal is pending in MCS-channel x
	Note: This bit is read only and it physically refers to the internal error state.

CY	
Description	Carry bit state
Loop	-
Bit Range	[4 : 4]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : Carry bit is cleared 1 : Carry bit is set
	Note: This bit is read only and it physically refers to the internal carry flag **STA.CY** .

Z	
Description	Zero bit state
Loop	-
Bit Range	[5 : 5]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : Zero bit is cleared 1 : Zero bit is set

Z	
	Note: This bit is read only and it physically refers to the internal zero flag **STA.Z** .

V	
Description	Overflow bit state
Loop	–
Bit Range	[6 : 6]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : Overflow bit is cleared 1 : Overflow bit is set
	Note: This bit is read only and it physically refers to the internal overflow flag **STA.V** .

N	
Description	Negative bit state
Loop	–
Bit Range	[7 : 7]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : Negative bit is cleared 1 : Negative bit is set
	Note: This bit is read only and it physically refers to the internal negative flag **STA.N** .

CAT	
Description	Cancel ARU transfer state
Loop	–

CAT	
Bit Range	[8 : 8]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : No cancellation request for ARU transfer 1 : CPU requested cancellation for ARU transfer
	Note: This bit is always cleared at the beginning of the execution of a blocking ARU instruction. A cancellation request can be applied by a write request to register **MCS[i]_CAT** . If the MCS program needs to detect a cancellation request it should evaluate the bit **STA.CAT** immediately after the cancelled instruction. **Note:** This bit is read only and it physically refers to the internal cancel ARU transfer status flag **STA.CAT** .

CWT	
Description	Cancel WURM instruction state
Loop	–
Bit Range	[9 : 9]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : No cancellation request for waiting instruction 1 : CPU requested cancellation for waiting instruction
	Note: This bit is always cleared at the beginning of the execution of a WURM, WURMX, or WURCX instruction. A cancellation request can be applied by a write request to register **MCS[i]_CWT** . If the MCS program needs to detect a cancellation request it should evaluate the bit CWT immediately after the cancelled instruction. **Note:** This bit is read only and it physically refers to the internal cancel waiting instruction status flag **STA.CWT** .

SAT	
Description	Successful ARU transfer bit
Loop	–

SAT	
Bit Range	$[10 : 10]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
	Note: This bit is read only and it physically refers to the internal state of the ARU transfer status flag **STA.SAT** .

17.11.2 MCS[i]_CH[x]_PC

Description	MCS[i] channel x program counter register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_PC
Address	$0x20000 * i + 0x100 * x + 0x20E0$
C-Name	GTM.CLS[i].MCS.CH[x].PC

Interface: MCS[i]

Name	MCS[i]_CH[x]_PC
Address	$0x100 * x + 0x20E0$
C-Name	

PC	
Description	Current Program Counter
Loop	–
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	–
Initial value	$4 * x$

PC	
Protect Enable Cond	MCS[i]_CH[x]_CTRL.EN == 1
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
	Note: The program counter is a byte wise address register pointer to the MCS memory. **Note:** The program counter is only writable if the corresponding MCS−channel is disabled. The bits 0 and 1 are always written as zeros. **Note:** The actual width of the program counter depends on the MCS configuration. The actual width is RAW+USR+2 bits meaning that only the bits 0 to RAW+USR+1 are available and the other bits (RAW+USR+2 to 31) are reserved.

17.11.3 MCS[i]_CH[x]_R[y]

Description	MCS[i] channel x general purpose register [y]
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}; y = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_R0
Address	$0x20000 * i + 0x100 * x + 0x2000$
C−Name	GTM.CLS[i].MCS.CH[x].R0
Name	MCS[i]_CH[x]_R1
Address	$0x20000 * i + 0x100 * x + 0x2004$
C−Name	GTM.CLS[i].MCS.CH[x].R1
Name	MCS[i]_CH[x]_R2
Address	$0x20000 * i + 0x100 * x + 0x2008$
C−Name	GTM.CLS[i].MCS.CH[x].R2
Name	MCS[i]_CH[x]_R3
Address	$0x20000 * i + 0x100 * x + 0x200C$
C−Name	GTM.CLS[i].MCS.CH[x].R3
Name	MCS[i]_CH[x]_R4
Address	$0x20000 * i + 0x100 * x + 0x2010$
C−Name	GTM.CLS[i].MCS.CH[x].R4
Name	MCS[i]_CH[x]_R5
Address	$0x20000 * i + 0x100 * x + 0x2014$
C−Name	GTM.CLS[i].MCS.CH[x].R5
Name	MCS[i]_CH[x]_R6
Address	$0x20000 * i + 0x100 * x + 0x2018$

C-Name	GTM.CLS[i].MCS.CH[x].R6
Name	MCS[i]_CH[x]_R7
Address	$0x20000 * i + 0x100 * x + 0x201C$
C-Name	GTM.CLS[i].MCS.CH[x].R7

Interface: MCS[i]

Name	MCS[i]_CH[x]_R0
Address	$0x100 * x + 0x2000$
C-Name	
Name	MCS[i]_CH[x]_R1
Address	$0x100 * x + 0x2004$
C-Name	
Name	MCS[i]_CH[x]_R2
Address	$0x100 * x + 0x2008$
C-Name	
Name	MCS[i]_CH[x]_R3
Address	$0x100 * x + 0x200C$
C-Name	
Name	MCS[i]_CH[x]_R4
Address	$0x100 * x + 0x2010$
C-Name	
Name	MCS[i]_CH[x]_R5
Address	$0x100 * x + 0x2014$
C-Name	
Name	MCS[i]_CH[x]_R6
Address	$0x100 * x + 0x2018$
C-Name	
Name	MCS[i]_CH[x]_R7
Address	$0x100 * x + 0x201C$
C-Name	

DATA	
Description	Data of general purpose register R[y]
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

DATA	
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync

Note:
This register is the same as described in *17.10.1 "R[y]"* .

17.11.4 MCS[i]_CH[x]_ACB

Description	MCS[i] channel x ARU control Bit register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_ACB
Address	$0x20000 * i + 0x100 * x + 0x2024$
C-Name	GTM.CLS[i].MCS.CH[x].ACB

Interface: MCS[i]

Name	MCS[i]_CH[x]_ACB
Address	$0x100 * x + 0x2024$
C-Name	

ACB[k]	
Description	ARU Control bit [k]
Loop	$k = \{n : 0 <= n <= 4\}$
Bit Range	$[k : k]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync

Note:
This register is the same as described in *17.10.4 "ACB"* .

17.11.5 MCS[i]_CH[x]_MHB

Description	MCS[i] channel x memory high byte register

Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_MHB
Address	$0x20000 * i + 0x100 * x + 0x203C$
C-Name	GTM.CLS[i].MCS.CH[x].MHB

Interface: MCS[i]

Name	MCS[i]_CH[x]_MHB
Address	$0x100 * x + 0x203C$
C-Name	

DATA	
Description	Data of memory high bit register MHB
Loop	-
Bit Range	$[7 : 0]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync

Note:
This register is the same as described in *17.10.10 "MHB"* .

17.11.6 MCS[i]_CH[x]_IRQ_NOTIFY

Description	MCS[i] channel x interrupt notification register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_IRQ_NOTIFY
Address	$0x20000 * i + 0x100 * x + 0x20E4$

C–Name	GTM.CLS[i].MCS.CH[x].IRQ_NOTIFY

Interface: MCS[i]

Name	MCS[i]_CH[x]_IRQ_NOTIFY
Address	$0x100 * x + 0x20E4$
C–Name	

MCS_IRQ	
Description	Interrupt request by MCS–channel x
Loop	–
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R–Coding	0 : No interrupt raised 1 : Interrupt was raised
W–Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU and MCS write access with value '1'. A read access leaves the bit unchanged. **Note:** By writing a '1' to this register, the bit field **STA.IRQ** is also cleared.

ERR_IRQ	
Description	MCS–channel x ERR interrupt.
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R–Coding	0 : No interrupt raised 1 : Interrupt was raised

ERR_IRQ	
W-Coding	0 : No action 1 : Clear interrupt
	Note: If the **STA.ERR** is triggered the **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** will also be set. **Note:** This bit will be cleared on a CPU and MCS write access with value '1'. A read access leaves the bit unchanged. **Attention:** The **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** flag must be considered as a functional interrupt that reflects MCS internal error states. This IRQ must not be confused with the global EIRQ functionality as described in section *3.12 "GTM-IP Interrupt Concept"* .

17.11.7 MCS[i]_CH[x]_IRQ_EN

Description	MCS[i] channel x interrupt enable register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_IRQ_EN
Address	$0x20000 * i + 0x100 * x + 0x20E8$
C-Name	GTM.CLS[i].MCS.CH[x].IRQ_EN

Interface: MCS[i]

Name	MCS[i]_CH[x]_IRQ_EN
Address	$0x100 * x + 0x20E8$
C-Name	

MCS_IRQ_EN	
Description	MCS-channel x MCS[i]_CH[x]_IRQ_NOTIFY.MCS_IRQ interrupt enable
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync

MCS_IRQ_EN	
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

ERR_IRQ_EN	
Description	MCS-channel x MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ interrupt enable
Loop	-
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable interrupt 1 : Enable interrupt
	Attention: The **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** interrupt must be considered as a functional interrupt that reflects MCS internal error states. This IRQ must not be confused with the global EIRQ functionality as described in section *3.12 "GTM-IP Interrupt Concept"* .

17.11.8 MCS[i]_CH[x]_IRQ_FORCINT

Description	MCS[i] channel x force interrupt register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_IRQ_FORCINT
Address	$0x20000 * i + 0x100 * x + 0x20EC$
C-Name	GTM.CLS[i].MCS.CH[x].IRQ_FORCINT

Interface: MCS[i]

Name	MCS[i]_CH[x]_IRQ_FORCINT
Address	$0x100 * x + 0x20EC$
C-Name	

TRG_MCS_IRQ	
Description	Trigger the bit MCS[i]_CH[x]_IRQ_NOTIFY.MCS_IRQ by software

TRG_MCS_IRQ	
Loop	–
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	RF_PROT && bitand(CLS[i]_AEI_ARB_WDATA, 5) != 0
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **MCS[i]_CH[x]_IRQ_NOTIFY.MCS_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_ERR_IRQ	
Description	Trigger the bit MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ by software
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	RF_PROT && bitand(CLS[i]_AEI_ARB_WDATA, 5) != 0
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** **Attention:** The **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** interrupt must be considered as a functional interrupt that reflects MCS internal error states. This IRQ must not be confused with the global EIRQ functionality as described in section *3.12 "GTM-IP Interrupt Concept"* .

17.11.9 MCS[i]_CH[x]_IRQ_MODE

Description	MCS[i] channel x IRQ mode configuration register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CH[x]_IRQ_MODE
Address	$0x20000 * i + 0x100 * x + 0x20F0$
C-Name	GTM.CLS[i].MCS.CH[x].IRQ_MODE

Interface: MCS[i]

Name	MCS[i]_CH[x]_IRQ_MODE
Address	$0x100 * x + 0x20F0$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	-
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

17.11.10 MCS[i]_CH[x]_EIRQ_EN

Description	MCS[i] channel x error interrupt enable register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	$MCS[i]\_CLK\_ENABLE == 1$

Interface: CPU

Name	MCS[i]_CH[x]_EIRQ_EN
Address	$0x20000 * i + 0x100 * x + 0x20F4$
C-Name	GTM.CLS[i].MCS.CH[x].EIRQ_EN

Interface: MCS[i]

Name	MCS[i]_CH[x]_EIRQ_EN
Address	$0x100 * x + 0x20F4$
C-Name	

MCS_EIRQ_EN	
Description	MCS[i]_CH[x]_EIRQ_EN.MCS_EIRQ_EN: MCS-channel x MCS_EIRQ error interrupt enable
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable error interrupt 1 : Enable error interrupt

ERR_EIRQ_EN	
Description	MCS-channel x ERR_EIRQ error interrupt enable
Loop	-
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
RW-Coding	0 : Disable error interrupt 1 : Enable error interrupt

ERR_EIRQ_EN	
	Note: By enabling the **MCS[i]_CH[x]_EIRQ_EN.ERR_EIRQ_EN** flag the functional interrupt **MCS[i]_CH[x]_IRQ_NOTIFY.ERR_IRQ** is mapped to the global EIRQ functionality as described in section *3.12 "GTM-IP Interrupt Concept"* .

17.11.11 MCS[i]_CTRL_STAT

Description	MCS[i] control and status register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CTRL_STAT
Address	$0x20000 * i + 0x2F00$
C-Name	GTM.CLS[i].MCS.CTRL_STAT

Interface: MCS[i]

Name	MCS[i]_CTRL_STAT
Address	$0x2F00$
C-Name	

SCD_MODE	
Description	Select MCS scheduling mode
Loop	-
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Accelerated Scheduling 0b01 : Round Robin Scheduling 0b10 : Single Priority Scheduling 0b11 : Multiple Priority Scheduling

SCD_CH	
Description	Channel selection for scheduling algorithm
Loop	-

SCD_CH	
Bit Range	[11 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	MCS-channel identifier used by several scheduling modes in different ways. Details about its usage can be found in section *17.3 "Scheduling"* . **Note:** The actual width of the bit field **MCS[i]_CTRL_STAT.SCD_CH** is calculated as ceil(log2(T+1)). Unused MSBs are reserved and read as zero.

RAM_RST	
Description	RAM reset bit
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	–
Initial value	0
Protect Enable Cond	RF_PROT || MCS[i]_CH[0]_CTRL.EN || MCS[i]_CH[1]_CTRL.EN || MCS[i]_CH[2]_CTRL.EN || MCS[i]_CH[3]_CTRL.EN || MCS[i]_CH[4]_CTRL.EN || MCS[i]_CH[5]_CTRL.EN || MCS[i]_CH[6]_CTRL.EN || MCS[i]_CH[7]_CTRL.EN
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No RAM reset is active 1 : MCS currently resets RAM content
W-Coding	0 : No action 1 : Initiate RAM reset
	Note: The RAM reset initializes the memory content with zeros. RAM access and enabling of MCS-channels is disabled during active RAM reset. **Note:** This bit is only writable if the bit **GTM_CTRL.RF_PROT** is cleared and all MCS-channels are disabled.

ERR_SRC_ID	
Description	Error source identifier
Loop	–
Bit Range	[22 : 20]
Access Type	R

ERR_SRC_ID	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0b000 : No HW generated Error occurred 0b001 : Detected ECC error 0b010 : Detected memory overflow 0b011 : Detected invalid opcode 0b100 : Divide by zero 0b101 : Invalid register write access to GPR from write protected channel 0b110 : Invalid memory write access to protected memory region 0b111 : Invalid AEI bus master access
	Note: This register is updated once, if an error was detected by the MCS. The register is set to its initial value 0b000 after setting **MCS[i]_ERR.ERR[x]** = 1 for any channel x. If multiple errors occur, **MCS[i]_CTRL_STAT.ERR_SRC_ID** is holding the first type of error which has occurred.

EN_TIM_FOUT	
Description	Enable routing of TIM[i]_CH[x]_F_OUT signal.
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	NTIM > 0
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Read access to register **CTRG** / **MCS[i]_CTRG** provides state of the internal trigger registers 1 : Read access to register **CTRG** / **MCS[i]_CTRG** provides state of the external signal *F_OUT* of TIM instance i channel x

EN_HVD	
Description	Enable Modified Harvard architecture.
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

EN_HVD	
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Modified Harvard architecture is disabled 1 : Modified Harvard architecture is enabled

HLT_AEIM_ERR	
Description	Halt on AEI bus master error.
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Ignore invalid AEI bus master access 1 : Halt MCS–channel on invalid AEI bus master access
	Note: If the register **MCS[i]_CTRL_STAT.HLT_AEIM_ERR** is set and an MCS–channel x is executing an invalid bus master access, the MCS–channel x is halted and the error behavior as described in section *17.5 "AEI Bus Master Interface"* is applied. **Note:** If the bus master is accessing a slave that does not insert wait cycles (e.g. register access) it takes two additional clock cycles until the MCS–channel is halted. Within that time span the MCS–channel can continue with its program execution, depending on the selected scheduling mode. **Note:** The registers **CCM[i]_AEIM_STA.AEIM_XPT_STA** and **CCM[i]_AEIM_STA.AEIM_XPT_ADDR** of the GTM submodule CCM are always updated on the first invalid AEI bus master access, independent of the state of **MCS[i]_CTRL_STAT.HLT_AEIM_ERR** .

17.11.12 MCS[i]_REG_PROT

Description	MCS[i] write protection register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_REG_PROT
Address	$0x20000 * i + 0x2F1C$

C–Name	GTM.CLS[i].MCS.REG_PROT

Interface: MCS[i]

Name	MCS[i]_REG_PROT
Address	$0x2F1C$
C–Name	

WPROT[x]	
Description	Register Write Protection of MCS–channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * x + 1 : 2 * x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b00 : No register write protection activated 0b01 : Predecessor MCS–channel cannot write to its **RS[y]** registers 0b10 : Current MCS–channel cannot write to its **R[y]** registers

Note:

The predecessor channel of MCS–channel 0 is MCS–channel T–1.

Note:

If an MCS–channel x is writing to a general purpose register that is write protected by register **MCS[i]_REG_PROT** the bit **STA.ERR** is set, the MCS–channel x is halted and the **MCS[i]_CTRL_STAT.ERR_SRC_ID** bit field is updated.

Note:

This register is only writable if the bit **GTM_CTRL.RF_PROT** is cleared.

17.11.13 MCS[i]_CTRG

Description	MCS[i] clear trigger control register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CTRG
Address	$0x20000 * i + 0x2E28$
C–Name	GTM.CLS[i].MCS.CTRG

Interface: MCS[i]

Name	MCS[i]_CTRG
Address	$0x2E28$
C-Name	

TRG[m]	
Description	Trigger bit [m].
Loop	$m = \{n : 0 <= n <= 7\}$
Bit Range	$[m : m]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	MCS[i]_CTRL_STAT.EN_TIM_FOUT == 0 0 : Trigger bit is 0 1 : Trigger bit is 1
R-Coding	MCS[i]_CTRL_STAT.EN_TIM_FOUT == 1 0 : Input signal *F_OUT* [m:m] of TIM instance i channel 0 is low 1 : Input signal *F_OUT* [m:m] of TIM instance i channel 0 is high
W-Coding	0 : No action 1 : Clear trigger bit

TRG[n]	
Description	Trigger bit [n].
Loop	$n = \{n : 8 <= n <= 15\}$
Bit Range	$[n : n]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	modify
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	MCS[i]_CTRL_STAT.EN_TIM_FOUT == 0 0 : Trigger bit is 0 1 : Trigger bit is 1
R-Coding	MCS[i]_CTRL_STAT.EN_TIM_FOUT == 1 0 : Input signal *F_OUT[n-8:n-8]* of TIM instance i+1 channel 0 is low 1 : Input signal *F_OUT[n-8:n-8]* of TIM instance i+1 channel 0 is high

TRG[n]	
W-Coding	0 : No action 1 : Clear trigger bit

TRG[o]	
Description	Trigger bit [o]
Loop	$o = \{n : 16 <= n <= 23\}$
Bit Range	$[o : o]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Trigger bit is 0 1 : Trigger bit is 1
W-Coding	0 : No action 1 : Clear trigger bit

Note:

The trigger bits TRGx (with x < 24) are accessible by all MCS-channels as well as the CPU. Setting a trigger bit can be performed with the **STRG** register, in the case of an MCS-channel or the **MCS[i]_STRG** register in the case of the CPU. Clearing a trigger bit can be performed with the **CTRG** register, in the case of an MCS-channel or the **MCS[i]_CTRG** register in the case of the CPU. Trigger bits can be used for signalizing specific events to MCS-channels or the CPU. An MCS-channel suspended with a WURM instruction can be resumed by setting the appropriate trigger bit.

Note:

Besides setting the trigger bits with register **STRG** / **MCS[i]_STRG** , the k-th trigger bit TRGk (with k < 16) can also be set by the external capture event that is enabled by the k-th bit of register **CCM[i]_EXT_CAP_EN** . If bit k is disabled, the k-th trigger bit TRGk can only be set by MCS or CPU.

Note:

In the scheduling modes Accelerated Scheduling and Round Robin Scheduling, a write access to **MCS[i]_CTRG** may take up to T + 1 clock cycles, since the write access is scheduled to the next CPU time slot determined by the MCS scheduler. In the modes Single Prioritization Scheduling and Multiple Prioritization Scheduling, no upper limit access time for a write access to **MCS[i]_CTRG** can be guaranteed. The High Prioritized tasks have to be disabled in order to guarantee fast write access to **MCS[i]_CTRG** .

Note:

The result of a read access to this register differs depending on the bit field **MCS[i]_CTRL_STAT.EN_TIM_FOUT** .

17.11.14 MCS[i]_STRG

Description	MCS[i] set trigger control register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_STRG

Address	$0x20000 * i + 0x2E2C$
C-Name	GTM.CLS[i].MCS.STRG

Interface: MCS[i]

Name	MCS[i]_STRG
Address	$0x2E2C$
C-Name	

TRG[k]	
Description	Trigger bit [k].
Loop	$k = \{n : 0 <= n <= 23\}$
Bit Range	$[k : k]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Trigger bit is 0 1 : Trigger bit is 1
W-Coding	0 : No action 1 : Set trigger bit

Note:

The trigger bits are accessible by all MCS-channels as well as the CPU. Setting a trigger bit can be performed with the **STRG** register, in the case of an MCS-channel or the **MCS[i]_STRG** register in the case of the CPU. Clearing a trigger bit can be performed with the **CTRG** register, in the case of an MCS-channel or the **MCS[i]_CTRG** register in the case of the CPU. Trigger bits can be used for signalizing specific events to MCS-channels or the CPU. An MCS-channel suspended with a WURM instruction can be resumed by setting the appropriate trigger bit.

Note:

Besides setting the trigger bits **STRG.TRG[k]** / **MCS[i]_STRG.TRG[k]** , the k-th trigger bit (with k < 16) can also be set by the external capture event that is enabled by the k-th bit of register **CCM[i]_EXT_CAP_EN** . If bit k is disabled, the k-th trigger bit TRGk can only be set by MCS or CPU.

Note:

In the scheduling modes Accelerated Scheduling and Round Robin Scheduling, a write access to **MCS[i]_STRG** may take up to T + 1 clock cycles, since the write access is scheduled to the next CPU time slot determined by the MCS scheduler. In the modes Single Prioritization Scheduling and Multiple Prioritization Scheduling, no upper limit access time for a write access to **MCS[i]_STRG** can be guaranteed. The High Prioritized tasks have to be disabled in order to guarantee fast write access to **MCS[i]_STRG** .

17.11.15 MCS[i]_SINT_IRQ_NOTIFY

Description	MCS[i] shared interrupt notification register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_SINT_IRQ_NOTIFY
Address	$0x20000 * i + 0x2F20$
C-Name	GTM.CLS[i].MCS.SINT_IRQ_NOTIFY

Interface: MCS[i]

Name	MCS[i]_SINT_IRQ_NOTIFY
Address	$0x2F20$
C-Name	

S_IRQ[k]	
Description	Shared interrupt [k] notify flag.
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[k : k]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU and MCS write access with value '1'. A read access leaves the bit unchanged.

Note:
This register physically refers to the internal MCS register **MSINT** .

17.11.16 MCS[i]_SINT_IRQ_EN

Description	MCS[i] shared interrupt enable register
Loop	$i = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_SINT_IRQ_EN
Address	$0x20000 * i + 0x2F24$
C-Name	GTM.CLS[i].MCS.SINT_IRQ_EN

Interface: MCS[i]

Name	MCS[i]_SINT_IRQ_EN
Address	$0x2F24$
C-Name	

S_IRQ[k]_EN	
Description	Shared interrupt [k]
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[k : k]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

17.11.17 MCS[i]_SINT_IRQ_FORCINT

Description	MCS[i] force shared interrupt register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_SINT_IRQ_FORCINT
Address	$0x20000 * i + 0x2F28$
C-Name	GTM.CLS[i].MCS.SINT_IRQ_FORCINT

Interface: MCS[i]

Name	MCS[i]_SINT_IRQ_FORCINT
Address	$0x2F28$
C-Name	

TRG_S_IRQ[k]	
Description	Trigger the bit MCS[i]_SINT_IRQ_NOTIFY.S_IRQ[k] by software
Loop	$k = \{n : 0 <= n <= 7\}$
Bit Range	$[k : k]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 7, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No status
W–Coding	0 : No action 1 : Force setting of **MCS[i]_SINT_IRQ_NOTIFY.S_IRQ[k]**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

17.11.18 MCS[i]_SINT_IRQ_MODE

Description	MCS[i] shared interrupt mode configuration register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_SINT_IRQ_MODE
Address	$0x20000 * i + 0x2F2C$
C–Name	GTM.CLS[i].MCS.SINT_IRQ_MODE

Interface: MCS[i]

Name	MCS[i]_SINT_IRQ_MODE
Address	$0x2F2C$
C–Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	–
Bit Range	$[1 : 0]$

IRQ_MODE	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

17.11.19 MCS[i]_HBP[h]_CTRL

Description	MCS[i] hardware break point h control register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; h = \{n : 0 <= n <= \text{NHBP} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_HBP[h]_CTRL
Address	$0x20000 * i + 0x20 * h + 0x2F40$
C-Name	GTM.CLS[i].MCS.HBP[h].CTRL

Interface: MCS[i]

Name	MCS[i]_HBP[h]_CTRL
Address	$0x20 * h + 0x2F40$
C-Name	

EN_CH[x]	
Description	Enable h-th hardware break point for channel x.
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

EN_CH[x]	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable hardware break point h for MCS–channel x. 1 : Enable hardware break point h for MCS–channel x.
	Setting the bit field **MCS[i]_HBP[k]_CTRL.EN_CH[x]** =0 always clears any upcoming break point events that are already stored within the MCS pipeline and related to the MCS–channel x. **Note:** A re–configuration for the next break point might either keep already stored break point events by setting **MCS[i]_HBP[k]_CTRL.EN_CH[x]** =1 or clear these events by setting **MCS[i]_HBP[k]_CTRL.EN_CH[x]** =0.

SCOPE	
Description	Define scope of h–th hardware break point.
Loop	–
Bit Range	[9 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : (NO_HLT) no halt, only IRQ. 1 : (HLT_CH) halt corresponding MCS–channel. 2 : (HLT_SYS) halt entire system.

TYPE	
Description	Define type of h–th hardware break point.
Loop	–
Bit Range	[14 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

TYPE	
RW-Coding	0 : (INSTR) instruction break point matching on configurable address range. 1 : (DADR_R) data read access break point matching on configurable address range. 2 : (DADR_W) data write access break point matching on configurable address range. 3 : (DADR_RW) data read or write access break point matching on configurable address range. 4 : (DPAT_R) data read access break point matching on configurable data pattern. 5 : (DPAT_W) data write access break point matching on configurable data pattern. 6 : (DPAT_RW) data read or write access break point matching on configurable data pattern.

AND	
Description	Logical AND conjunction of h-th hardware break point
Loop	-
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Logical AND conjunction between h-th and h+1-th hardware break point is disabled. 1 : Logical AND conjunction between h-th and h+1-th hardware break point is enabled.
	Note: The logical AND conjunction of *MATCH_COND* [h:h] and *MATCH_COND* [h+1:h+1] is only applicable if h-th and h+1-th break point are data access break points. If one of these break points is configured as an instruction break point or if the successor break point h+1 is not available, the actual state of this bit field is ignored and the logical AND conjunction is not applied.

NOT	
Description	Logical negation of h-th hardware break point
Loop	-
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Logical negation of h-th hardware break point is disabled. 1 : Logical negation of h-th hardware break point is enabled.

17.11.20 MCS[i]_HBP[h]_PATTERN

Description	MCS[i] hardware break point pattern register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; h = \{n : 0 <= n <= \text{NHBP} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_HBP[h]_PATTERN
Address	$0x20000 * i + 0x20 * h + 0x2F44$
C-Name	GTM.CLS[i].MCS.HBP[h].PATTERN

Interface: MCS[i]

Name	MCS[i]_HBP[h]_PATTERN
Address	$0x20 * h + 0x2F44$
C-Name	

DATA	
Description	Define pattern or address of h-th hardware break point.
Loop	-
Bit Range	$[31 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

If **MCS[i]_HBP[h]_CTRL.TYPE** is either set to DPAT_R, DPAT_W, or DPAT_RW, the bits of the bit field **MCS[i]_HBP[h]_PATTERN.DATA** define the data pattern of the match condition. In all other configurations, the bit field **MCS[i]_HBP[h]_PATTERN.DATA** defines an address range for a match condition. This match condition is true, if the corresponding word address x of a data access or an instruction fetch is in the range **MCS[i]_HBP[h]_PATTERN.DATA** [15:2] <= x <= **MCS[i]_HBP[h]_PATTERN.DATA** [31:18].

17.11.21 MCS[i]_HBP[h]_STATUS

Description	MCS[i] hardware break point status register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; h = \{n : 0 <= n <= \text{NHBP} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_HBP[h]_STATUS
Address	$0x20000 * i + 0x20 * h + 0x2F48$
C-Name	GTM.CLS[i].MCS.HBP[h].STATUS

Interface: MCS[i]

Name	MCS[i]_HBP[h]_STATUS
Address	$0x20 * h + 0x2F48$
C-Name	

HALT_CH[x]	
Description	Indicate that MCS-channel x has fired the h-th hardware break point and either MCS-channel x or the entire GTM was halted due to that break point, depending on the state of bit field MCS[i]_HBP[h]_CTRL.SCOPE.
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : MCS channel x is not halted at break point h. 1 : MCS channel x is halted at break point h.
W-Coding	0 : No action 1 : Resume MCS channel x from break point h.
	Note: This bit will be cleared on a CPU write access with value '1'. A write access with '0' or a read access leaves the bit unchanged. **Note:** The clearing of a bit field **MCS[i]_HBP[h]_STATUS.HALT_CH[x]** will resume the halted channel x from the h-th break point. If the bit field **MCS[i]_HBP[h]_CTRL.SCOPE** is set to value HLT_SYS the entire GTM is resumed and the resume is signalized to microcontroller system.

17.11.22 MCS[i]_HBP[h]_IRQ_NOTIFY

Description	MCS[i] hardware break point interrupt notification register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; h = \{n : 0 <= n <= \text{NHBP} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_HBP[h]_IRQ_NOTIFY
Address	$0x20000 * i + 0x20 * h + 0x2F4C$
C-Name	GTM.CLS[i].MCS.HBP[h].IRQ_NOTIFY

Interface: MCS[i]

Name	MCS[i]_HBP[h]_IRQ_NOTIFY
Address	$0x20 * h + 0x2F4C$
C-Name	

HBP_IRQ	
Description	Interrupt notify flag of the h-th hardware break point
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access with value '1'. A read access leaves the bit unchanged.

17.11.23 MCS[i]_HBP[h]_IRQ_EN

Description	MCS[i] hardware break point interrupt enable register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; h = \{n : 0 <= n <= \text{NHBP} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_HBP[h]_IRQ_EN
Address	$0x20000 * i + 0x20 * h + 0x2F50$
C-Name	GTM.CLS[i].MCS.HBP[h].IRQ_EN

Interface: MCS[i]

Name	MCS[i]_HBP[h]_IRQ_EN

Address	$0x20 * h + 0x2F50$
C-Name	

HBP_IRQ_EN	
Description	Interrupt Enable bit of the h-th hardware break point
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

17.11.24 MCS[i]_HBP[h]_IRQ_FORCINT

Description	MCS[i] force hardware break point interrupt register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; h = \{n : 0 <= n <= \text{NHBP} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_HBP[h]_IRQ_FORCINT
Address	$0x20000 * i + 0x20 * h + 0x2F54$
C-Name	GTM.CLS[i].MCS.HBP[h].IRQ_FORCINT

Interface: MCS[i]

Name	MCS[i]_HBP[h]_IRQ_FORCINT
Address	$0x20 * h + 0x2F54$
C-Name	

TRG_HBP_IRQ	
Description	Trigger the bit MCS[i]_HBP[h]_IRQ_NOTIFY.HBP_IRQ by software
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False

TRG_HBP_IRQ	
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 0, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No status
W–Coding	0 : No action 1 : Force setting of **MCS[i]_HBP[h]_IRQ_NOTIFY.HBP_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

17.11.25 MCS[i]_HBP[h]_IRQ_MODE

Description	MCS[i] break point h interrupt mode configuration register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}; h = \{n : 0 <= n <= \text{NHBP} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_HBP[h]_IRQ_MODE
Address	$0x20000 * i + 0x20 * h + 0x2F58$
C–Name	GTM.CLS[i].MCS.HBP[h].IRQ_MODE

Interface: MCS[i]

Name	MCS[i]_HBP[h]_IRQ_MODE
Address	$0x20 * h + 0x2F58$
C–Name	

IRQ_MODE	
Description	IRQ mode selection for all break point interrupts
Loop	–
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	–

IRQ_MODE	
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

17.11.26 MCS[i]_RESET

Description	MCS[i] reset register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_RESET
Address	$0x20000 * i + 0x2F04$
C-Name	GTM.CLS[i].MCS.RESET

Interface: MCS[i]

Name	MCS[i]_RESET
Address	$0x2F04$
C-Name	

RST[x]	
Description	Software reset of channel x
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status

RST[x]	
W-Coding	0 : No action 1 : Reset channel x

Note:
If a reset for channel x is initiated, the operation of MCS-channel x is stopped immediately and nearly all register bit fields that are uniquely associated to MCS-channel x are set to their initial values. A list of all registers that are affected by a channel reset is mentioned in section *17.7 "MCS Software Reset of Channels"* .

Note:
The **MCS[i]_RESET.RST[x]** (x∈{0, 1,..., (T-1)}) bits are cleared automatically after write access of CPU.

17.11.27 MCS[i]_CAT

Description	MCS[i] cancel ARU transfer instruction
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CAT
Address	$0x20000 * i + 0x2F08$
C-Name	GTM.CLS[i].MCS.CAT

Interface: MCS[i]

Name	MCS[i]_CAT
Address	$0x2F08$
C-Name	

CAT[x]	
Description	Cancel ARU transfer of channel x
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : Bit field **STA.CAT** is cleared. 1 : Bit field **STA.CAT** is set.

CAT[x]	
W-Coding	0 : No action 1 : Cancel any pending blocking ARU read or write transfer

Note:
The **MCS[i]_CAT.CAT[x]** (x∈{0, 1,..., (T-1)}) bit inside the STA register of the corresponding MCS-channel is set and any pending blocking ARU read or write request is canceled. The MCS-channel resumes with the instruction that is following the blocking ARU transfer instruction.

Note:
The **MCS[i]_CAT.CAT[x]** (x∈{0, 1,..., (T-1)}) bit is cleared by the corresponding MCS-channel, when the channel is entering a blocking ARU read or write instruction.

17.11.28 MCS[i]_CWT

Description	MCS[i] cancel waiting instruction
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_CWT
Address	$0x20000 * i + 0x2F0C$
C-Name	GTM.CLS[i].MCS.CWT

Interface: MCS[i]

Name	MCS[i]_CWT
Address	$0x2F0C$
C-Name	

CWT[x]	
Description	Cancel waiting instruction for channel x
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : Bit field **STA.CWT** is cleared. 1 : Bit field **STA.CWT** is set.

CWT[x]	
W-Coding	0 : No action 1 : Cancel any pending waiting instruction.

Note:

The **MCS[i]_CWT.CWT[x]** (x∈{0, 1,..., (T-1)}) bit inside the STA register of the corresponding MCS-channel is set and any pending waiting instruction (WURM, WURMX, WURC, or WUCE) is canceled. The MCS-channel resumes with the instruction that is following the waiting instruction.

Note:

The **MCS[i]_CWT.CWT[x]** (x∈{0, 1,..., (T-1)}) bit is cleared by the corresponding MCS-channel, when the channel reaches a waiting instruction.

17.11.29 MCS[i]_ERR

Description	MCS[i] error register
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_ERR
Address	$0x20000 * i + 0x2F10$
C-Name	GTM.CLS[i].MCS.ERR

Interface: MCS[i]

Name	MCS[i]_ERR
Address	$0x2F10$
C-Name	

ERR[x]	
Description	Error State of MCS-channel x
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async MCS[i]_RESET_CH[x]: sync
R-Coding	0 : Error bit is cleared 1 : Error bit is set

ERR[x]	
W-Coding	0 : No action 1 : Clear error bit

Note:
The CPU can read the **MCS[i]_ERR.ERR[x]** (x∈{0, 1,..., (T-1)}) bits in order to determine the current error state of the corresponding MCS-channel x.

Note:
The error state is also evaluated by the submodule MON, if this module is available.

Note:
Writing a value 1 to this bit field resets the corresponding error state by clearing bit field **STA.ERR** and further the bit field **MCS[i]_CTRL_STAT.ERR_SRC_ID** is set to its reset value.

17.11.30 MCS[i]_MEM

Description	MCS[i] memory region
Loop	$i = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	$\text{NMCS} > 0$
Size	$\text{MCS\_MEM\_SIZE}[i]$
Clock Active Cond	$\text{MCS}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MCS[i]_MEM
Address	$0x20000 * i + 0x10000$
C-Name	GTM.CLS[i].MCS_MEM

DATA	
Description	MCS memory location
Loop	-
Bit Range	$[31 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	MCS[i]_RAM_INIT: sync

Note:
The memory size described above is the maximal memory size. However, the actual size of the memory can be calculated as 2^{MP1-4}, as described in section *17.4 "Memory Organization"* .

17.12 MCS Port Description

17.12.1 MCS Ports

Loop	–
Condition	–
Format	–

DBG_MCS[i]_BP_EN	
Description	Signal for enabling debugging with breakpoints in i-th MCS instance
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

MCS_ERR	
Description	Error signal of MCS
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

MCS_CH_MCA	
Description	Activity signals of MCS-channels
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

17.13 MCS Signal Description

17.13.1 MCS_RAM_INIT

Loop	$j = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	–
Format	–

MCS[j]_RAM_INIT	
Description	MCS[j] memory initialization active
Loop	–
Condition	–
Signal Type	ARRAY[NMCS]
Assignment	MCS[j]_CTRL_STAT.RAM_RST == 1

17.13.2 MCS Reset

Loop	$j = \{n : 0 <= n <= \text{NMCS} - 1\}; x = \{n : 0 <= n <= 7\}$
Condition	–
Format	–

MCS[j]_RESET_CH[x]	
Description	MCS[j] channel x reset active signal
Loop	–
Condition	–
Signal Type	ARRAY[NMCS][8]
Assignment	MCS[j]_RESET.RST[x] == 1

17.13.3 MCS Signals

Loop	–
Condition	–
Format	–

BP_ACTION	
Description	Signalize break point action.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

MATCH_COND	
Description	Signalize break point match condition to preceding hardware break point.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_IRQ	
Description	Shared IRQ signal of current MCS instance
Loop	–
Condition	–
Signal Type	INT
Assignment	–

18 ADC Interface (ADCIF)

18.1 Overview

The clusters of the GTM can provide a dedicated interface for the connection of up to 32 external Analog Digital Converter (ADC) channels, which can be mapped arbitrarily to physical instances of single or multiple channel ADCs.
An ADC interface is directly mapped into the address map of the AEI bus master interface of the current cluster's MCS, meaning that the available AEI bus master instructions (BRD and BRDI as described in section *17.9 "Instruction Set"*) are used to control the connected ADCs.
Since the control of the connected ADCs is silicon vendor specific, the GTM specification does not provide a complete specification for controlling connected ADCs. However, to ensure software compatibility at least for the basic features of an ADC, the functionality described in the following is common to all silicon vendors.

Indices and their range as used inside this chapter are:

i:={0, NADC−1} index of ARU instance
y:={0, 1,..., 31} index ADC channel

18.2 Basic ADC Functions

The address map of the AEI bus master interface reserves two unique address items for each ADC channel. The registers can be referred by **ADC[i]_CH[y]_DATA** and **ADC[i]_CH[y]_STA** for the channel [y] in the range from 0 to 31.
The MCS can read from address **ADC[i]_CH[y]_DATA** in order to get the conversion result of the ADC that is connected to ADC channel y. The conversion result is represented as a signed 24−bit value and it is stored in the register A (A ∈ GREG) as referred by the corresponding MCS instruction BRD or BRDI. Additionally, each read access to **ADC[i]_CH[y]_DATA** triggers the ADC, that is connected to channel y. Any read access to **ADC[i]_CH[y]_DATA** also provides 8 status bits that are stored in register **MCS[i]_CH[x]_MHB** . The bit **MCS[i]_CH[x]_MHB** [7:7] has always the mnemonic **ADC[i]_CH[y]_DATA.ADC_ACK** and the bit **MCS[i]_CH[x]_MHB** [6:6] has always the mnemonic **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** . If bit **ADC[i]_CH[y]_DATA.ADC_ACK** is set the result of the data conversion (register A) and the corresponding status bits (bits **MCS[i]_CH[x]_MHB** [6:0]) are valid. If **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** is set the current conversion result is new and has never been read by a previous bus read access. The meaning of the bits **MCS[i]_CH[x]_MHB** [5:0] are vendor specific. Otherwise, if **ADC[i]_CH[y]_DATA.ADC_ACK** and **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** are cleared, the read data is invalid and the **MCS[i]_CH[x]_MHB** [4:0] indicates the channel identifier (with **MCS[i]_CH[x]_MHB** [4:0] != y) that is currently processed by the ADC. A write access to **ADC[i]_CH[y]_DATA** has no functionality and is always ignored.
The MCS can read from address **ADC[i]_CH[y]_STA** to get additional 31−bit wide vendor specific status information of ADC channel y. The lower 24 bits of the status information is stored in register A (A ∈ GREG) as referred by the corresponding MCS instruction BRD or BRDI. The upper 7 bits of the status information is stored in register **MCS[i]_CH[x]_MHB** [6:0]. The bit **MCS[i]_CH[x]_MHB** [7:7] has always the mnemonic **ADC[i]_CH[y]_STA.ADC_ACK** . If bit **ADC[i]_CH[y]_STA.ADC_ACK** is set, the result of status information in register A and bits **MCS[i]_CH[x]_MHB** [6:0] is valid. Otherwise, if **ADC[i]_CH[y]_DATA.ADC_ACK** and **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** are cleared the status information is invalid. Otherwise, if **ADC[i]_CH[y]_DATA.ADC_ACK** and **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** are cleared, the read data is invalid and the **MCS[i]_CH[x]_MHB** [4:0] indicates the channel identifier (with **MCS[i]_CH[x]_MHB** [4:0] != y) that is currently processed by the ADC. A write access to **ADC[i]_CH[y]_STA** has no functionality and is always ignored.

If **ADC[i]_CH[y]_DATA.ADC_ACK** is cleared and **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** is set, an error is signaled, the AEI status bits will be unequal to "00". Details about the status value can be investigated by reading register **CCM[i]_AEIM_STA** . The following status information is defined for the AEI status values:

- 00 − no error occurred,
- 01 − optional information register not implemented (only register **ADC[i]_CH[y]_STA**),
- 10 − illegal ADC access (e.g. ADC not enabled),
- 11 − unsupported address (ADC channel y not available).

18.3 ADCIF Configuration Registers Description

The ADCIF configuration registers are accessible by the MCS with the AEI bus master interface.

18.3.1 ADC[i]_CH[y]_DATA

Description	ADC[i] channel [y] ADC data register
Loop	$i = \{n : 0 <= n <= \mathrm{NADC} - 1\}; y = \{n : 0 <= n <= 31\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	ADC[i]_CH[y]_DATA
Address	$0x8 * y + 0x5400$
C-Name	

DATA	
Description	ADC converted data of channel [y]. In integer signed format.
Loop	-
Bit Range	$[23 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

STATUS	
Description	Status of ADC channel [y] read request.
Loop	-
Bit Range	$[29 : 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	Depending on the value of **ADC[i]_CH[y]_DATA.ADC_ACK** , **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** this bit field has special meanings: **ADC[i]_CH[y]_DATA.ADC_ACK** = **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** = 0: Channel actually operated by the ADC. **ADC[i]_CH[y]_DATA.ADC_ACK** = 1 or **ADC[i]_CH[y]_DATA.ADC_NEW_DATA** = 1: Vendor information about the ADC conversion of channel [y].

ADC_NEW_DATA	
Description	Read data of ADC channel [y].
Loop	-
Bit Range	$[30 : 30]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-

ADC_NEW_DATA	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : ADC data was already read by a previous read command 1 : ADC data is new; was never read before

ADC_ACK	
Description	Read data of ADC channel [y] valid.
Loop	–
Bit Range	[31 : 31]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Requested ADC data is invalid 1 : Requested ADC data is valid

18.3.2 ADC[i]_CH[y]_STA

Description	ADC[i] channel [y] ADC status information register
Loop	$i = \{n : 0 <= n <= \text{NADC} - 1\}; y = \{n : 0 <= n <= 31\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{MCS}[i]\_\text{CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	ADC[i]_CH[y]_STA
Address	$0x8 * y + 0x5404$
C-Name	

INFO	
Description	Conversion status information of channel [y].
Loop	–
Bit Range	[30 : 0]
Access Type	R
Volatile	True
Multithread	False

INFO	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Depending on the value of **ADC[i]_CH[y]_STA.ADC_ACK** , this bit field has special meanings: **ADC[i]_CH[y]_STA.ADC_ACK** =0 and **ADC[i]_CH[y]_STA.INFO** [30:30]=0: **ADC[i]_CH[y]_STA.INFO** [29:24] will indicate the channel actually operated by the ADC. **ADC[i]_CH[y]_STA.ADC_ACK** =1: Vendor information about the ADC conversion of channel [y].

ADC_ACK	
Description	Data valid of read of ADC channel [y].
Loop	–
Bit Range	[31 : 31]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Requested ADC status information is invalid 1 : Requested ADC status information is valid

19 Memory Configuration (MCFG)

19.1 Overview

The Memory Configuration submodule (MCFG) is an infrastructure module that organizes physical memory blocks and maps them to the RAM ports 0 and 1 of available Multi–Channel Sequencer (MCS) modules.

The module MCFG can be configured by the configuration parameters NMCFG, MCS_MAW, and MCS_ERM.
If NMCFG is 1, the module MCFG is implemented in GTM and if NMCFG is 0, MCFG is not implemented. The vector elements MCS_MAW[i] and MCS_ERM[i] represent the i–th element of the corresponding vector variable MCS_MAW and MCS_MAW, whereas the index i corresponds to the instance number of an MCS instance. MCS_MAW[i] defines the memory address width of the physically connected large RAM block that is connected to RAM port 0 of the i–th MCS. The parameter MCS_ERM[i] defines, if the RAM block that is associated to RAM port 1 of the i–th MCS has full size (MCS_ERM[i] is 1) or half size (MCS_ERM[i] is 0).
If NMCFG is 0, the vector variables MCS_MAW and MCS_ERM, define individual but fixed RAM sizes for different MCS instance. Section *17.4 "Memory Organization"* describes how the RAM sizes are defined.
If NMCFG is 1, all elements vector of MCS_MAW must be equal and further all elements of vector MCS_ERM must be equal. Therefore, these parameters are no longer instance specific and the following description can use the simplified scalar parameters MAW = MCS_MAW[0] and ERM = MCS_ERM[0].

Depending on the value of parameter ERM, the MCFG module assumes externally connected physical RAM modules with different sizes. If ERM is 0, MCFG assumes that each MCS instance provides a large physical memory block with 2^{MAW} memory locations each 32 bit wide which leads to a RAM module with 2^{MAW+2} (byte wise) memory addresses. Further, each MCS instance provides a small physical memory block with 2^{MAW-1} memory locations each 32 bit wide leading to a RAM module with 2^{MAW+1} (byte wise) memory addresses. If ERM is 1, MCFG assumes that each MCS instance provides two large physical memory blocks each with 2^{MAW} memory locations each 32–bit leading to a RAM module with 2^{MAW+2} (byte wise) memory addresses.

In order to support different memory sizes for different MCS instances, the MCFG module provides three layout configurations for reorganization of memory pages mapped to the RAM ports of neighboring MCS modules. Figure *140 "Memory Layout Configurations (ERM = 0)"* shows all layout configurations for the case that ERM = 0 and Figure *142 "Memory Layout Configurations (ERM = 1)"* shows the layout configurations for the case that ERM = 1. Each box in these pictures represents a physical memory block.

The layout configuration DEFAULT always assigns a memory block of size 2^{MAW} x32 bits to MCS RAM port 0. Depending on the ERM, RAM port 1 of each MCS is assigned to a memory block of size 2^{MAW-1} x32 bits (ERM = 0) or 2^{MAW} x32 bits (ERM = 1).

The layout configuration SWAP is swapping the memory block assigned to RAM port 1 of the current MCS instance with the memory block assigned to RAM port 0 of the successive MCS instance. If ERM = 0, this means that the memory of the current MCS instance is increased by 2^{MAW-1} x32 bits but the memory of the successor is decreased by 2^{MAW-1} x32 bits compared to the DEFAULT configuration. If ERM = 1, the SWAP configuration has no effect on the memory sizes of the individual MCS instances.

The layout configuration BORROW is borrowing the memory block assigned to RAM port 0 of the successive MCS instance for the current instance. This means, the memory of the current MCS module is increased by 2^{MAW} x32 bits but the memory of the successor is decreased by 2^{MAW} x32 bits compared to the DEFAULT configuration.

Considering the order of the mentioned MCS modules, it should be noted that the successor of the last MCS instance is the first MCS instance MCS0.

The actual sizes of the memory pages mapped to the MCS RAM ports 0 and 1, depends on the layout configuration of the current instance MCS[i] and the layout configuration of the preceding memory instance MCS[i–1]. The sizes of these memory pages can be obtained by the layout parameters MP0 and MP1, as described in the specification of the MCS.

Table *141 "Memory Layout Parameters (ERM = 0)"* and Figure *143 "Memory Layout Parameters (ERM = 1)"* summarize the layout parameters MP0 and MP1 of MCS instance MCS[i] for the case that ERM = 0 and ERM = 1. Note that the predecessor of instance MCS0 is last available MCS instance.

The addressing of memory port 0 ranges from 0 to MP0–4 and the addressing of memory page 1 ranges from MP0 to MP1–4.

This document assumes that the GTM implementation embeds 8 MCS instances. However, the actual number of implemented MCS instances can be obtained from [1].

Figure 140 Memory Layout Configurations (ERM = 0)

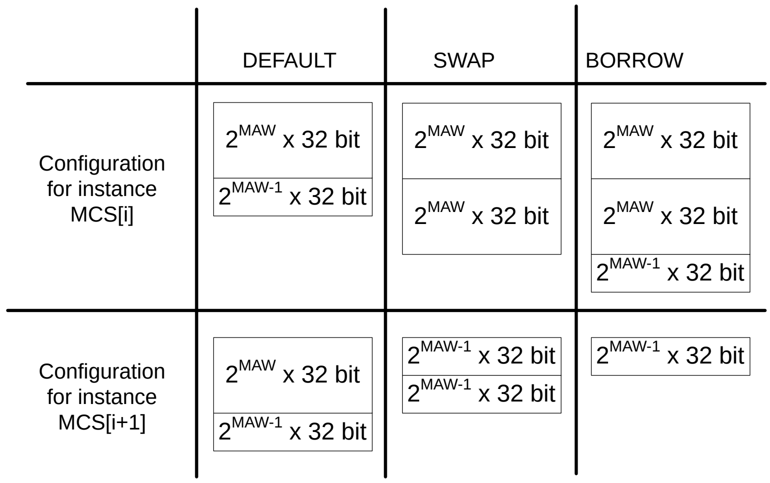


Figure 141 Memory Layout Parameters (ERM = 0)

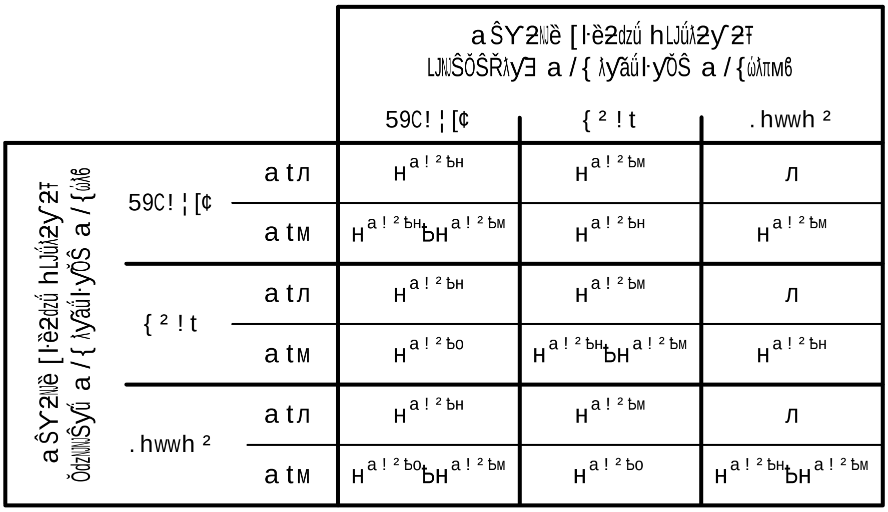


Figure 142 Memory Layout Configurations (ERM = 1)

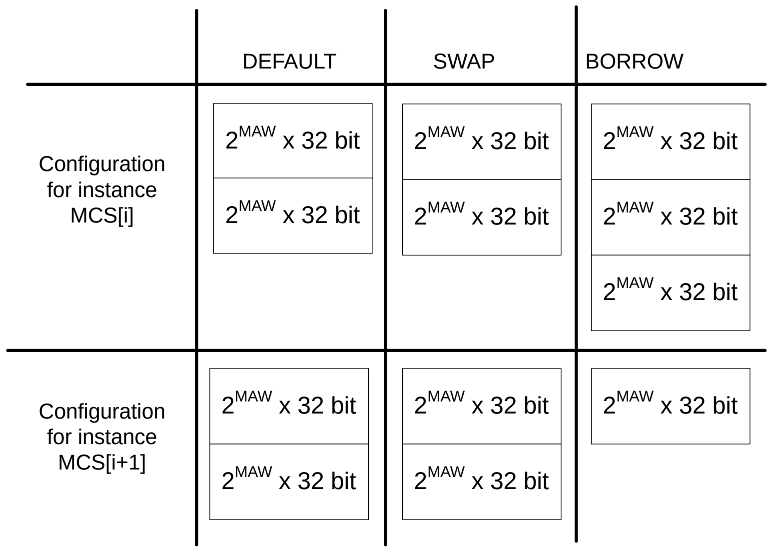
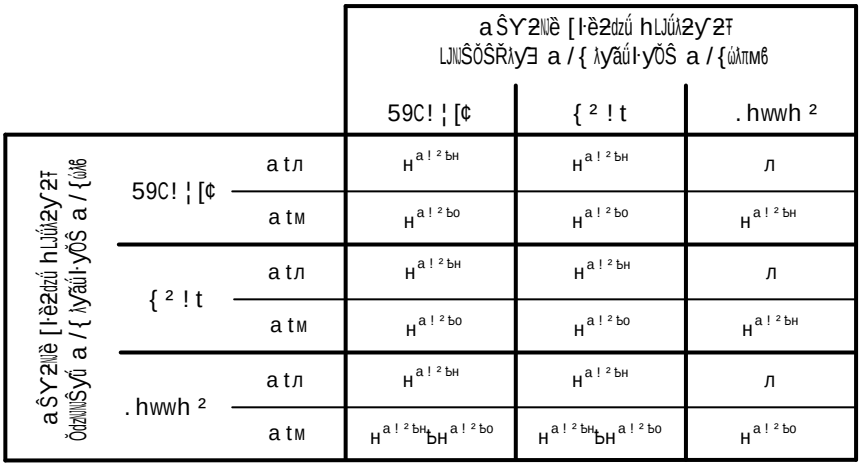


Figure 143 Memory Layout Parameters (ERM = 1)



19.2 MCFG Configuration Register Description

19.2.1 MCFG_CTRL

Description	MCFG Memory layout configuration.
Loop	
Condition	$\mathrm{NMCS} > 1 \&\& \mathrm{NMCFG} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCFG\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MCFG_CTRL
Address	$0x600$
C-Name	GTM.CLS[0].MCFG.CTRL

Interface: MCS[i]

Name	MCFG_CTRL
Address	$0x600$
C-Name	

MEM[k]	
Description	Configure memory pages for MCS-instance MCS[k].
Loop	$k = \{n : 0 <= n <= 9\}$
Bit Range	$[2 * k + 1 : 2 * k]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NMCS} > k \&\& \mathrm{NMCFG} > 0$
Initial value	0
Protect Enable Cond	RF_PROT == 1

MEM[k]	
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : DEFAULT configuration 0b01 : SWAP configuration 0b10 : BORROW configuration

Note:
It should be noted that the actual GTM-IP implementation may embed less MCS instances than mentioned in this register (see [1]). In this case, this register only implements the register bits for available MCS instances.

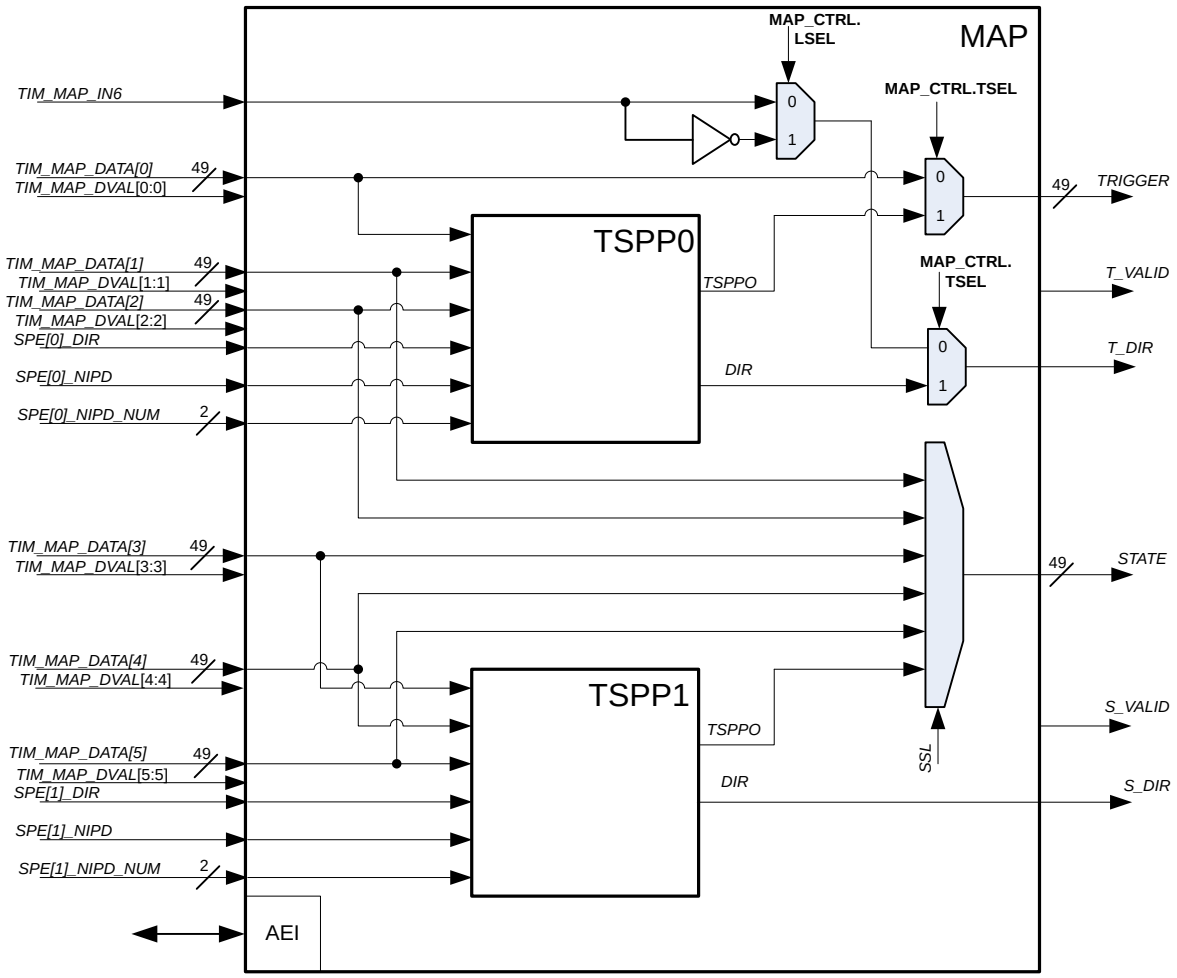
Note:
This register is only writable, if the bit **GTM_CTRL.RF_PROT** is cleared.

20 TIM0 Input Mapping Module (MAP)

20.1 Overview

The MAP submodule generates the two input signals *TRIGGER* and *STATE* for the submodule DPLL by evaluating the output signals of the channel 0 up to channel 5 of submodule TIM0. By using the TIM as input submodule, the filtering of the input signals can be done inside the TIM channels themselves. The MAP submodule architecture is depicted in figure *144 "MAP Sub-module Architecture"* .

Figure 144 MAP Sub-module Architecture



Generally, the MAP submodule can route the channel signals coming from TIM0 in three ways. First, it is possible to route the whole 49 bits of data coming from channel 0 of module TIM0 (*TIM_MAP_DATA[0]*) to the *TRIGGER* signal which is then provided to the DPLL together with the *T_VALID* signal.

Second, the MAP module can route one of the five signals coming from the module TIM0 (i.e. the signals coming from channel 1 up to channel 5) to the output signal *STATE* which is then provided to the module DPLL together with the *S_VALID* signal.

Third, the *TRIGGER* , *T_VALID* , *STATE* and *S_VALID* signals can be generated out of the TIM Signal Preprocessing (TSPP) subunits. This is done in combination with the Sensor Pattern Evaluation (SPE) submodule described in chapter *23 "Sensor Pattern Evaluation (SPE)"* .

The signal *TRIGGER* is generated in subunit TSPP0 out of the TIM0 signals coming from channel 0 up to channel 2.

The signal *STATE* is generated in subunit TSPP1 out of the TIM signals coming from channel 3 up to channel 5.

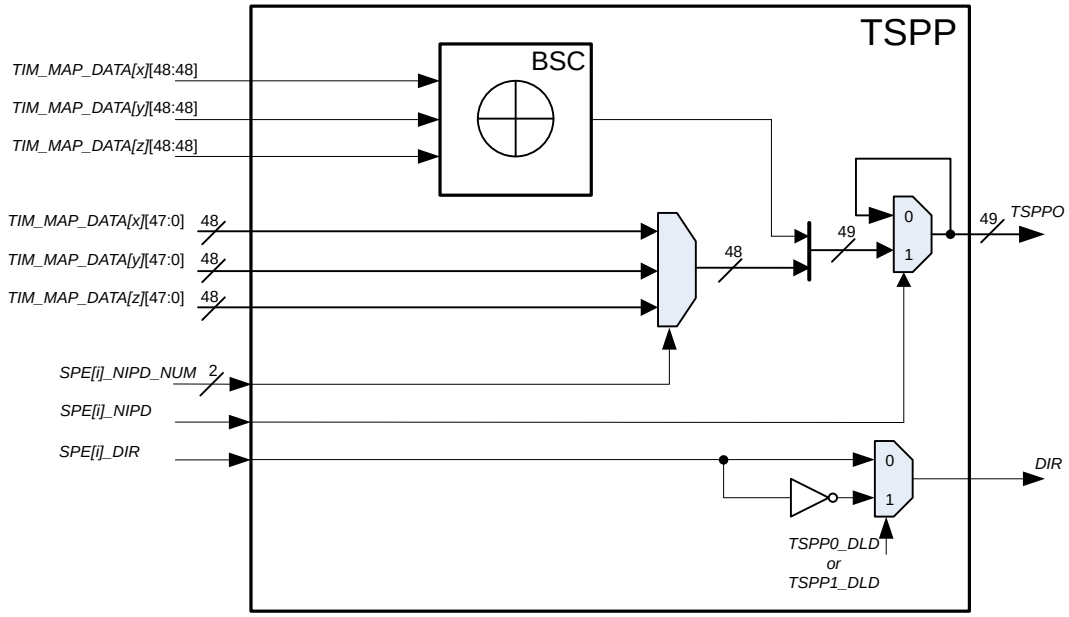
This is only done, when the TSPP0 and TSPP1 subunits are enabled and when the *SPE[i]_NIPD* signal is raised by the SPE submodule. The *SPE[i]_NIPD_NUM* signal encodes, which of the 3 *TIM_MAP_DATA[y]* input signals have been changed. The *SPE[i]_DIR* signal is routed through the TSPP0 and TSPP1 subunits and implements the *T_DIR* or *S_DIR* signal.

A third method to provide a direction signal to DPLL is to use TIM0 channel 6 input (*TIM_MAP_IN6*) and to route it instead of the *DIR* signal coming from TSPP0 to the MAP output *T_DIR* (set **MAP_CTRL.TSEL** =0)

20.2 TIM Signal Preprocessing (TSPP)

The TSPP combines the three 49 bit input streams coming from the TIM0 submodule and generates one combined 49 bit output stream *TSPPO* . The input stream combination is done in the unit Bit Stream Combination (BSC). The architecture of the TSPP is shown in figure *145 "TIM Signal Preprocessing (TSPP) Sub-Unit Architecture"* .

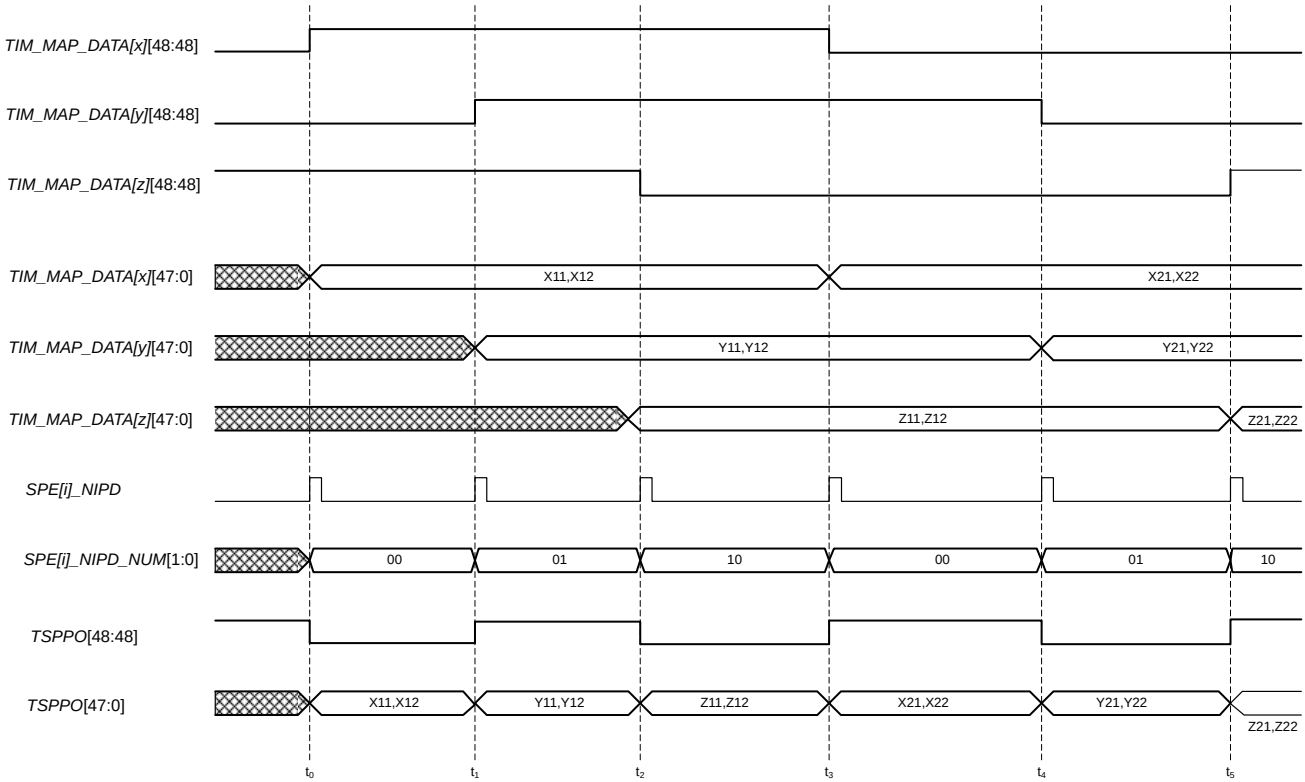
Figure 145 TIM Signal Preprocessing (TSPP) Sub-Unit Architecture



20.2.1 Bit Stream Combination

The BSC subunit is used to xor-combine the three most significant bits *TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DATA[y]* [48:48] and *TIM_MAP_DATA[z]* [48:48] of the TIM0 inputs. The xor-combined signal is merged with the remaining 48 bits of one of the three input signals *TIM_MAP_DATA[x]* [47:0], *TIM_MAP_DATA[y]* [47:0] or *TIM_MAP_DATA[z]* [47:0], the *TSPPO* signal. The selection is done with the *SPE[i]_NIPD_NUM* input signal coming from the SPE submodule. The action, when the 49 bits are transferred to *TSPPO* and *T_VALID* or *S_VALID* signal is raised, is determined by the *SPE[i]_NIPD* signal coming from the SPE submodule. The *TSPPO* output signal generation is shown in the example in Figure *146 "TSPP Signal Generation for Signal TSPPO"* .

Figure 146 TSPP Signal Generation for Signal TSPPO



The *SPE[i]_NIPD_NUM* input signal determines, which data is routed to the *TSPPO* signal. At the first edge of *TIM_MAP_DATA[x]* [48:48], the new data X11 and X12 are routed to *TSPPO* [47:0]. The values X11 and X12 are the two 24-bit values coming from the TIM input channel

TIM_MAP_DATA[x] . The next edge is at time t $_1$ on signal *TIM_MAP_DATA[y]* [48:48]. Therefore, at time t $_1$ the *TSPPO* [48:48] signal level changes and the *TSPPO* [47:0] is set to Y11 and Y12 and so forth.

20.3 MAP Configuration Register Description

20.3.1 MAP_CTRL

Description	MAP Control register
Loop	
Condition	$\mathrm{NMAP} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MAP\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MAP_CTRL
Address	$0x640$
C-Name	GTM.CLS[0].MAP.CTRL

Interface: MCS[i]

Name	MAP_CTRL
Address	$0x640$
C-Name	

TSEL	
Description	TRIGGER signal output select.
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : *TIM_MAP_DATA[0]* selected as *TRIGGER* output signal. *TIM_MAP_IN6* (TIM0 channel 6 input) is used as direction signal *T_DIR* . 1 : *TSPPO* of TSPP0 selected as *TRIGGER* output signal.

SSL	
Description	STATE signal output select
Loop	-
Bit Range	$[3:1]$
Access Type	RW
Volatile	False
Multithread	False

SSL	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b000 : *TIM_MAP_DATA[1]* [48:48] selected as *STATE* output signal. 0b001 : *TIM_MAP_DATA[2]* [48:48] selected as *STATE* output signal. 0b010 : *TIM_MAP_DATA[3]* [48:48] selected as *STATE* output signal. 0b011 : *TIM_MAP_DATA[4]* [48:48] selected as *STATE* output signal. 0b100 : *TIM_MAP_DATA[5]* [48:48] selected as *STATE* output signal. 0b101 : *TSPPO* of TSPP1 selected as *STATE* output signal. 0b110 : Same as '000' 0b111 : Same as '000'

LSEL	
Description	TIM_MAP_DATA[6] input level selection
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : *TIM_MAP_DATA[6]* [48:48] input level '0' encodes TRIGGER in forward direction. 1 : *TIM_MAP_DATA[6]* [48:48] input level '1' encodes TRIGGER in forward direction.

TSPP0_EN	
Description	Enable of TSPP0 subunit
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : TSPP0 disabled – TSPPO and DIR are tied to 0. 1 : TSPP0 enabled.

TSPP0_DLD	
Description	DIR level definition bit
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : *SPE[i]_DIR* signal is routed through as is. 1 : *SPE[i]_DIR* signal is inverted.

TSPP0_I0V	
Description	Disable of TSPP0 TIM_MAP_DATA[0][48:48] input line
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Input line enabled. 1 : Input line disabled; input for TSPP0 is set to 0.

TSPP0_I1V	
Description	Disable of TSPP0 TIM_MAP_DATA[1][48:48] input line
Loop	–
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

TSPP0_I1V	
RW-Coding	0 : Input line enabled. 1 : Input line disabled; input for TSPP0 is set to 0.

TSPP0_I2V	
Description	Disable of TSPP0 TIM_MAP_DATA[2][48:48] input line
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Input line enabled. 1 : Input line disabled; input for TSPP0 is set to 0.

TSPP1_EN	
Description	Enable of TSPP1 subunit
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : TSPP1 disabled – TSPPO and DIR are tied to 0. 1 : TSPP1 enabled.

TSPP1_DLD	
Description	DIR level definition bit
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

TSPP1_DLD	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : *SPE[i]_DIR* signal is routed through as is. 1 : *SPE[i]_DIR* signal is inverted.

TSPP1_I0V	
Description	Disable of TSPP1 TIM_MAP_DATA[3][48:48] input line
Loop	–
Bit Range	[28 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Input line enabled. 1 : Input line disabled; input for TSPP1 is set to 0.

TSPP1_I1V	
Description	Disable of TSPP1 TIM_MAP_DATA[4][48:48] input line
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Input line enabled. 1 : Input line disabled; input for TSPP1 is set to 0.

TSPP1_I2V	
Description	Disable of TSPP1 TIM_MAP_DATA[5][48:48] input line
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False

TSPP1_I2V	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Input line enabled. 1 : Input line disabled; input for TSPP1 is set to 0.

20.4 MAP Port Description

20.4.1 MAP Ports

Loop	–
Condition	–
Format	–

DIR	
Description	Output port of submodules TSPP0/TSPP1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TSPPO	
Description	Output port of submodules TSPP0/TSPP1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

S_DIR	
Description	Direction control signal for DPLL State
Loop	–
Condition	–
Logical Name	–

S_DIR	
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

T_DIR	
Description	Direction control signal for DPLL TRIGGER
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

21 Digital PLL Module (DPLL)

21.1 Overview

The digital PLL (DPLL) submodule is used for frequency multiplication. The purpose of this module is to get a higher precision of position or value information also in the case of applications with rapidly changing input frequencies. There are two input signals *TRIGGER* and *STATE* for which periodic events are processed. The time period between two active events is called an increment. Each increment is divided into a given number of sub increments by pulses called *SUB_INC* . The resolution of the generated pulses is restricted by the period of the *CCM[0]_CLK_RES* [0:0] clock or the *TS_CLK* respectively (see description of the modules TBU, CMU). The input signals *TRIGGER* and *STATE* can have the meaning of position information of linear or angular motions, mass flow values, temperature, pressure or level of liquids. By means of the DPLL, the load of the CPU can be reduced essentially by relieving it from repeated or periodic standard tasks.

The DPLL has to perform the following tasks:

- prediction of the duration of the current increment in chapter *21.7 "Prediction of the current increment duration"*
- generation of *SUB_INC1* , *SUB_INC2* pulses for up to 2 position counters in normal or emergency mode (see chapter *21.9.3 "Sub pulse generation for DPLL_CTRL_1.SMC=0"*)
- synchronization of the actual position (under CPU control, see chapter *21.9.6.1 "Synchronization description"*)
- possibility of seamless switch to emergency mode and back under CPU control, see configuration register **DPLL_CTRL_0** at chapter *21.11 " DPLL Registers Description"*
- prediction of position and time related events in chapter *21.8 "Calculations for Actions"*

Indices as used inside this chapter are :

- NOAC= Number Of Action Channels
- n:={0, 1,..., (NOAC−1)} index of Action (sub functions)
- m = loop index of registers (belonging to Action channels)
- p = index of data out of memory according to profile of flywheel (*TRIGGER* and *STATE* channel)
- pm, pn, pt, pq, ps, pr = (indices used for profile pointers in the context of the signal processing equations. They are introduced formally in the chapters of use)

All estimation values that follow time for the DPLL are given for a system clock frequency of 100 MHz.

21.2 Requirements and demarcation

The two input signals *TRIGGER* and *STATE* can be sensor signals from the same device or from two independent devices. When they come from the same device the *TRIGGER* input is typically a more frequent signal and *STATE* is a less frequent signal. In such a case the *STATE* signal can support an emergency mode, when no *TRIGGER* signal is available. There are also applications supported when *STATE* and *TRIGGER* are independent signals from different devices. Both input signals are combined with a validation signal *T_VALID* or *S_VALID* respectively, which shows the appearance of new data and must result in a data fetch and a start of the corresponding state machine to perform the calculations (see explanation below).

When *STATE* is a redundant signal of the same device, only the *TRIGGER* input is used to generate the *SUB_INC1* pulses in normal mode. A configuration called emergency mode is possible, for which the *SUB_INC1* pulses are generated using the *STATE* input signal.
The decision to switch into the emergency mode and back is made outside the DPLL. The CPU must switch the configuration bit **DPLL_CTRL_0.RMO** (reference mode) (see chapter *21.11 " DPLL Registers Description"*). Because a switch in emergency mode can appear suddenly, the information about the duration of the *STATE* input up to FULL_SCALE of the last increment should always be stored as a precaution.

The filtering as well as the combination or choice of the input signals is made in the TIM submodule (see chapter *13 "Timer Input Module (TIM)"*) by use of a configurable filter algorithm for each slope and signal as well as in the MAP module (see chapter *20 "TIM0 Input Mapping Module (MAP)"*) the right *TRIGGER* or *STATE* signal is selected by a multiplexer or in the SPE module (see chapter *23 "Sensor Pattern Evaluation (SPE)"*) different signals are combined to a *TRIGGER* or *STATE* signal by using an antivalence operation.

The filter delay value of the signal is transmitted from the TIM module in the FT part of the corresponding signal, because the delay conditions of the signals can change during application.

The filter delays depend also on the filter algorithms used. Only the effective filter delay can be considered in the DPLL.

In order to provide the timing conditions to the DPLL, the input *TRIGGER* signals should have a time stamp (and optionally also a filter value and a signal level value, as stated above) with an appropriate resolution. The resolution of the time stamps can either be the same resolution as the input time base *CCM[0]_TBU_TS0* (see chapter *148 "DPLL Block Diagram"*) or 8 times higher, selected by configuration bits in the **DPLL_CTRL_1** register (see chapter **DPLL_CTRL_1**). The time base *CCM[0]_TBU_TS0* is used to predict events in the future, called actions.

At the *SUB_INC1* and *SUB_INC2* outputs a predefined number of pulses between each active slope of the *TRIGGER* / *STATE* signal is generated, when the corresponding pulse generator is enabled by the enable bits **DPLL_CTRL_1.SGE1** =1 and **DPLL_CTRL_1.SGE2** =1 (see chapter **DPLL_CTRL_1**).

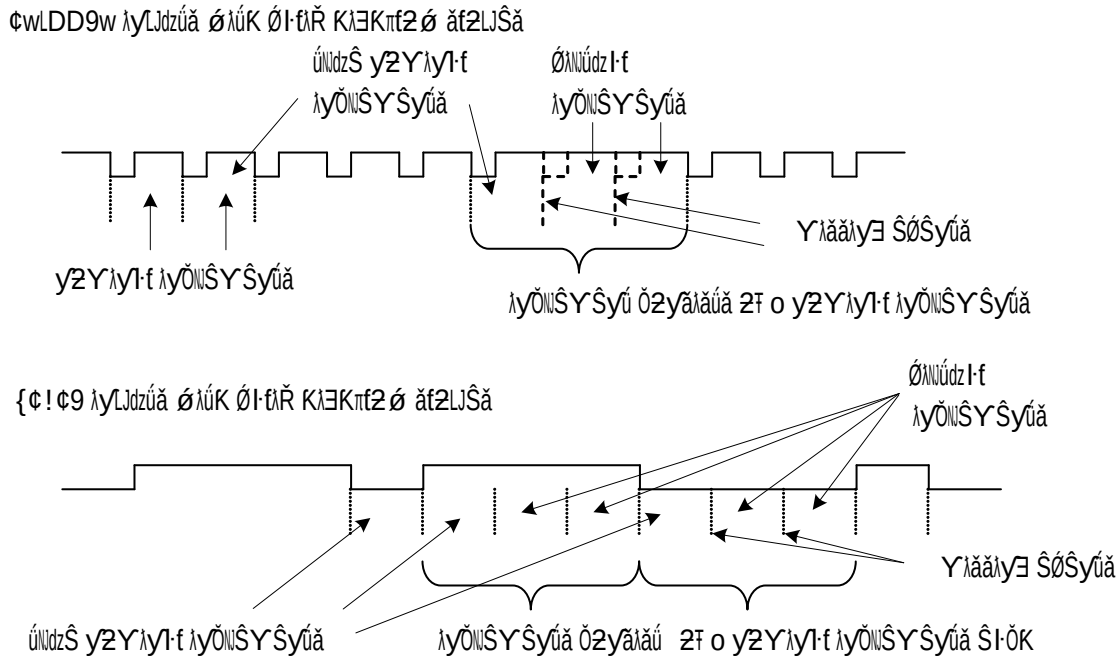
Depending on the configuration, different strategies can be used to correct a wrong pulse number.

The FULL_SCALE range is divided into a fix number of nominal increments. Nominal increments have the same size. The number of nominal increments in HALF_SCALE is specified in the **DPLL_CTRL_0** register (see chapter *21.11 " DPLL Registers Description"*).
For synchronization purposes some *TRIGGER* / *STATE* input signals can be suppressed depending on the current position. Therefore, an increment as duration between two active input events can either be a nominal increment or it can consist of more than one nominal increment.
While a true nominal increment starts with an active event a virtual increment (of always nominal size) is an increment which starts with a missing event. Each increment which represents a gap (e.g. for synchronization purposes), consists of exactly one true nominal increment and at least one virtual increment, each of them having the same nominal duration (see figure below).

21.3 Input signal courses

Typical input signal courses are shown in the figure below.

Figure 147 Trigger and State Input Signal



21.3.1 Implementation-related constraints on input signals

The time elapsed between an active *STATE* / *TRIGGER* event and the associated inactive *STATE* / *TRIGGER* event must always be greater than 10µs (1000 *CLS[0]_CLK* cycles).

21.4 State Extension

By default, the DPLL supports up to 64 *STATE* events in FULL_SCALE. By setting **DPLL_CTRL_11.STATE_EXT** to 1, the DPLL state extension is enabled (see *22 "MCS to DPLL Interface of DPLL Module (MCS2DPLL)"*) and the support is extended to up to 128 *STATE* events in FULL_SCALE. When the state extension is used, the data structures (**DPLL_RDT_S[p]** , **DPLL_TSF_S[p]** , **DPLL_ADT_S[p]** , **DPLL_DT_S[p]**) in RAM1c are not used. The data structures are instead emulated/managed by MCS[0].

The variables defined in Table *49 "State Exension Variables"* are used throughout the chapter:

Table 49 State Exension Variables

Variable	Refers to (if DPLL_CTRL_11.STATE_EXT == 0)	Refers to (if DPLL_CTRL_11.STATE_EXT == 1)
DPLL_NUSC_SYN_S	**DPLL_NUSC.SYN_S**	**DPLL_NUSC_EXT1.SYN_S**
DPLL_NUSC_SYN_S_OLD	**DPLL_NUSC.SYN_S_OLD**	**DPLL_NUSC_EXT1.SYN_S_OLD**
DPLL_NUSC_NUSE	**DPLL_NUSC.NUSE**	**DPLL_NUSC_EXT2.NUSE**
DPLL_NUSC_FSS	**DPLL_NUSC.FSS**	**DPLL_NUSC_EXT2.FSS**
DPLL_NUSC_VSN	**DPLL_NUSC.VSN**	**DPLL_NUSC_EXT2.VSN**
DPLL_APS_APS	**DPLL_APS.APS**	**DPLL_APS_EXT.APS**
DPLL_APS_1C2	**DPLL_APS.APS_1C2**	**DPLL_APS_EXT.APS_1C2**

Variable	Refers to (if DPLL_CTRL_11.STATE_EXT == 0)	Refers to (if DPLL_CTRL_11.STATE_EXT == 1)
DPLL_APS_1C3_1C3	**DPLL_APS_1C3.APS_1C3**	**DPLL_APS_1C3_EXT.APS_1C3**
DPLL_APS_SYNC_APS_1C2_EXT	**DPLL_APS_SYNC.APS_1C2_EXT**	**DPLL_APS_SYNC_EXT.APS_1C2_EXT**
DPLL_APS_SYNC_APS_1C2_STATUS	**DPLL_APS_SYNC.APS_1C2_STATUS**	**DPLL_APS_SYNC_EXT.APS_1C2_STATUS**
DPLL_APS_SYNC_APS_1C2_OLD	**DPLL_APS_SYNC.APS_1C2_OLD**	**DPLL_APS_SYNC_EXT.APS_1C2_OLD**
DPLL_SNU	**DPLL_CTRL_0.SNU**	**DPLL_CTRL_EXT.SNU**
DPLL_SYN_NS	**DPLL_CTRL_1.SYN_NS**	**DPLL_CTRL_EXT.SYN_NS**

21.5 Block and interface description

The block description of the DPLL is shown in the following figure.

Figure 148 DPLL Block Diagram

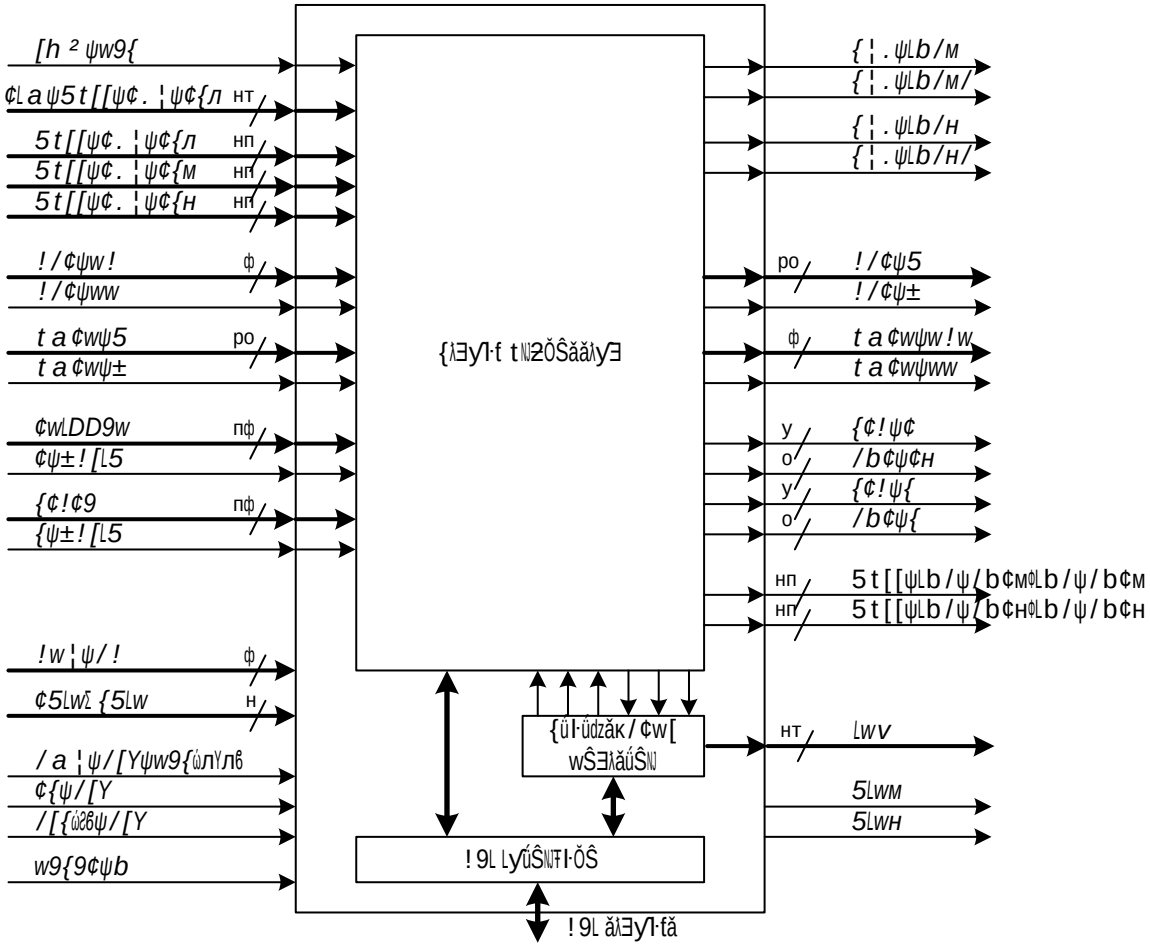


Table *21.16 "DPLL Port Description"* summarizes the interface signals of the DPLL shown by the block diagram above.

21.6 DPLL Architecture

21.6.1 Purpose of the module

The DPLL generates a predefined number of incremental signal pulses within the period between two events of an input *TRIGGER* or *STATE* signal, when the corresponding pulse generator is enabled. The resolution of the pulses is restricted by the frequency of the time stamp clock (*TS_CLK*). Changes in the period length of the predicted time period of the current increment will result in a change of the pulse frequency in order to get the same number of pulses. This adoption can be performed by DPLL hardware, software or with support of DPLL hardware in different modes.
The basic part of a DPLL is to make a prediction of the current period between two *TRIGGER* and/or *STATE* signal edges. Disturbances and systematic failures must be considered as well as changes of increment duration caused by acceleration and deceleration of the supervised process. Therefore, a good estimation is to be done using some measuring values from the past. When the process to be predicted takes a

steady and differentiable course not only the current increment but also some more increments for the future can be predicted. By utilizing such calculations, actions for the future can be predicted.

21.6.2 Explanation of the prediction methodology

As already shown in chapter *21.1 "Overview"* the DPLL has to perform different tasks. The basic function for all these tasks is the prediction of the current increment which is based on a relation between increments in the past. Because the relation between two succeeding intervals at a fixed position remains also valid in the case of acceleration or deceleration the prediction of the duration of the current time interval is done by a similarity transformation. Having a good estimation of the current time interval, all the other tasks can be done easily by calculations explained in chapter *21.7 "Prediction of the current increment duration"* .

21.6.3 Clock topology

All registers are read using the system clock *CLS[0]_CLK* . The clock used for the generation of the normal pulses is *TS_CLK* . The clock resolution of *TS_CLK* may not exceed 50% of *CLS[0]_CLK* . The same applies for the *CCM[0]_RES_CLK* [0:0] which is used for rapid pulse generation. Normally, the generated *SUB_INC1* / *SUB_INC2* pulses have the highest frequency but not higher than *TS_CLK* (or half of *CLS[0]_CLK*). When normal pulses and rapid pulses are generated, the frequency of pulse generation can be doubled. All operations are performed using the system clock (*CLS[0]_CLK*). For more details refer to the Figure ({ref::DPLL_9427}).

21.6.4 Clock generation

The clock is generated outside the DPLL.

21.6.5 Typical frequencies

For the system clock a reasonable clock frequency should be applied to give the DPLL module sufficient computational power to calculate all needed values (prediction of next increment, Actions) in time. The typical system clock frequency is in the range from 40 MHz up to 150 MHz.

21.6.6 Time stamps and systematic corrections

The time stamps for the input signals *TRIGGER* and *STATE* have 24 bits each. These bits represent the value of the 24 bit free running counter running with a clock frequency selected by the configuration of the TBU. Using a typical frequency of 20 MHz the time stamp represents a relative value of time with a resolution of 50 ns.

The input signals have to be filtered. The filter is not part of the DPLL. The time stamps can have a delay caused by the filter algorithm used. There are delayed and undelayed filter algorithms available and the delay value can depend on a time or a position value.

Systematic deviations of *TRIGGER* inputs can be corrected by a profile, which also considers systematic missing *TRIGGER* s. The increments containing missing *TRIGGER* S are divided into the corresponding number of nominal increments whereas the duration of a nominal increment is the greatest divider of all increments duration.
For each increment, the number of enclosed nominal increments is stored in a profile as **DPLL_ADT_T[p].NT** value for *TRIGGER* . When the increment is a nominal increment the **DPLL_ADT_T[p].NT** value is 1.
For the *TRIGGER* input the value **DPLL_ADT_T[p].NT** is stored in the **DPLL_ADT_T[p]** field in RAM region 2c.

In case **DPLL_CTRL_0.AMT** = 1, the **DPLL_ADT_T[p]** values in the RAM region 2c must also contain the adapting information for the *TRIGGER* signal, which for each increment considers a systematic physical deviation **DPLL_ADT_T[p].PD** from the perfect nominal increment value with a resolution according to the chosen value of **DPLL_CTRL_0.MLT** +1, which describes the number of *SUB_INC1* pulses for a nominal increment.

The value **DPLL_ADT_T[p].PD** for the *TRIGGER* describes the number of missing or surplus pulses with a signed int 13 bit value, to be added to **DPLL_CTRL_0.MLT** +1 directly. Thus, the correction value is also applicable for the synchronization gaps in the case of missing *TRIGGER* inputs. In this case the number of provided *SUB_INC1* pulses for a nominal increment (**DPLL_CTRL_0.MLT** +1) + **DPLL_ADT_T[p].PD** is multiplied by **DPLL_ADT_T[p].NT** .
The **DPLL_ADT_T[p].NT** value of the current increment is stored in the variable **DPLL_NUTC.SYN_T** (see chapter **DPLL_NUTC**).

In case **DPLL_CTRL_0.RMO** = 1 for **DPLL_CTRL_1.SMC** =0 (emergency mode) the time stamp of *STATE* is used to generate the output signal *SUB_INC1* .

More inaccuracy should be accepted in emergency mode because usually there are only fewer events available for FULL_SCALE according to the value DPLL_SNU.

For the *STATE* signal the systematic deviations of the increments can be corrected in the same way as for *TRIGGER* by profile and adaptation information as described below.

Systematic deviations of *STATE* inputs can be corrected by a profile, which also considers systematic missing *STATE* events. The increments containing missing *STATE* are divided into the corresponding number of nominal increments whereas the duration of a nominal increment is the greatest divider of all increments duration.
For each increment this number of enclosed nominal increments is stored in a profile as **DPLL_ADT_S[p].NS** value for *STATE* . When the increment is a nominal increment the **DPLL_ADT_S[p].NS** value is 1.
For the *STATE* input the value **DPLL_ADT_S[p].NS** is stored in the **DPLL_ADT_S[p]** field in RAM region 1c3.

In case **DPLL_CTRL_0.AMS** = 1 the **DPLL_ADT_S[p]** values in the RAM region 1c3 must contain the adapting information for the *STATE* signal, which for each increment considers a systematic physical deviation **DPLL_ADT_S[p].PD_S** from the perfect nominal increment value with a resolution according to the chosen value of **DPLL_MLS1.MLS1** , which describes the number of *SUB_INC1* pulses for a nominal increment (see below).

The number of pulses *SUB_INC1* for a nominal *STATE* increment in emergency mode (for **DPLL_CTRL_1.SMC** =0) is given by the value of **DPLL_MLS1.MLS1** = (**DPLL_CTRL_0.MLT** + 1)* (**DPLL_CTRL_0.TNU** + 1) /(DPLL_SNU + 1) in order to get the same number of pulses in FULL_SCALE for normal and emergency mode. This value has to be configured by the CPU.

The value **DPLL_ADT_S[p].PD_S** for the *STATE* describes the number of missing or surplus pulses with a signed int 16 bit value, to be added to **DPLL_MLS1.MLS1** directly. Thus the correction value is also applicable for the synchronization gaps in the case of missing *STATE* inputs. In this case the number of provided *SUB_INC1* pulses for a nominal increment **DPLL_MLS1.MLS1** + **DPLL_ADT_S[p].PD_S** is multiplied by **DPLL_ADT_S[p].NS** .
The current **DPLL_ADT_S[p].NS** value is stored in the variable DPLL_NUSC_SYN_S (see chapter **DPLL_NUSC**).

21.6.7 DPLL Architecture overview

As shown in the block diagram of chapter *148 "DPLL Block Diagram"* , the DPLL can process different input signals. The signal *TRIGGER* is the normal input signal which gives the detailed information of the supervised process. It can be for instance the information of water or other liquid level representing the volume of the liquid, where each increasing millimeter results in a *TRIGGER* signal generation. In order to get a predefined filling level, without overflow also the inertia of the system must be taken into account. Hence, some delay for closing the inlet valve and also the remaining water amount in the pipe must be considered in order to start the closing action earlier as the filling level will be reached.

A second input signal *STATE* sends an additional (redundant) information for instance at some centimeters and because of intervals with different distances it also gives information about the system state with the direction of the water flow (in or out), while the *TRIGGER* signal must not contain information concerning the flow direction. In some applications the inactive slope of *TRIGGER* can be utilized to transmit direction information. In the case of faults in the *TRIGGER* signal, the *STATE* signal should be processed to still reach the desired value, possibly with some loss of accuracy.

The measuring scale can have some systematic failures, because not all millimeter or centimeter distances measured mean the same value. This could be due to changes in the thickness of the measuring cylinder or the inaccurate position of the marks. These systematic failures are well known by the system and for improvement of the prediction, the signals *ADT_T* and *ADT_S* are stored in the internal RAM for the correction of the systematic failures of *TRIGGER* and *STATE* respectively.

The input signals *TRIGGER* and *STATE* are represented as a time stamp signal each, which is stored in the 24 bit TS–part of the corresponding signal.

Information concerning the delay of this signal by filtering of disturbances is stored in the 24 bit FT–part of the signal.

In order to calculate upcoming actions, time stamps can be related to the current time which is provided by the input signal *CCM[0]_TBU_TS0* .

After reaching the desired water level, the water is filled in a bottle by draining. After that the water filling is repeated. The water level at draining is observed by the same sensor signals (the same number of *TRIGGER* pulses), but the duration of the draining could be different from the filling time. Both times together form the FULL_SCALE region, while one of them is a HALF_SCALE region, which can differ in time but not in the number of pulses, especially for *TRIGGER* .

For synchronization purposes some *TRIGGER* marks can be omitted in order to set the system to a proper synchronization value (maybe before the upper filling value is reached).

In emergency situations when the *TRIGGER* signals are missed, the *STATE* signal is used instead.

The *PMTR[n]* input signals announce the request for a position minus time calculation for up to 32 events.

All NOAC events can be activated using *AEN[n]* =1 (Action enable) bits. Each of these enable bits are asked by the routing engine for a read access. The corresponding read request is generated by the *AEN[n]* bit while **DPLL_STATUS.CAIP1** and **DPLL_STATUS.CAIP2** is zero. **DPLL_STATUS.CAIP1** and **DPLL_STATUS.CAIP2** are two bits of the **DPLL_STATUS** register for NOAC/2 actions each with the meaning "calculation of actions in progress", controlled by the state machine. See chapter *21.9.6 "Scheduling of the Calculation"* for scheduling the operations and chapter *51 "State description of the State Machine Table"* for a detailed description of the state machine.

When such a request is serviced by the ARU (in this case **DPLL_STATUS.CAIP1** =0 and **DPLL_STATUS.CAIP2** =0) the values for position and time are written in the corresponding RAM 1a region (0x0200... 0x025C for the position value and 0x0260... 0x02BC for the delay value), the control bits for the corresponding action are set accordingly. When a new PMTR value arrives, an old value is overwritten without notice and the shadow bit of **DPLL_ACT_STA.ACT_N[n]** is cleared while the **DPLL_ACT_STA.ACT_N[n]** (new action) bit in the **DPLL_ACT_STA** register is set. The **DPLL_ACT_STA.ACT_N[n]** is cleared, when the currently calculated action value is in the past. Overwriting of old information is possible without data inconsistency because the read request to ARU is suppressed during action calculations by the **DPLL_STATUS.CAIP1** and **DPLL_STATUS.CAIP2** bits. In this way, the last possible PMTR value is always used consistently.

21.6.8 DPLL Architecture description

The DPLL block diagram in chapter *148 "DPLL Block Diagram"* will now be explained in detail in combination with some sample configurations of the control registers. There are different configuration bits available which can adopt the DPLL to the use case (see chapter *21.11 " DPLL Registers Description"*).
For example in HALF_SCALE, let the *TRIGGER* number **DPLL_CTRL_0.TNU** be 0x3B (which is for **DPLL_CTRL_0.TNU** +1 = 60 decimal, which means 120 events in FULL_SCALE) and the number of *SUB_INC1* pulses between two *TRIGGER* events **DPLL_CTRL_0.MLT** be 0x257 (this

means 600 pulses per *TRIGGER* event). After this, the FULL_SCALE region can be divided into 72000 parts. Each of this part is associated with its own *SUB_INC1* pulse. For a run through FULL_SCALE all 72000 pulses should appear but maybe with a different pulse frequency between two *TRIGGER* events. For this example, after every 600 pulses at the *SUB_INC1* output, the next *TRIGGER* event is to be expected with the corresponding new time stamp.

SUB_INC1 pulses missing due to acceleration have to be taken into account within the next increment. Not one pulse has to be missed or added because of calculation inaccuracy in average for a sufficient number of FULL_SCALE periods. This means that not one pulse is sent in addition and all missing pulses are to be caught up on afterwards.

For the systematic arrangement of *TRIGGER* inputs the profile (as already mentioned in chapter *21.6.6 "Time stamps and systematic corrections"* is stored in the RAM region 2c (see register **DPLL_ADT_T[p]**). In this field the relative position of gaps can be stored in the **DPLL_ADT_T[p].NT** value and also physical deviations in the **DPLL_ADT_T[p].PD** value.

For the consideration of systematic missing *TRIGGER* s the actual **DPLL_ADT_T[p].NT** value of the profile is stored in the **DPLL_NUTC.SYN_T** bits (see chapter **DPLL_NUTC**).

In normal mode the physical deviation values **DPLL_ADT_T[p].PD** could be used to balance the local systematic inaccuracy of the *TRIGGER* signal. The value of **DPLL_ADT_T[p].PD** is the pulse difference in the corresponding nominal increment and means the number of sub pulses to be added to the nominal number of pulses. **DPLL_ADT_T[p].PD** is a signed integer value using 13 bits, up to +/−4096 pulses can be added for each increment.

The **DPLL_ADT_T[p].NT** has the value 1, when a nominal increment is assumed. An integer number greater than 1 shows the number of nominal increments to be considered for a gap. For the actual increment after synchronization the corresponding **DPLL_ADT_T[p].NT** value is stored in **DPLL_NUTC.SYN_T** .

Using the *STATE* input there are similar configuration bits available (see chapter *21.11 " DPLL Registers Description"*) .
Let for example, in HALF_SCALE the *STATE* number DPLL_SNU be 0xB (which is for DPLL_SNU+1 =12 decimal and while **DPLL_CTRL_1.SYSF** =0 that means 24 events in FULL_SCALE). In order to get the same number of *SUB_INC1* pulses for FULL_SCALE as above for *TRIGGER* s, the value (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1)=600 is divided by 2*(DPLL_SNU+1)=24 and multiplied with 2*(**DPLL_CTRL_0.TNU** +1)=120. The result 3000 must be stored in **DPLL_MLS1.MLS1** by the CPU (see chapter **DPLL_MLS1**).

For the systematic arrangement of *STATE* inputs the profile (as already mentioned in chapter *21.6.6 "Time stamps and systematic corrections"* is stored in the RAM region 1c3 (see chapter **DPLL_ADT_S[p]**). In this field the relative position of gaps can be stored in the **DPLL_ADT_S[p].NS** value and also physical deviations in the **DPLL_ADT_S[p].PD_S** value.

For the consideration of systematic missing *STATE* s the actual **DPLL_ADT_S[p].NS** value of the profile is stored in the DPLL_NUSC_SYN_S bits (see chapter **DPLL_NUSC**).

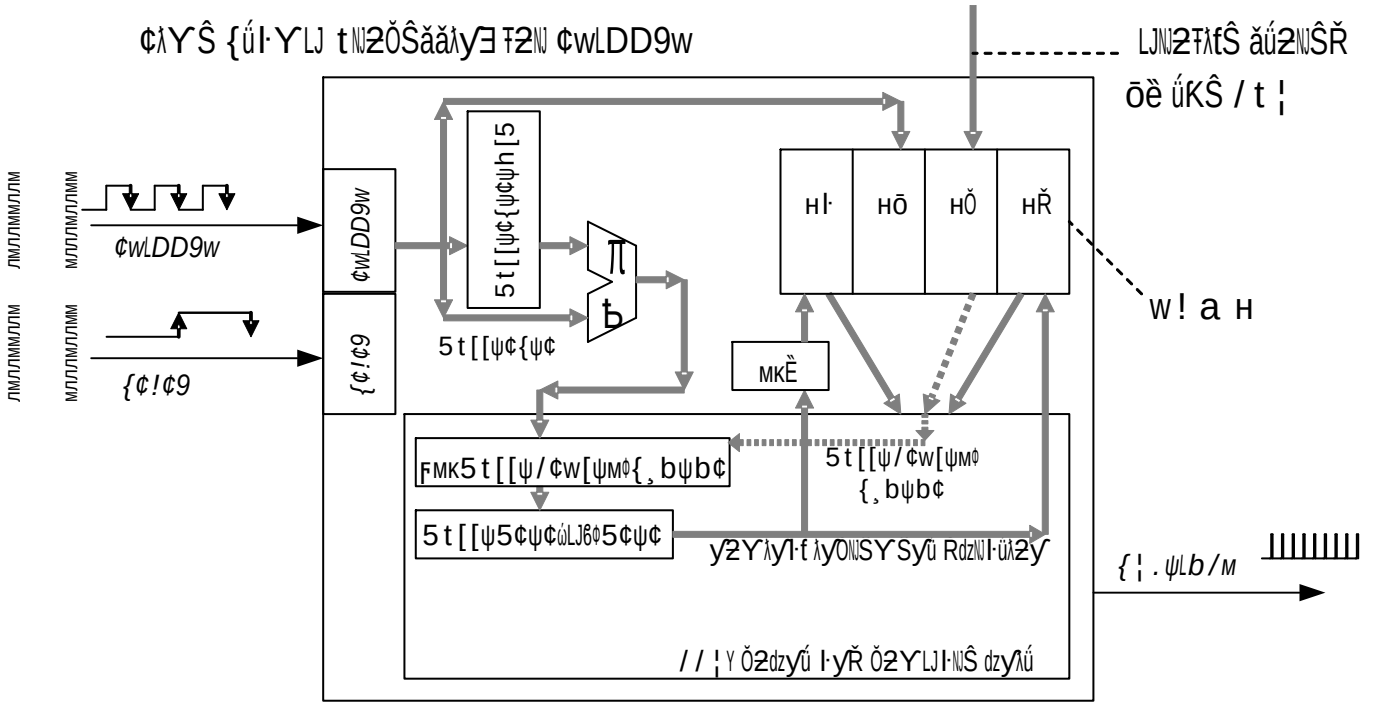
In emergency mode the physical deviation values **DPLL_ADT_S[p].PD_S** could be used to balance the local systematic inaccuracy of the *STATE* signal. The value of **DPLL_ADT_S[p].PD_S** (see chapter **DPLL_ADT_S[p]**) is the pulse difference in the corresponding increment and means the number of sub pulses to be added to the nominal number of pulses per increment. **DPLL_ADT_S[p].PD_S** is a signed integer value using 16 bits: up to +/−32768 pulses can be added for each increment.

In emergency mode the physical deviation values **DPLL_ADT_S[p].PD_S** could be used to balance the local systematic inaccuracy of the *STATE* signal. The value of **DPLL_ADT_S[p].PD_S** (see chapter **DPLL_ADT_S[p]**) is the pulse difference in the corresponding nominal increment and means the number of sub pulses to be added to the nominal number of pulses. **DPLL_ADT_S[p].PD_S** is a signed integer value using 16 bits: up to +/−32768 pulses can be added for each increment.

The **DPLL_ADT_S[p].NS** has the value 1, when a nominal increment is assumed. An integer number greater than 1 shows the number of nominal increments to be considered for a gap. For the actual increment after synchronization the corresponding **DPLL_ADT_S[p].NS** value is stored in DPLL_NUSC_SYN_S.

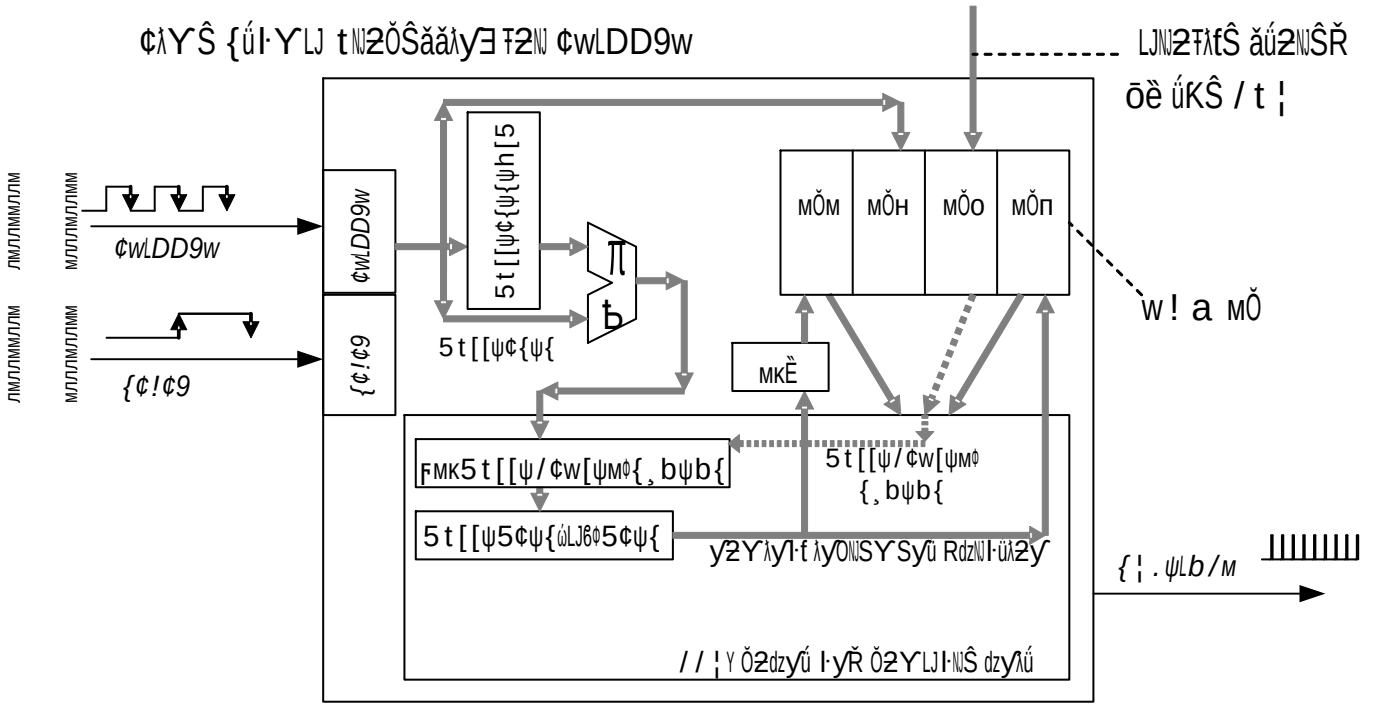
21.6.9 Block diagrams of time stamp processing.

Figure 149 Time Stamp Processing Trigger



As shown in the block diagram above the time stamp difference of two succeeding input events is calculated. For the prediction of the current increment duration such values from the past are used. For this purpose the measured and calculated values of the last FULL_SCALE period are stored in the RAM. For the *TRIGGER* input there are 4 different RAM parts in the RAM region 2:
2a stores the reciprocals of each nominal increment duration **DPLL_RDT_T[p].RDT_T**
2b stores the time stamps of each active input event **DPLL_TSF_T[p].TSF_T**
2c is used for the profile **DPLL_ADT_T[p]** and
2d for the nominal increment duration **DPLL_DT_T[p].DT_T** .
Because the prediction is based on the relations of increments in the past this relation can be calculated easily by the multiplication of increment duration values with the reciprocal value of another increment. In order not to be forced to distinguish between gaps and "normal" increments duration also for gaps only the nominal duration and the corresponding reciprocal values are stored in the RAM field. This is possible by consideration of the **DPLL_ADT_T[p].NT** value in the profile. The measured increment duration is divided by **DPLL_ADT_T[p].NT** .

Figure 150 Time Stamp Processing State



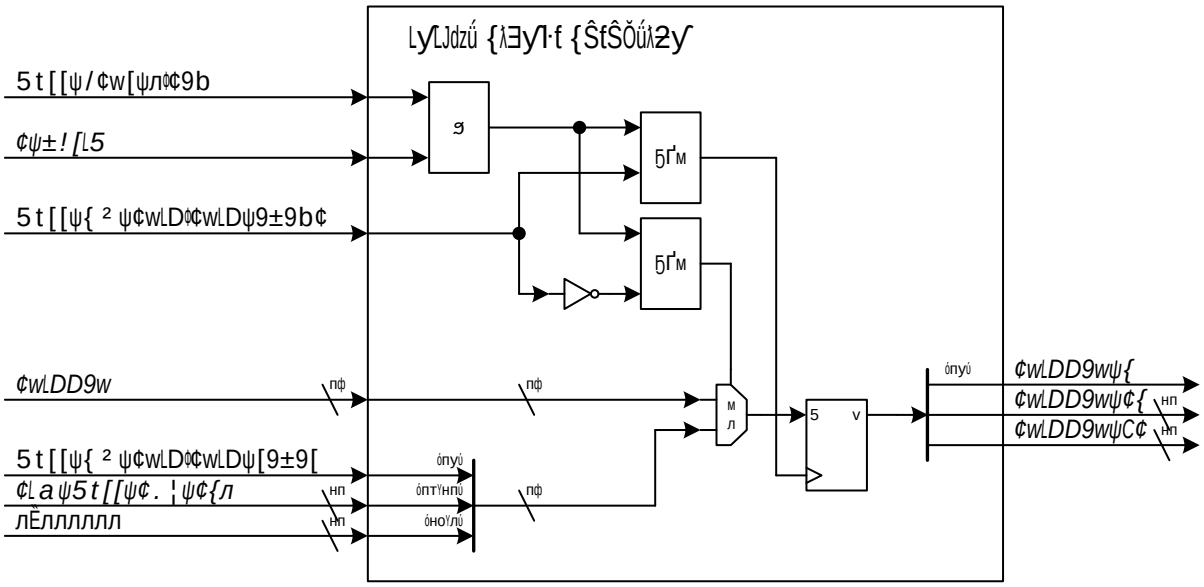
For the *STATE* input there are also 4 different RAM parts in the RAM region 1c:

- 1c1 stores the reciprocals of each nominal increment duration **DPLL_RDT_S[p].RDT_S**
- 1c2 stores the time stamps of each active input event **DPLL_TSF_S[p].TSF_S**
- 1c3 is used for the profile **DPLL_ADT_S[p]** and
- 1c4 for the nominal increment duration **DPLL_DT_S[p].DT_S** .

The calculations are performed similar to the *TRIGGER* input. The **DPLL_ADT_S[p].NS** value in the profile shows the appearance of a gap.

21.6.10 Input signals and input signal selection

Figure 151 Input signal event and input signal selection



As described in *21.6.7 "DPLL Architecture overview"* the input signals *TRIGGER* and *STATE* are represented as time stamp signals (TS part of signal). Additionally delays introduced by input signal filtering can be added (FT part of signal).

In order to resolve dedicated system conditions, e.g. loss of input signal or unwanted change in direction due to disturbed input signal, it is possible to replace the input signals delivered via TIM0 by the activation of an input signal with writing to the register bit field **DPLL_SW_TRIG.TRIG_EVENT** =1 (*TRIGGER*) or **DPLL_SW_TRIG.STATE_EVENT** =1 (*STATE*). When the input signal is triggered under the above condition a part of the current value of the input signals *TIM_DPLL_TBU_TS0* is used as time stamp (TS part) to process the input signal event. In this case the FT part of the input signals is set to zero. For further details see chapter **DPLL_SW_TRIG** . The picture *151 "Input signal event and input signal selection"* is showing the functional diagram for the *TRIGGER* input signal channel. The *STATE* input signal channel can be handled in a similar manner. The necessary register bit fields for *STATE* are described in chapter **DPLL_SW_TRIG** as well.

21.6.11 Register and RAM address overview

The address map of the DPLL is divided into register and memory regions as defined in the table of chapter *50 "Register and RAM address map"* . The addresses from 0x0000 to 0x00FC are reserved for registers, from 0x0100 to 0x01FC are reserved for action registers to serve the ARU at immediately read request.
The RAM is divided into 3 independent accessible parts 1a, 1b+c and 2.
The part 1a from 0x0200 to 0x037C is used for PMTR values received from ARU and intermediate calculation values; there is no write access from the CPU possible, while the DPLL is enabled.
The RAM 1b part from 0x0400 to 0x05FC is reserved for RAM variables and the RAM part 1c from 0x0600 to 0x09FC is used for the *STATE* signal values.
The RAM region 2 from 0x4000 to 0x7FFC is reserved for the *TRIGGER* signal values. RAM region 1a has a size of 288 bytes, RAM 1b+c uses 1.125 Kbytes while RAM region 2 size ranges from 1.5 to 12 Kbytes, depending on the number of *TRIGGER* events in FULL_SCALE.
The **DPLL_AOSV_2** register is used to determine the beginning of each part. The memory size is set by device configuration variable DPLL_RR2_MEM_SIZE (see *86 "GTM Device Configuration Variables, Fixed or Derived Values"*).

The table in chapter *50 "Register and RAM address map"* gives the DPLL address map overview.

Registers are used to control the DPLL and to show its status. Also, parameters are stored in registers when useful. The table below shows the addresses for status and control registers as well as values stored in additional registers. The meaning of the registers are explained in the register overview *21.14 " DPLL RAM Region 2 value description"* while the bit positions of the status and control registers are described in detail in chapter *21.11 " DPLL Registers Description"* .

Time stamps for *TRIGGER* and *STATE* can either have the same resolution as the *CCM[0]_TBU_TS0* input or 8 times higher. This is configured in the **DPLL_CTRL_1** register (see chapter **DPLL_CTRL_1**). While the *CCM[0]_TBU_TS0* is used for action predictions the higher resolution of *TRIGGER* and *STATE* inputs can be used for a more accurate pulse generation.

The time stamp fields of *TRIGGER* and *STATE* are stored in the corresponding RAM regions in such a way, that entries for the virtual increments are also provided for a gap. This is due to the necessity to calculate time differences between a given number of (real and virtual) input events independent of a gap. Therefore, the gap is extended in the RAM fields 2b and 1c2.
For all other RAM regions in RAM 2 and RAM 1c the gap is considered as one increment.

For the access to the RAM fields, there must be address pointers. When the device starts, all address pointers have a value zero and the first measured and calculated values are stored in the beginning of the corresponding RAM field.
Because the position of the device is usually unknown at the beginning no profile information can be used. The profile regions must have their own address pointers each, which are set by the CPU as soon as the position is known. By setting the appropriate value to the address pointer **DPLL_APT_2C.APT_2C** of the *TRIGGER* profile or DPLL_APS_1C3_1C3 of the *STATE* profile respectively the synchronization bits **DPLL_STATUS.SYT** or **DPLL_STATUS.SYS** are set. In the following the gap information can be used.

Because the time stamp fields are extended at the gaps there must be additional address pointers for these regions: **DPLL_APT.APT_2B** for *TRIGGER* time stamps and DPLL_APS_1C2 for *STATE* time stamps. These address pointers must be incremented by **DPLL_ADT_T[p].NT** or **DPLL_ADT_S[p].NS** respectively when a gap appears.

Table 50 Register and RAM address map

Addr. range Start	Addr. range End	Value number	Byte #	Content	Indication	Region	RAM size
0x0000	0x0FC	64	256	Register	used/reserved	0	no RAM
0x100	0x1FC	64	192	Action registers	direct read from ARU	0	no RAM
0x0200	0x03FC	128	384	PMTR values RAM 1a	CPU R/Pw access, when DPLL disabled; ARU has highest priority	1a with own ports	RAM part 1a: 384 bytes
0x0400	0x05FC	128	384	Variables RAM 1b	R and monitored W access by the CPU	1b	RAM part 1b+c: 1.125 Kbytes
0x0600	0x09FC	256	768	*STATE* data	R and monitored W access by the CPU	1c	
0x0600	0x06FC	64	192	**DPLL_RDT_S[p].RDT_S**	*STATE* reciprocal values	1c1	
0x0700	0x07FC	64	192	**DPLL_TSF_S[p].TSF_S**	*STATE_TS* values	1c2	
0x0800	0x08FC	64	192	**DPLL_ADT_S[p]**	adapted values of *STATE*	1c3	
0x0900	0x09FC	64	192	**DPLL_DT_S[p].DT_S**	nom. *STATE* inc.	1c4	
0x4000	0x47FC ... 0x7FFC	512 ... 4096	1536 ... 12288	*TRIGGER* data	R and monitored W access of CPU	2	RAM part 2: 1.-5... 12 Kbytes
0x4000	0x41FC...4FFC	128... 1024	384...3072	**DPLL_RDT_T[p].RDT_T**	*TRIGGER* reciprocal values	2a	
0x4200...5000	0x43FC...5FFC	128... 1024	384...3072	**DPLL_TSF_T[p].TSF_T**	*TRIGGER_TS* values	2b	
0x4400...6000	0x45FC...6FFC	128... 1024	384...3072	**DPLL_ADT_T[p]**	adapted values of *TRIGGER*	2c	
0x4600...7000	0x47FC...7FFC	128... 1024	384...3072	**DPLL_DT_T[p].DT_T**	nom. *TRIGGER* increments	2d	

21.6.11.1 RAM Region 1

RAM region 1 has a size of 1.5 Kbytes and is used to store variables and parameters as well as the measured and calculated values for increments of *STATE* . The RAM 1 region is divided into two independent accessible RAM parts (a and b+c) with own ports. The address information is shown in the table above and the detailed description is performed in the following chapters. The RAM 1a is used to store the PMTR values received from ARU and in addition some intermediate calculation results of actions. RAM region 1b is used for variables needed for the prediction of increments, while RAM 1c is used to store time stamps, profile and duration of all the *STATE* inputs of the last FULL_SCALE region. All variables and values of RAM 1b+c part use a data width of up to 24 bits.

The RAM is to be initialized by the DPLL after HW–reset. All RAM cells must have a zero value after performing the initialization procedure. This is performed when setting the **DPLL_RAM_INI.INIT_RAM** bit. The DPLL is only available after finishing this procedure. The initialization progress is shown in the status bits of the same register.

- RAM Region 1a: used for storage of PMTR values received from ARU; read and write access by the CPU is only possible, when the DPLL is disabled. The CPU Address range: 0x0200 – 0x03FC

- RAM Region 1b: used for intermediate calculations and auxiliary values, data width of 3 bytes used for 24 bit values. A write access to this region results in an interrupt to the CPU, when enabled. Address range: 0x0400 – 0x05FC

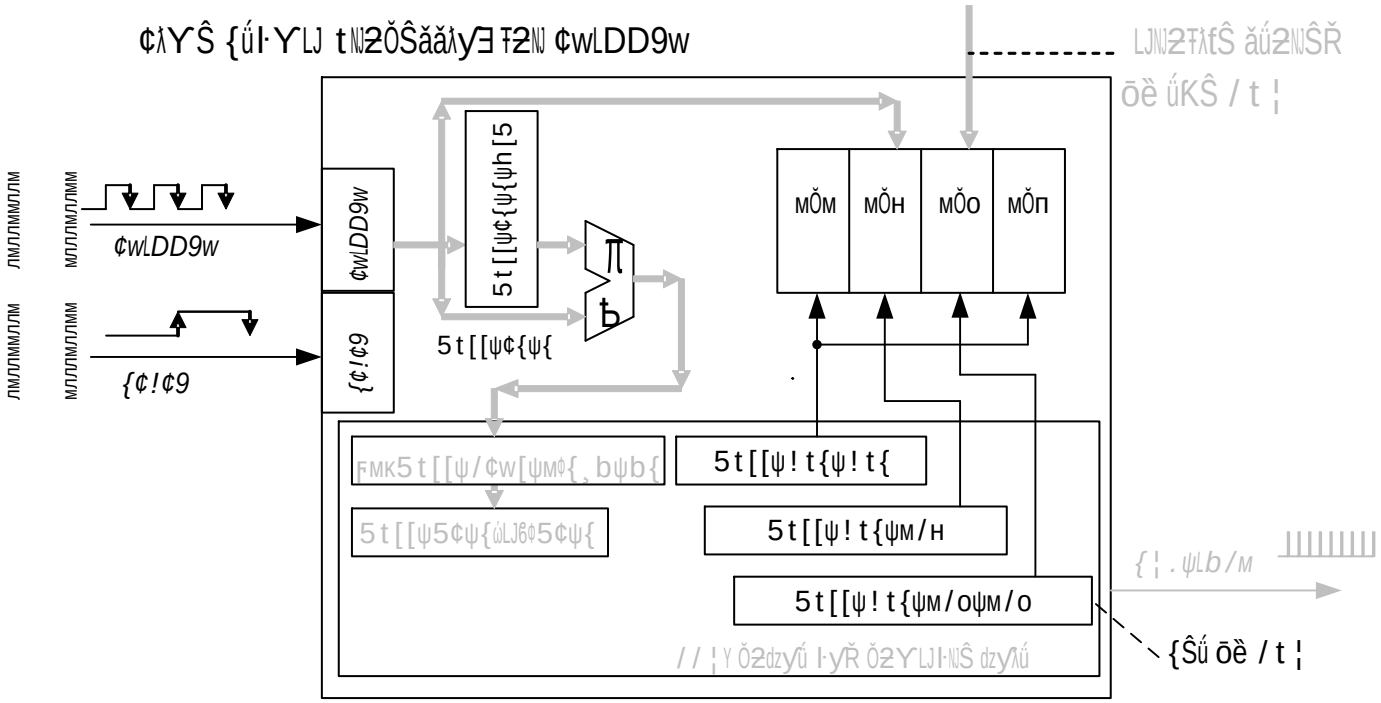
- RAM Region 1c: values of all *STATE* increments in FULL_SCALE, data width of 3 bytes used for 24 bit values. A write access to this region results in an interrupt to the CPU, when enabled. Address range: 0x0600 – 0x09FC

In RAM region 1c there is a difference in the amount of data. While for the RAM regions 1c1, 1c3 and 1c4 there are 2*(DPLL_SNU + 1 – DPLL_SYN_NS) entries for **DPLL_CTRL_1.SYSF** =0 or 2*(DPLL_SNU+1) – DPLL_SYN_NS entries for **DPLL_CTRL_1.SYSF** =1, for the RAM region 1c2 there are 2*(DPLL_SNU+1) entries (see **DPLL_CTRL_0** and **DPLL_CTRL_1** registers). For the latter also the virtual events are considered, that means the gap is divided into equidistant parts each having the same position share as increments without a gap. For that reason the CPU must extend the stored **DPLL_TSF_S[p].TSF_S** values in the RAM region 1c2 before the DPLL_APS_1C3_1C3 is written. The write access to DPLL_APS_1C3_1C3 sets the **DPLL_STATUS.SYS** bit in order to show the end of the synchronization process. Only when the **DPLL_STATUS.SYS** bit is set the **DPLL_ID_PMTR_[n].ID_PMTR** values can consider duration that is more than the last increment for the action prediction by setting DPLL_NUSC_NUSE to a corresponding value.

Note:

RAM regions 1b and 1c have a common port.

Figure 152 Address Pointer for RAM 1c



The address pointers for RAM region 1c are shown in the diagram above. While the address pointer DPLL_APS_APS points to the RAM regions 1c1 and 1c4, the address pointer DPLL_APS_1C2 points to the time stamp field in the region 1c2. This is necessary, because in the time stamp field the gaps are extended to the number of nominal increments (see explanation above and also the synchronization procedure explained in chapter *21.9.6.1 "Synchronization description"*).
The address pointer DPLL_APS_1C3_1C3 is set by the CPU when the position is known and therefore, the relation to the other address pointers is calculated. This setting of this profile address pointer synchronizes the RAM fields to one another. The synchronization is shown in the **DPLL_STATUS** register (see chapter **DPLL_STATUS**) by the **DPLL_STATUS.SYS** bit.

21.6.11.2 RAM Region 2

The RAM region 2 size ranges from 1.5 to 12 Kbytes and is used to store measured and calculated values for increments of *TRIGGER* . The address information is explained in chapter *21.14 " DPLL RAM Region 2 value description"* while the meaning is explained in this chapter *21.11 " DPLL Registers Description"* .
The memory size is set by device configuration variable DPLL_RR2_MEM_SIZE (see *86 "GTM Device Configuration Variables, Fixed or Derived Values"*).

Since there are upto 512 *TRIGGER* events in HALF_SCALE, the fields 2a, b, c and d must have up to 1024 storage places each. For 3 bytes word size this means up to 12 k bytes of RAM region 2.

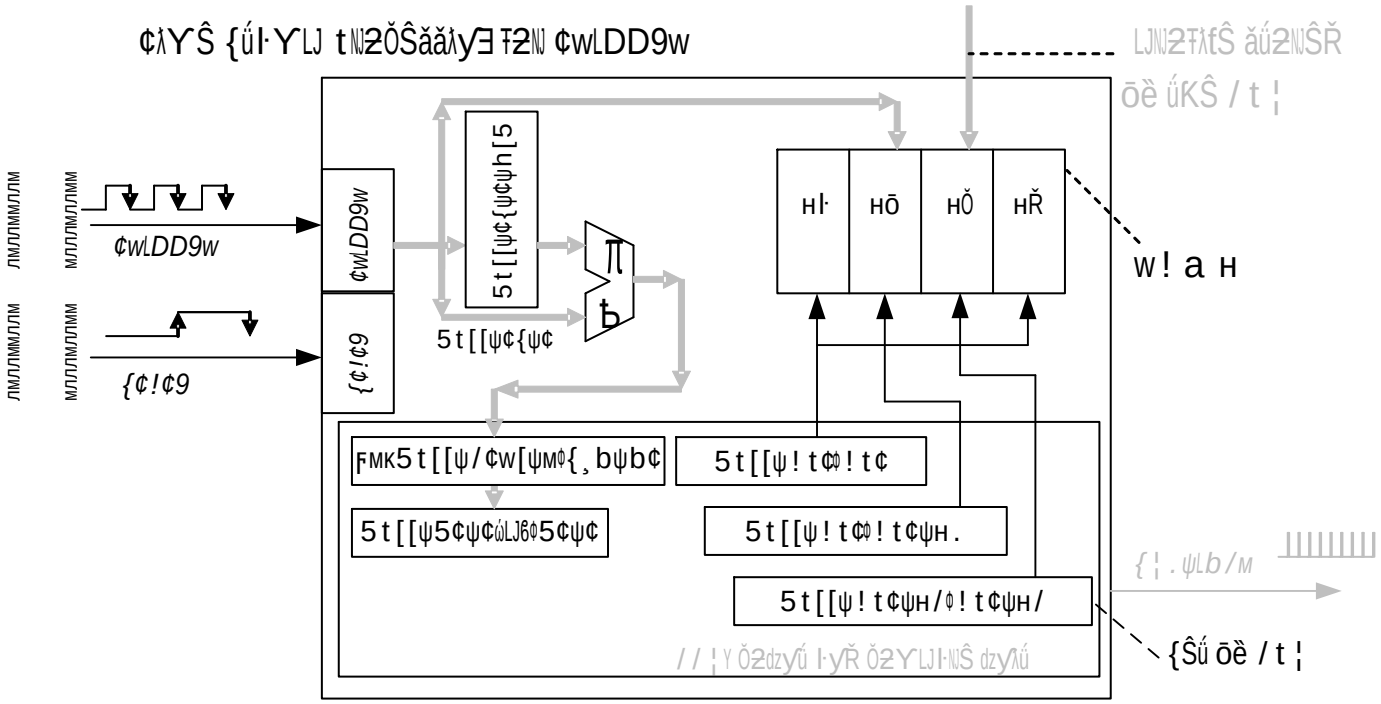
In order to save RAM size for configurations with less *TRIGGER* events the RAM is configurable by the offset switch Register **DPLL_OSW** (0x001C) and the address offset value register of RAM region 2 **DPLL_AOSV_2** (0x0020). The RAM is to be initialized by the DPLL

after HW-reset. All RAM cells must have a zero value after performing the initialize procedure. The DPLL is only available after finishing this procedure.

In RAM region 2, there is a difference in the amount of data. While there are 2*(**DPLL_CTRL_0.TNU** + 1 - **DPLL_CTRL_1.SYN_NT**) entries for the RAM regions 2a, 2c and 2d, for the RAM region 2b there are 2*(**DPLL_CTRL_0.TNU** + 1) entries (see **DPLL_CTRL_0** and **DPLL_CTRL_1** registers). For the latter also the virtual events are considered, that means the gap is divided into equidistant parts each having the same position share as increments without a gap. For this reason, the CPU must extend the stored **DPLL_TSF_T[p].TSF_T** values in the RAM region 2b before the **DPLL_APT_2C.APT_2C** is written.

The write access to **DPLL_APT_2C.APT_2C** sets the **DPLL_STATUS.SYT** bit to show the end of the synchronization process. Only when the **DPLL_STATUS.SYT** bit is set the PMTR values can consider more than the last increment duration for the action prediction by setting **DPLL_NUTC.NUTE** to a value greater than one.

Figure 153 Address Pointer for RAM 2



The address pointers for RAM region 2 are shown in the diagram above. While the address pointer **DPLL_APT.APT** points to the RAM regions 2a and 2d, the address pointer **DPLL_APT.APT_2B** points to the time stamp field in the region 2b. This is necessary, because in the time stamp field the gaps are extended to the number of nominal increments (see explanation above and also to the synchronization procedure explained in chapter *21.9.6.1 "Synchronization description"*).
The address pointer **DPLL_APT_2C.APT_2C** is set by the CPU when the position is known and therefore, the relation to the other address pointers is calculated. This setting of this profile address pointer synchronizes the RAM fields to one another. The synchronization is shown in the **DPLL_STATUS** register (see chapter **DPLL_STATUS**) by the **DPLL_STATUS.SYT** bit.

21.6.12 Software reset and DPLL deactivation

The DPLL module allows different options of deactivation and/or reset.
To stop the operation of the DPLL module it is possible to deactivate the DPLL by setting **DPLL_CTRL_1.DEN** = 0. This stops the calculations for the generation of the sub increments and the actions. Some control register areas are only configurable in this mode, but not all register signals are set to an initial state. The RAM memory is not affected by DPLL deactivation at all. The behavior of the DPLL output signals and registers when deactivated is described in this document.
The deeper option to reconfigure the DPLL is the use of the software reset. When the DPLL module is deactivated, the setting **DPLL_CTRL_1.SWR** = 1 performs a reset of all DPLL registers and state controllers. The RAM memory is not affected by the software reset at all. After the software reset the DPLL module remains in deactivated state and the control registers must be configured again before operation (activation by **DPLL_CTRL_1.DEN** = 1) can start again.
The RAM modules can be reset (written to all zero) by activation of the memory init control bit **DPLL_RAM_INI.INIT_RAM** .
If the RAM initialization is automatically done after power on reset or not, depends on the GTM implementation.
A special case is the configuration of the control bit **DPLL_CTRL_11.STATE_EXT** . If this bit shall be modified during operation a software reset of the DPLL module is strongly recommended. A RAM initialization should also be considered depending on the given application case.

21.6.12.1 DPLL Software reset

The configuration register **DPLL_CTRL_1** is used for software reset of the DPLL module.

Writing via the configuration interface the value 0b1 to the bitfield **DPLL_CTRL_1.SWR** while **DPLL_CTRL_1.DEN** ==0, will immediately reset the content of the following registers to the initial hardware reset state.

Internal registers inside the DPLL:

- **DPLL_CTRL_0**
- **DPLL_CTRL_1**
- **DPLL_CTRL_2**
- **DPLL_CTRL_3**
- **DPLL_CTRL_4**
- **DPLL_CTRL_5**
- **DPLL_ACT_STA**
- **DPLL_OSW**
- **DPLL_AOSV_2**
- **DPLL_APT**
- **DPLL_APS**
- **DPLL_APT_2C**
- **DPLL_APS_1C3**
- **DPLL_NUTC**
- **DPLL_NUSC**
- **DPLL_NTI_CNT**
- **DPLL_IRQ_NOTIFY**
- **DPLL_IRQ_EN**
- **DPLL_IRQ_MODE**
- **DPLL_EIRQ_EN**
- **DPLL_INC_CNT1**
- **DPLL_INC_CNT2**
- **DPLL_APT_SYNC**
- **DPLL_APS_SYNC**
- **DPLL_TBU_TS0_T**
- **DPLL_TBU_TS0_S**
- **DPLL_ADD_IN_LD1**
- **DPLL_ADD_IN_LD2**
- **DPLL_STATUS**
- **DPLL_ID_PMTR_[n]**
- **DPLL_CTRL_0_SHADOW_TRIGGER**
- **DPLL_CTRL_0_SHADOW_STATE**
- **DPLL_CTRL_1_SHADOW_TRIGGER**
- **DPLL_CTRL_1_SHADOW_STATE**
- **DPLL_RAM_INI**
- **DPLL_TSAC[n]**
- **DPLL_PSAC[n]**
- **DPLL_ACB_[n]**
- **DPLL_CTRL_11**
- **DPLL_THVAL2**
- **DPLL_TIDEL**
- **DPLL_SIDEL**
- **DPLL_CTN_MIN**
- **DPLL_CTN_MAX**
- **DPLL_CSN_MIN**
- **DPLL_CSN_MAX**
- **DPLL_STA**
- **DPLL_INCF1_OFFSET**

- **DPLL_INCF2_OFFSET**
- **DPLL_DT_T_START**
- **DPLL_DT_S_START**
- **DPLL_STA_MASK**
- **DPLL_STA_FLAG**
- **DPLL_INC_CNT1_MASK**
- **DPLL_INC_CNT2_MASK**
- **DPLL_NUSC_EXT1**
- **DPLL_NUSC_EXT2**
- **DPLL_APS_EXT**
- **DPLL_APS_1C3_EXT**
- **DPLL_APS_SYNC_EXT**
- **DPLL_CTRL_EXT**
- **DPLL_SW_TRIG**
- **DPLL_MP_T**
- **DPLL_MP_S**
- **DPLL_CTRL_12**

21.7 Prediction of the current increment duration

21.7.1 The use of increments in the past

Past values to be considered for the prediction of *TRIGGER* Signal

In order to take into account values of increments for *TRIGGER* s in the past, the **DPLL_NUTC.NUTE** value is configured to determine the number of past values. In addition, the **DPLL_NUTC.VTN** has a value according to the number of virtual increments in the **DPLL_NUTC.NUTE** region. Because gaps come into the **DPLL_NUTC.NUTE** region or leave it, the **DPLL_NUTC.VTN** value must be updated by the CPU until **DPLL_NUTC.NUTE** is set to HALF_SCALE or FULL_SCALE. For the RAM regions 2a and 2d the value **DPLL_NUTC.NUTE** – **DPLL_NUTC.VTN** is to be considered while for the RAM region 2b only the **DPLL_NUTC.NUTE** value is to be considered. This is due to the fact that the time stamp entries in a gap are extended to the number of nominal increments, but duration entries are not.

Past values to be considered for the prediction of *STATE* signal

In order to take into account values of increments for *STATE* in the past, the DPLL_NUSC_NUSE value is configured to determine the number of past values. In addition, the DPLL_NUSC_VSN has a value according to the number of virtual increments in the DPLL_NUSC_NUSE region. Because gaps come in to the DPLL_NUSC_NUSE region or leave it the DPLL_NUSC_VSN value must be updated by the CPU until DPLL_NUSC_NUSE is set to HALF_SCALE or FULL_SCALE. For the RAM regions 1c1 and 1c4 in the past the value DPLL_NUSC_NUSE–DPLL_NUSC_VSN is to be considered while for the RAM region 1c2 only the DPLL_NUSC_NUSE value is to be considered. This is due to the fact that time stamp entries in a gap are extended to the number of nominal increments, but duration entries are not.

21.7.2 Increment prediction in Normal Mode and for first PMSM forwards

For the prediction of increments and Actions in normal mode the values are calculated as described in the following equations.

The ascending order of calculation must be hold in order not to lose results still needed. It is important for *TRIGGER* values to calculate and store all values according to equations from *21.7.2 "Increment prediction in Normal Mode and for first PMSM forwards "* up to *21.8.2 "Action calculations for trigger signal backwards"* in the RAM region 2 before equations of chapter *21.8.5 "Update of RAM in Normal and Emergency Mode"* , while the last one *21.8.5.4 "Update of RAM by DT_T[p] and RDT_T[p] after calculation"* overwrites **DPLL_DT_T[p].DT_T** when **DPLL_NUTC.NUTE** (see chapter **DPLL_NUTC**) is set to the FULL_SCALE range. Because the old value of **DPLL_DT_T[p].DT_T** is also needed for equations of chapters *21.7.3.7 "Calculate the current increment (nominal value)"* and *21.8.1 "Action calculations for trigger signal forwards"* this value is stored temporarily at **DPLL_DT_T_ACT.DT_T_ACT** as shown by equation *21.7.2.1 "Calculate trigger time stamps"* or equation *21.7.2.2 "Calculate DPLL_DT_T_ACT.DT_T_ACT (nominal value)"* respectively until all prediction calculations are done and after that equations of chapters *21.8.5.1 "Update the time stamp values for TRIGGER signal "* , *21.8.5.2 "Extend the time stamp values for TRIGGER signal in forward direction"* , *21.8.5.3 "Extend the time stamp values for trigger signal in backward direction"* , and equation *[eq_140]* of chapter *21.8.5.4 "Update of RAM by DT_T[p] and RDT_T[p] after calculation"* updates **DPLL_DT_T[p].DT_T** : update **DPLL_DT_T[p].DT_T** after calculations of equation *[eq_100]* . For p= **DPLL_APT.APT** calculates in normal mode.

When using filter information of *TRIGGER_FT* , selected by **DPLL_CTRL_0_SHADOW_TRIGGER.IDT** =1, it must be distinguished by **DPLL_CTRL_0.IFP** , if this filter information is time or position related.
In order to make possible to perform the automatic resolution corrections of equations *[eq_3]* and *[eq_4]* the filter unit in TIM module must operate using the time stamp clock.

BOSCH

21.7.2.1 Calculate trigger time stamps

For calculation of time stamps use the filter delay information and an additional *TRIGGER* input delay value stored in register **DPLL_TIDEL.TIDEL** (initial zero)

[eq_1]

$TS\_T_1 = TRIGGER\_TS - TIDEL$

[eq_2]

IF(DPLL_CTRL_0_SHADOW_TRIGGER.IDT=1 and DPLL_CTRL_0_SHADOW_TRIGGER.IFP=0)

$TS\_T = TS\_T_1 - FTV\_Tx$

with

[eq_3]

IF(LOW_RES = 1 and DPLL_CTRL_1.TS0_HRT = 0)

FTV_Tx = FTV_T/8

[eq_4]

IF(LOW_RES = 0 or DPLL_CTRL_1.TS0_HRT = 1)

FTV_Tx = FTV_T

and

[eq_5]

IF(DPLL_CTRL_0_SHADOW_TRIGGER.IDT=1 and DPLL_CTRL_0_SHADOW_TRIGGER.IFP=1)

$TS\_T = TS\_T_1 - FTV\_T * (CDT\_TX/DPLL\_NMB\_T.NMB\_T)\_old$

this can be also calculated using the value of ADD_IN_CALN:

[eq_6]

IF(DPLL_CTRL_0_SHADOW_TRIGGER.IDT=1 and DPLL_CTRL_0_SHADOW_TRIGGER.IFP=1)

$TS\_T = TS\_T_1 - FTV\_T * (1/ADD\_IN\_CALN\_old)$

For (**DPLL_CDT_TX.CDT_TX** / **DPLL_NMB_T.NMB_T**)_old and (1/ADD_IN_CALN_old) consider values, calculated for the last increment; position related filter values are only considered up to at least 1 ms time between two *TRIGGER* events. The reciprocal value is stored using a 32 bit fractional part, while only the 24 lower bits are used (for explanation see note at **DPLL_CTRL_0.RMO**). The value of 1/ADD_IN_CALN_old or (**DPLL_CDT_TX.CDT_TX** / **DPLL_NMB_T.NMB_T**)_old is set to 0xFFFFFF in the case of an overflow.

DPLL_CDT_TX.CDT_TX is the predicted duration of the last *TRIGGER* increment and **DPLL_NMB_T.NMB_T** the calculated number of *SUB_INC1* events in the last increment, because the new calculations are done by equations *21.7.4.4 "Calculate the current increment value"* and *21.9.3.1 "Calculate the number of pulses to be sent in normal mode using the automatic end mode condition"* for the current increment after that. Therefore in equation *[eq_6]* , the value *ADD_IN* of the last increment is used (see equations *21.9.3 "Sub pulse generation for DPLL_CTRL_1.SMC=0"*). **DPLL_NUTC.SYN_T_OLD** is the number of *TRIGGER* events including missing *TRIGGER* s as specified in the **DPLL_NUTC** register for the last increment, with the initial value of 1.

For storage of time stamps in the RAM see also equations of chapters *21.8.5.1 "Update the time stamp values for TRIGGER signal "* ff. after calculation of Actions, chapter *21.8.5.1 "Update the time stamp values for TRIGGER signal "* .

21.7.2.2 Calculate DPLL_DT_T_ACT.DT_T_ACT (nominal value)

[eq_7]

DT_T_ACT= (TS_T – TS_T_OLD)/SYN_T_OLD

For the case **DPLL_STATUS.SYT** =0 (still no synchronization to the profile) the values **DPLL_NUTC.SYN_T** and **DPLL_NUTC.SYN_T_OLD** are still assumed as having the value 1.

Correct the current increment duration value got by equation *[eq_7]* in the case of physical deviations (**DPLL_CTRL_11.ADT** =1) by

[eq_8]

$DT\_T\_ACT = DT\_T\_ACT * (1 - PDC\_T + PDC\_T^2 - PDC\_T^3)$

with the relative correction value for the last increment

[eq_9]

IF(DPLL_CTRL_1.SMC=0)

PDC_T = PD_OLD / (DPLL_CTRL_0_SHADOW_TRIGGER.MLT+1)

[eq_10]

IF(DPLL_CTRL_1.SMC=1)

PDC_T = PD_OLD / (DPLL_MLS1.MLS1)

The term $(1 - PDC\_T + PDC\_T^2 - PDC\_T^3)$ of equation *[eq_8]* is representing the third order Taylor series of the term 1/(1+PDC_T), which is chosen to reduce the additional time delay due to the more complex computation.

21.7.2.3 Calculate RDT_T_ACT (nominal value)

[eq_11]

RDT_T_ACT = 1 / DT_T_ACT

21.7.2.4 Calculate QDT_T_ACT

Relation of the recent last two increment values for **DPLL_APT.APT** =p in forward direction (*DIR1* =0)

[eq_12]

QDT_T_ACT = DT_T_ACT * RDT_T[p-1]

QDT_T_ACT as well as QDT_T[p-pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

21.7.2.5 Calculate the error of last prediction

When pq = **DPLL_NUTC.NUTE** - **DPLL_NUTC.VTN** , consider only the last valid prediction values for *DIR1* =0 for the error calculation. Calculate the error of the last prediction when using only RDT_T_FS1, **DPLL_DT_T[p-pq].DT_T** , **DPLL_DT_T[p-pq-1].DT_T** and **DPLL_DT_T[p-1].DT_T** for the prediction of **DPLL_DT_T[p].DT_T** :

[eq_13]

EDT_T = DT_T_ACT - (DT_T[p-1] * QDT_T[p-pq])

with

[eq_14]

IF(DPLL_NUTC.FST= 0)

QDT_T[p-pq] = DT_T[p-pq] * RDT_T[p-pq-1]

[eq_15]

IF(DPLL_NUTC.FST=1)

QDT_T[p-pq] = DT_T[p-pq] * RDT_T_FS1

DPLL_NUTC.FST = 1 means **DPLL_NUTC.NUTE** =FULL_SCALE
while QDT_T_ACT as well as QDT_T[p-pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

21.7.2.6 Calculate the weighted average error

[eq_16]

IF(DPLL_STATUS.SYT=1)

MEDT_T := (EDT_T + MEDT_T) / 2

21.7.2.7 Calculate the current increment value

[eq_17]

IF(DPLL_CTRL_11.ADT=0)

CDT_TX _nom = (DT_T_ACT + MEDT_T) * QDT_T[p-pq+1]

[eq_18]

IF(DPLL_CTRL_11.ADT=1)

CDT_TX _nom_corr = CDT_TX _nom * (1 + CDC_T)

with:

[eq_19]

IF(DPLL_CTRL_1.SMC=0)

CDC_T = PD / (DPLL_CTRL_0_SHADOW_TRIGGER.MLT+1)

[eq_20]

IF(DPLL_CTRL_1.SMC=1)

CDC_T = PD / (DPLL_MLS1.MLS1)

and

[eq_21]

IF(pq > 1)

QDT_T[p-pq+1] = DT_T[p-pq+1] * RDT_T[p-pq]

and for pq=1 use equation *21.7.2.4 "Calculate QDT_T_ACT"* .

while QDT_T_ACT as well as QDT_T[p-pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

The **DPLL_CDT_TX_NOM.CDT_TX_NOM** value is limited by the relation

[eq_22]

DPLL_CTN_MIN.CTN_MIN < CDT_TX_nom < DPLL_CTN_MAX.CTN_MAX

When the calculated value exceeds one of the limits, it is replaced by the corresponding limit value.

The expected duration to the next *TRIGGER* event is:

[eq_23]
IF(DPLL_CTRL_11.ADT=0)
CDT_TX = CDT_TX _nom * SYN_T

[eq_24]
IF(DPLL_CTRL_11.ADT=1)
CDT_TX = CDT_TX _nom_corr * SYN_T

QDT_T[p-pq+1] uses a 6 bit integer part and an 18 bit fractional part.

Note:

In the case of an overflow in equations *[eq_17]* or *[eq_24]* sets the value to 0xFFFFFF and the corresponding **DPLL_STATUS.CTO** bit. In the case of negative values set **DPLL_CDT_TX.CDT_TX** to 0x0 without any effect to the **DPLL_STATUS.CTO** bit.

21.7.3 Increment prediction in Emergency Mode and for second PMS-M forwards

The ascending order of calculations for *STATE* and storage of the values in the RAM region 1c must be hold in order not to lose results still needed. The same considerations as done for **DPLL_DT_T_ACT.DT_T_ACT** are valid for **DPLL_DT_S_ACT.DT_S_ACT** (chapters *21.8.5.5 "Update the time stamp values for STATE signal"* , *21.8.5.6 "Extend the time stamp values for STATE signal"* , *21.8.5.7 "Extend the time stamp values for STATE signal for backward direction"* , and equations of chapter *21.8.5.8 "Update of RAM by DT_S[p] and RDT_S[p] after calculation"*): update **DPLL_DT_S[p].DT_S** only after calculations of equation *[eq_113]* or *[eq_126]* .

When using filter information of **DPLL_FTV_S.STATE_FT** , selected by **DPLL_CTRL_0_SHADOW_STATE.IDS** =1, it must be distinguished by **DPLL_CTRL_0_SHADOW_STATE.IFP** , if this filter information is time or position related.
In order to make possible to perform the automatic resolution corrections of equation *[eq_27]* the filter unit in TIM must operate using the time stamp clock.

21.7.3.1 Calculate STATE signal time stamps

For calculation of time stamps use the filter delay information, the additional *STATE* input delay value stored in the register **DPLL_SIDEL.SIDEL** (initial zero) and use p=DPLL_APS_APS while *DIR2* =0:

[eq_25]
$TS\_S_{1} = STATE\_TS - SIDEL$

[eq_26]
IF(DPLL_CTRL_0_SHADOW_STATE.IDS=1 and DPLL_CTRL_0_SHADOW_STATE.IFP=0)
$TS\_S = TS\_S_{1} - FTV\_Sx$

with

[eq_27]
IF(*LOW_RES* = 1 and DPLL_CTRL_1.TS0_HRS = 0)
FTV_Sx = FTV_S / 8

[eq_28]
IF(*LOW_RES* = 0 or DPLL_CTRL_1.TS0_HRS = 1)
FTV_Sx = FTV_S

and

[eq_29]
IF(DPLL_CTRL_0_SHADOW_STATE.IDS=1 and DPLL_CTRL_0_SHADOW_STATE.IFP=1)
$TS\_S = TS\_S_{1} - FTV\_S * (CDT\_SX / NMB\_S)\_old$

this can also be calculated using the value of ADD_IN_CALE:

[eq_30]
IF(DPLL_CTRL_0_SHADOW_STATE.IDS=1 and DPLL_CTRL_0_SHADOW_STATE.IFP=1)
$TS\_S = TS\_S_{1} - FTV\_S * (1 / ADD\_IN\_CALE)\_old$

see also equations in chapter *21.7.2.1 "Calculate trigger time stamps"* for *TRIGGER* .

For (**DPLL_CDT_SX.CDT_SX** / **DPLL_NMB_S.NMB_S**)_old and (1/ADD_IN_CALE)_old consider values, calculated for the last increment; position related filter values are only considered up to at least 1 ms time between two *STATE* events. The reciprocal value is stored using a 32-bit fractional part, while only the 24 lower bits are used (for explanation see note at **DPLL_CTRL_0.RMO**). The value of 1/ADD_IN_CALE_old or (**DPLL_CDT_SX.CDT_SX** / **DPLL_NMB_S.NMB_S**)_old is set to 0xFFFFFF in the case of an overflow.

Note:

DPLL_CDT_SX.CDT_SX is the predicted duration of the last *STATE* increment and **DPLL_NMB_S.NMB_S** the calculated number of *SUB_INC1* events in the last increment, because the new calculations are done by equations *21.7.3.7 "Calculate the current increment (nominal value)"* or *21.7.5.4 "Calculate the current increment value"* and equation *[eq_190]* respectively for the current increment after that. Therefore in equation *[eq_30]* , the value *ADD_IN* of the last increment is used (see equation *[eq_196]*). DPLL_NUSC_SYN_S_OLD is the number of increments including missing *STATE* s as specified in the **DPLL_NUSC** register for the last increment with the initial value of 1. The update to the RAM region 1c4 is done after all related calculations (see equations of chapter *21.8.5.8 "Update of RAM by DT_S[p] and RDT_S[p] after calculation"* for this reason).

21.7.3.2 Calculate DT_S_ACT (nominal value)

[eq_31]

DT_S_ACT= (TS_S – TS_S_OLD) / SYN_S_OLD

For the case **DPLL_STATUS.SYS** =0 (still no synchronization to the profile) the values DPLL_NUSC_SYN_S and DPLL_NUSC_SYN_S_OLD are still assumed as having the value 1.

Correct the current increment duration value got by equation *[eq_31]* in the case of physical deviations (**DPLL_CTRL_11.ADS** =1) by

[eq_32]

$DT\_S\_ACT = DT\_S\_ACT * (1 - PDC\_S + PDC\_S^{2} - PDC\_S^{3})$

with the relative correction value for the last increment

[eq_33]

IF(DPLL_CTRL_1.SMC=0)

PDC_S = PD_S_OLD/(DPLL_MLS1.MLS1)

[eq_34]

IF(DPLL_CTRL_1.SMC=1)

PDC_S = PD_S_OLD/(DPLL_MLS2.MLS2)

The term $(1 - PDC\_S + PDC\_S^{2} - PDC\_S^{3})$ of equation *[eq_32]* represents the third order Taylor series of the term 1/(1+PDC_S), which is chosen to reduce the additional time delay due to the more complex computation.

21.7.3.3 Calculate RDT_S_ACT (nominal value)

[eq_35]

RDT_S_ACT = 1 / DT_S_ACT

21.7.3.4 Calculate QDT_S_ACT

Relation of the last two increment values for **DPLL_APS_EXT.APS** =p in forward direction (*DIR2* =0)

[eq_36]

QDT_S_ACT = DT_S_ACT * RDT_S[p–1]

QDT_S_ACT as well as QDT_S[p–pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

21.7.3.5 Calculate the error of last prediction

With pq= DPLL_NUSC_NUSE–DPLL_NUSC_VSN when using QDT_S[p–pq] and **DPLL_DT_S[p–1].DT_S** for the prediction of **DPLL_DT_S[p].DT_S**

[eq_37]

EDT_S = DT_S_ACT – (DT_S[p–1] * QDT_S[p–pq])

and with

[eq_38]

IF(DPLL_NUSC_FSS=0)

QDT_S[p–pq] = DT_S[p–pq] * RDT_S[p–pq–1]

[eq_39]

IF(DPLL_NUSC_FSS=1)

QDT_S[p–pq] = DT_S[p–pq] * RDT_S_FS1

DPLL_NUSC_FSS=1 means DPLL_NUSC_NUSE=FULL_SCALE.
QDT_S_ACT as well as QDT_S[p–pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

21.7.3.6 Calculate the weighted average error

[eq_40]

IF(DPLL_STATUS.SYS=1)

MEDT_S = (EDT_S + MEDT_S) / 2

21.7.3.7 Calculate the current increment (nominal value)

[eq_41]
IF(DPLL_CTRL_11.ADS=0)
CDT_SX _nom = (DT_S_ACT + MEDT_S) * QDT_S[p-pq+1]
[eq_42]
IF(DPLL_CTRL_11.ADS=1)
CDT_SX _nom_corr = CDT_SX _nom * (1+CDC_S)

with:

[eq_43]
IF(DPLL_CTRL_1.SMC=0)
CDC_S = DPLL_ADT_T[p].PD/ (DPLL_MLS1.MLS1)
[eq_44]
IF(DPLL_CTRL_1.SMC=1)
CDC_S = DPLL_ADT_T[p].PD/ (DPLL_MLS2.MLS2)

and

[eq_45]
IF(pq>1)
QDT_S[p-pq+1] = DT_S[p-pq+1] * RDT_S[p-pq]

and for pq=1 use equation *[eq_36]* .

While QDT_S_ACT as well as QDT_S[p-pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

The **DPLL_CDT_SX_NOM.CDT_SX_NOM** value is limited by the relation

[eq_46]
DPLL_CSN_MIN.CSN_MIN < CDT_SX_nom < DPLL_CSN_MAX.CSN_MAX

When the calculated value exceeds one of the limits, it is replaced by the corresponding limit value.

The expected duration to the next *STATE* event is

[eq_47]
IF(DPLL_CTRL_11.ADT=0)
CDT_SX = CDT_SX _nom * SYN_T
[eq_48]
IF(DPLL_CTRL_11.ADT=1)
CDT_SX = CDT_SX _nom_corr * SYN_S

Note:
In the case of an overflow in equations *[eq_41]* or *[eq_47]* , *[eq_48]* set the value to 0xFFFFFF and the corresponding **DPLL_STATUS.CSO** bit in the **DPLL_STATUS** register. In the case of negative values set **DPLL_CDT_SX.CDT_SX** to 0x0 without any effect to the **DPLL_STATUS.CSO** bit.
All 5 steps above (chapter *21.7.3.1 "Calculate STATE signal time stamps"* in chapter *21.7.3.7 "Calculate the current increment (nominal value)"* are only needed in emergency mode. For the normal mode the calculations of equations in chapter *21.7.3.1 "Calculate STATE signal time stamps"* to chapter *21.7.3.4 "Calculate QDT_S_ACT"* are done solely in order to get the values needed for a sudden switch to emergency mode.

21.7.4 Increment prediction in Normal Mode and for first PMSM backwards

21.7.4.1 Calculate QDT_T_ACT backwards

[eq_49]
QDT_T_ACT = DT_T_ACT * RDT_T[p+1]

QDT_T_ACT as well as QDT_T[p+pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

21.7.4.2 Calculate of the error of last prediction

When pq = **DPLL_NUTC.NUTE** - **DPLL_NUTC.VTN** and *DIR1* =1 using only QT_T[p+pq] and **DPLL_DT_T[p+1].DT_T** for the prediction of **DPLL_DT_T[p].DT_T** .

[eq_50]
EDT_T = DT_T_ACT - (DT_T[p+1] * QDT_T[p+pq]

with

[eq_51]
IF(DPLL_NUTC.FST=0)
QDT_T[p+pq] = DPLL_DT_T[p].DT_T[p+pq] * DPLL_RDT_T[p].RDT_T[p+pq+1]

[eq_52]
IF(DPLL_NUTC.FST=1)
QDT_T[p+pq] = DT_T[p+pq] * RDT_T_FS1

DPLL_NUTC.FST =1 means **DPLL_NUTC.NUTE** =FULL_SCALE (see **DPLL_NUTC** register).
QDT_T_ACT as well as QDT_T[p+pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

21.7.4.3 Calculate the weighted average error

[eq_53]
IF(DPLL_STATUS.SYT=1)
MEDT_T := (EDT_T + MEDT_T) / 2

21.7.4.4 Calculate the current increment value

[eq_54]
IF(DPLL_CTRL_11.ADT=0)
CDT_TX _nom = (DT_T_ACT + MEDT_T) * QDT_T[p+pq-1]

[eq_55]
IF(DPLL_CTRL_11.ADT=1)
CDT_TX _nom_corr = CDT_TX _nom * (1 + CDC_T)

with

[eq_56]
IF(DPLL_CTRL_1.SMC=0)
CDC_T = PD / (MLT + 1)

see equation *[eq_19]*

[eq_57]
IF(DPLL_CTRL_1.SMC=1)
CDC_T = DPLL_ADT_T[p].PD/ (DPLL_MLS1.MLS1)

see equation *[eq_20]* and

[eq_58]
IF(pq>1)
QDT_T[p+pq-1] = DT_T[p+pq-1] * RDT_T[p+pq]

for pq=1 use equation *21.7.2.4 "Calculate QDT_T_ACT"*

while QDT_T_ACT as well as QDT_T[p+pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

The **DPLL_CDT_TX_NOM.CDT_TX_NOM** value is limited by the relation

[eq_59]
DPLL_CTN_MIN.CTN_MIN < CDT_TX_nom < DPLL_CTN_MAX.CTN_MAX

When the calculated value exceeds one of the limits, it is replaced by the corresponding limit value.

The expected duration to the next *TRIGGER* event

[eq_60]
IF(DPLL_CTRL_11.ADT=0)
DPLL_CDT_TX.CDT_TX = DPLL_CDT_TX_NOM.CDT_TX_NOM * DPLL_NUTC.SYN_T

see equation *[eq_17]*

[eq_61]
IF(DPLL_CTRL_11.ADT=1)
DPLL_CDT_TX.CDT_TX = CDT_TX_nom_corr * SYN_T

see equation *[eq_24]*

Note:

In the case of an overflow in equations *[eq_17]* or *[eq_24]* set the value to 0xFFFFFF and the corresponding **DPLL_STATUS.CTO** bit. In the case of negative values set **DPLL_CDT_TX.CDT_TX** (_nom) to 0x0.

21.7.5 Increment prediction in Emergency Mode and for second PMS-M backwards

21.7.5.1 Calculate QDT_S_ACT backwards

[eq_62]
QDT_S_ACT = DT_S_ACT * RDT_S[p+1]

QDT_S_ACT as well as QDT_S[p+pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

21.7.5.2 Calculate the error of the last prediction

While pq= DPLL_NUSC_NUSE–DPLL_NUSC_VSN, use only QDT_S[p+pq] and DT_S[p+1] for the prediction of **DPLL_DT_S[p].DT_S**
[eq_63]
EDT_S = DT_S_ACT – (DT_S[p+1] * QDT_S[p+pq])

with
[eq_64]
IF(DPLL_NUSC_FSS=0)
QDT_S[p–pq] = DPLL_DT_S[p].DT_S[p+pq] * DPLL_RDT_S[p].RDT_S[p+pq+1]
[eq_65]
IF(DPLL_NUSC_FSS=1)
QDT_S[p–pq] = DPLL_DT_T[p].DT_T[p+pq] * RDT_S_FS1

DPLL_NUSC_FSS=1 means DPLL_NUSC_NUSE=FULL_SCALE (see NUSC register).
QDT_S_ACT as well as QDT_S[p–pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

21.7.5.3 Calculate the weighted average error

[eq_66]
IF(DPLL_STATUS.SYS=1)
MEDT_S = (EDT_S + MEDT_S) / 2

21.7.5.4 Calculate the current increment value

[eq_67]
IF(DPLL_CTRL_11.ADS=0)
CDT_SX _nom = (DT_S_ACT + MEDT_S) * QDT_S[p+pq–1]
[eq_68]
IF(DPLL_CTRL_11.ADS=1)
CDT_SX _nom_corr = CDT_SX _nom * (1+CDC_S)

with:
[eq_69]
IF(DPLL_CTRL_1.SMC=0)
CDC_S = DPLL_ADT_T[p].PD/ (DPLL_MLS1.MLS1)

Compare with equation *[eq_43]*
[eq_70]
IF(DPLL_CTRL_1.SMC=1)
CDC_S = DPLL_ADT_T[p].PD/ (DPLL_MLS2.MLS2)

Compare with equation *[eq_44]* .
With
[eq_71]
IF(pq>1)
QDT_S[p+pq–1] = DT_S[p+pq–1] * RDT_S[p+pq]

for pq=1 use equation *21.7.5.1 "Calculate QDT_S_ACT backwards"* .

While QDT_S_ACT as well as QDT_S[p+pq] have a 24 bit value using a 6 bit integer part and an 18 bit fractional part.

The **DPLL_CDT_SX_NOM.CDT_SX_NOM** value is limited by the relation
[eq_72]
DPLL_CSN_MIN.CSN_MIN < DPLL_CDT_SX_NOM.CDT_SX_NOM < DPLL_CSN_MAX.CSN_MAX

(Compare with equation *[eq_46]*)
When the calculated value exceeds one of the limits, it is replaced by the corresponding limit value.

Then calculate the expected duration to the next *STATE* event

[eq_73]
IF(DPLL_CTRL_11.ADT=0)

DPLL_CDT_SX.CDT_SX = DPLL_CDT_SX_NOM.CDT_SX_NOM * DPLL_NUTC.SYN_T

[eq_74]
IF(DPLL_CTRL_11.ADT=1)

DPLL_CDT_SX.CDT_SX = CDT_SX _nom_corr * DPLL_NUSC_SYN_S

Compare with equation *[eq_47]*

Note:

In the case of an overflow in equations *[eq_41]* , *[eq_47]* set the value to 0xFFFFFF and the corresponding **DPLL_STATUS.CSO** bit. In the case of negative values set **DPLL_CDT_SX.CDT_SX** (**DPLL_CDT_SX_NOM.CDT_SX_NOM**) to 0x0.
All 5 calculation steps above (chapters *21.7.3.1 "Calculate STATE signal time stamps"* to *21.7.5.4 "Calculate the current increment value"*) are only needed in emergency mode. For the normal mode the calculations of equations *21.7.3.1 "Calculate STATE signal time stamps"* and *21.7.5.1 "Calculate QDT_S_ACT backwards"* are done solely in order to get the values needed for a sudden switch to emergency mode.

21.8 Calculations for Actions

As already shown for the calculation of the current interval by equations of chapter *21.7.2 "Increment prediction in Normal Mode and for first PMSM forwards "* to chapter *21.7.5 "Increment prediction in Emergency Mode and for second PMSM backwards"* for the prediction of Actions a similar calculation is to be done, as shown by the equations of chapter *21.8.1 "Action calculations for trigger signal forwards"* to chapter *21.8.4 "Action calculations for STATE signal backwards "* . The calculation of Actions is also needed when the DPLL is used for synchronous motor control applications (**DPLL_CTRL_1.SMC** =1, see **DPLL_CTRL_1** register). For action prediction purposes the measured time periods of the past (one FULL_SCALE back, when the corresponding **DPLL_NUTC.NUTE** or DPLL_NUSC_NUSE values are set properly by the CPU) are used. The calculation can be explained by the following assumptions, which are considerably simple:

Take the corresponding increments for prediction in the past and put the sum of it in relation to the increment (**DPLL_DT_T[p].DT_T** , **DPLL_DT_S[p].DT_S** , with p >= 0, which is represented by the time stamp difference) which is exactly one FULL_SCALE period in the past. Make a prediction for the coming sum of increments using the current measured increment (**DPLL_DT_T_ACT.DT_T_ACT** or **DPLL_-DT_S_ACT.DT_S_ACT** respectively) and add a weighted average error (equations in chapters *21.7.2.5 "Calculate the error of last prediction"* and chapter *21.7.2.6 "Calculate the weighted average error"* or chapter *21.7.3.5 "Calculate the error of last prediction"* and chapter *21.7.3.6 "Calculate the weighted average error"* respectively, calculated for one increment prediction) before multiplication with the relation PDT_T e.g. of equation *[eq_88]* or PDT_S e.g. of equation *[eq_101]* respectively in order to get the result as described by equations *[eq_87]* or *[eq_113]* respectively.

In order to avoid division operations instead of the increment (**DPLL_DT_T[p].DT_T** , **DPLL_DT_S[p].DT_S** , with p > 0) in the past its reciprocal value (**DPLL_RDT_T[p].RDT_T** , **DPLL_RDT_S[p].RDT_S** , with p > 0) is used, which is stored also in RAM. For the calculation of Actions always perform a new refined calculation as long as the resulting time stamp is not in the past. In the other case the **DPLL_TSAC[n].-TSAC** / **DPLL_PSAC[n].PSAC** values (time/position stamp of action calculated) is set to the time/position stamp of the last input event (*TRIGGER* / *STATE*), the **DPLL_ACT_STA.ACT_N[n]** bit is reset, while the corresponding **DPLL_ACT_STA.ACT_N[n]** bit in the shadow register is set. Each new **DPLL_ID_PMTR_[n].ID_PMTR** value will set this **DPLL_ACT_STA.ACT_N[n]** bit again and reset the corresponding shadow bit until a new calculation is performed.

Please make sure that the prediction parameters are chosen such that under all conditions (acceleration/deceleration) the values of PDT_T, PDT_S and **DPLL_DTA[n].DTA** respectively do not exceed the value 0xFFFFFF. This requirement can limit the predicted position range in the case of very low speed.

Action updates at highest speed
Up to 32 action values can be calculated. For the shortest increment duration (23.4 µs) not all of them can be updated with each active input event. Please note the following conditions and parameters for an estimation of possible results.
All time estimation values are given for a system clock frequency of 100 MHz and the assumption, that the calculation of the DPLL is not impeded by a remote read or write access to the DPLL RAMs. Each RAM access is to be considered by an additional delay of typically about 40 ns ($t_{remote\_RAM\_access}$). When using a different system clock frequency the calculation duration is extended accordingly. Further, the additional delay is also impacted by the type of access and the number of wait states on the AEI bus.

1. Typical time needed for basic operations (RAM update, pointer calculation and *SUB_INC1* and *SUB_INC2* generation for normal, emergency mode or one PMSM: t_{basic_0} = 9.9 µs.
2. Typical time needed basic operations (RAM update, pointer calculation and *SUB_INC1* and *SUB_INC2* generation for two PMSMs: t_{basic_1} = 11.0 µs.
3. Typical time needed to calculate one action: t_{Action_i} = 3.7 µs.

Note:

The above mentioned values are observed as worst case values, when both the state machines *TRIGGER* and *STATE* are in operation.

These values allow the calculation of at least 3 action values for each input event for all specified increments duration. The complete time needed for the basic operation, n action calculations and k remote RAM access operations can be calculated as follows
$t_{complete} = t_{basic\_0/1} + n*t_{Action\_i} + k*t_{remote\_RAM\_access}$.

Typical applications

Normal and emergency mode
For a typical application with the shortest increment duration of 100 µs in normal or emergency mode the calculation of up to 24 action values can be performed for each active input event.

One PMSM
For one PMSM and a typical shortest increment time of 39 µs there is the calculation of up to 7 Action values possible for each input event.

Two PMSMs with restricted Action calculations
When only one PMSM uses the Action calculation service and the shortest increment duration is 39 µs, there can up to 7 Actions served for each active input event.

Two PMSMs with unrestricted Action calculations
When 2 PMSMs are used and both use the Action calculation service at a minimal increment duration of 39 µs there are up to 7 Action calculations possible for each of the two engines – that means up to 14 Action calculations per increment in average.

21.8.1 Action calculations for trigger signal forwards

Valid for **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0 or for **DPLL_CTRL_1.SMC** =1 with
p= **DPLL_APT.APT_2B** , pt= **DPLL_APT.APT** , pm=NA[n] (part w), mb=NA[n](part b)/1024, **DPLL_NUTC.NUTE** – **DPLL_NUTC.VTN** =pq, **DPLL_NUTC.NUTE** =pn

Note:
All 5 steps in equations *21.8.1.1 "For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w)"* to *[eq_87]* are only calculated in normal mode.

21.8.1.1 For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w)

[eq_75]
IF(DIR1=0 and pq>pm)
PDT_T[n] = (TSF_T[p+pm–pn] – TSF_T[p–pn] + mb* DT_Tx[pt–pq+1]) * RDT_T[pt–pq]

with
[eq_76]
IF(DPLL_CTRL_1.TS0_HRT=0)
DT_Tx[pt–pq+1] = DPLL_DT_T[p].DT_T[pt–pq+1]

Compare with equations *[eq_79]* , *[eq_82]* or
[eq_77]
IF(DPLL_CTRL_1.TS0_HRT=1)
DT_Tx[pt–pq+1] = DPLL_DT_T[p].DT_T[pt–pq+1]/8

Compare with equations *[eq_80]* , *[eq_83]* and the multiplication with mb means the fractional part of NA[n].

For **DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0, calculate for *DIR1* =0 all 32 Actions in forward direction, if requested; in the case **DPLL_CTRL_1.SMC** =1 calculate up to 16 actions 0 to 15 depending on the *TRIGGER* input.

21.8.1.2 For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w) and DPLL_NUTC.NUTE= 2*(DPLL_CTRL_0.TNU+1)

[eq_78]
IF(for DPLL_STATUS.SYT=1 and DPLL_NUTC.NUTE= 2*(DPLL_CTRL_0.TNU+1) and pq>pm and DIR1=0:)
PDT_T[n] = (TSF_T[p+pm] – TSF_T[p]+ mb* DT_Tx[pt–pq+1]) * RDT_T[pt]

with
[eq_79]
IF(DPLL_CTRL_1.TS0_HRT=0)
DT_Tx[pt–pq+1] = DPLL_DT_T[p].DT_T[pt–pq+1]

Compare with equations *[eq_76]* , *[eq_82]* or
[eq_80]
IF(DPLL_CTRL_1.TS0_HRT=1)
DT_Tx[pt–pq+1] = DPLL_DT_T[p].DT_T[pt–pq+1]/8

Compare with equations *[eq_77]* , *[eq_83]*

21.8.1.3 For DPLL_NUTC.NUTE – DPLL_NUTC.VTN <= NA[n] (part w) and DPLL_NUTC.NUTE > 1

[eq_81]
IF(DIR1=0 and DPLL_NUTC.NUTE – DPLL_NUTC.VTN=pq and pq <= pm and pn>1 and pt=DPLL_APT.APT)
PDT_T[n] = (pm+mb)*DT_Tx[pt–pq+1] * RDT_T[pt–pq]

with

[eq_82]
IF(DPLL_CTRL_1.TS0_HRT=0)
DT_Tx[pt–pq+1] = DPLL_DT_T[p].DT_T[pt–pq+1]

Compare with equations *[eq_79]* , *[eq_76]* or

[eq_83]
IF(DPLL_CTRL_1.TS0_HRT=1)
DT_Tx[pt–pq+1] = DPLL_DT_T[p].DT_T[pt–pq+1] /8

Compare with equations *[eq_77]* , *[eq_80]*

Note:

Make the calculations above in chapters *21.8.1.1 "For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w)"* , *21.8.1.2 "For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w) and DPLL_NUTC.NUTE= 2*(DPLL_CTRL_0.TNU+1)"* , *21.8.1.3 "For DPLL_NUTC.NUTE – DPLL_NUTC.VTN <= NA[n] (part w) and DPLL_NUTC.NUTE > 1"* before updating the **DPLL_TSF_T[p].TSF_T** values according to equations in chapter *21.8.5.1 "Update the time stamp values for TRIGGER signal "* .

21.8.1.4 For DPLL_NUTC.NUTE = 1

[eq_84]
IF(pn=1)
PDT_T[n] = (pm+mb)* DT_T_ax* RDT_T[pt–1]

with

[eq_85]
IF(DPLL_CTRL_1.TS0_HRT=0)
DT_T_ax = DT_T_ACT

Compare with equations *[eq_129]* , *[eq_98]* or

[eq_86]
IF(DPLL_CTRL_1.TS0_HRT=1)
DT_T_ax = DT_T_ACT/8

Compare with equations *[eq_99]* , *[eq_133]*

Note:

For the relevant last increment add the fractional part of **DPLL_DT_T_ACT.DT_T_ACT** as described in **DPLL_NA[n]** .

21.8.1.5 Calculate the duration value until Action

[eq_87]
DTA[i] = (DT_T_ACT + MEDT_T)* PDT_T[i]

Compare with equation *[eq_100]*

21.8.2 Action calculations for trigger signal backwards

valid for **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0 or for **DPLL_CTRL_1.SMC** =1 with
p= **DPLL_APT.APT_2B** , pt= **DPLL_APT.APT** , pm= **DPLL_NA[n]** (part w), mb= **DPLL_NA[n]** (part b)/1024, pq= **DPLL_NUTC.NUTE** – **DPLL_NUTC.VTN** and pn= **DPLL_NUTC.NUTE**

For **DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0 calculate for *DIR1* =1 all 32 Actions in backward direction for special purposes; in the case **DPLL_CTRL_1.SMC** =1 calculate up to 16 Actions 0 to 15 in dependence of the *TRIGGER* input.

Note:

All 5 steps in equations *21.8.2.1 "For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w)"* to *21.8.2.5 "Calculate the duration value for an Action"* are only calculated in normal mode or when **DPLL_CTRL_1.SMC** =1.

21.8.2.1 For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w)

[eq_88]
IF(DIR1=1 and pq>pm)

PDT_T[n] = (TSF_T[p–pm+pn] – TSF_T[p+pn] + mb* DT_Tx[pt+pq–1]) * RDT_T[pt+pq]

with

[eq_89]
IF(DPLL_CTRL_1.TS0_HRT=0)

DT_Tx[pt+pq–1] = DPLL_DT_T[p].DT_T[pt+pq–1]

Compare with equations *[eq_92]* , *[eq_95]* or

[eq_90]
IF(DPLL_CTRL_1.TS0_HRT=1)

DT_Tx[pt+pq–1] = DPLL_DT_T[p].DT_T[pt+pq–1] /8

Compare with equations *[eq_93]* , *[eq_96]*

21.8.2.2 For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w) and DPLL_NUTC.NUTE= 2*(DPLL_CTRL_0.TNU+1)

[eq_91]
IF(DPLL_STATUS.SYT=1 and DPLL_NUTC.NUTE = 2*(DPLL_CTRL_0.TNU+1) and pq>pm and DPLL_NUTC.VTN=2*DPLL_CTRL_1.SYN_NT and DIR1=1)

PDT_T[n] = (TSF_T[p–pm] – TSF_T[p]+ mb* DT_Tx[pt+pq–1]) * RDT_T[pt]

Note:

In this case **DPLL_NUTC.NUTE** – **DPLL_NUTC.VTN** = 2*(**DPLL_CTRL_0.TNU** +1– **DPLL_CTRL_1.SYN_NT**)

with

[eq_92]
IF(DPLL_CTRL_1.TS0_HRT=0)

DT_Tx[pt+pq–1] = DPLL_DT_T[p].DT_T[pt+pq–1]

Compare with equations *[eq_89]* , *[eq_95]* or

[eq_93]
IF(DPLL_CTRL_1.TS0_HRT=1)

DT_Tx[pt+pq–1] = DPLL_DT_T[p].DT_T[pt+pq–1] /8

Compare with equations *[eq_90]* , *[eq_96]*

Note:

Make the calculations above in chapters *21.8.2.1 "For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w)"* and *21.8.2.2 "For DPLL_NUTC.-NUTE – DPLL_NUTC.VTN > NA[n] (part w) and DPLL_NUTC.NUTE= 2*(DPLL_CTRL_0.TNU+1)"* before updating the **DPLL_TSF_T[p].TSF_T** values according to equations in chapter *21.8.5.1 "Update the time stamp values for TRIGGER signal "* .

21.8.2.3 For DPLL_NUTC.NUTE – DPLL_NUTC.VTN <= NA[n] (part w) and DPLL_NUTC.NUTE > 1

[eq_94]
IF(DPLL_NUTC.NUTE – DPLL_NUTC.VTN=pq and pq <= pm and pn>1 and pt=DPLL_APT.APT)

PDT_T[n] = (pm+mb)*DT_Tx[pt+pq–1] * RDT_T[pt+pq]

with

[eq_95]
IF(DPLL_CTRL_1.TS0_HRT=0)

DT_Tx[pt+pq–1] = DPLL_DT_T[p].DT_T[pt+pq–1]

Compare with equations *[eq_89]* , *[eq_92]* or

[eq_96]
IF(DPLL_CTRL_1.TS0_HRT=1)

DT_Tx[pt+pq–1] = DPLL_DT_T[p].DT_T[pt+pq–1] /8

Compare with equations *[eq_90]* , *[eq_93]*

21.8.2.4 For DPLL_NUTC.NUTE = 1

[eq_97]
IF(pn=1)

PDT_T[n] = (pm+mb)* DT_T_ax * RDT_T[pt+1]

with

[eq_98]
IF(DPLL_CTRL_1.TS0_HRT=0)

DT_T_ax = DPLL_DT_T_ACT.DT_T_ACT

Compare with equations *[eq_129]* , *[eq_85]* or

[eq_99]
IF(DPLL_CTRL_1.TS0_HRT=1)

DT_T_ax = DPLL_DT_T_ACT.DT_T_ACT/8

Compare with equations *[eq_86]* , *[eq_133]*

For the relevant last increment add the fractional part of **DPLL_DT_T_ACT.DT_T_ACT** as described in **DPLL_NA[n]** .

21.8.2.5 Calculate the duration value for an Action

[eq_100]

DTA[n] = (DT_T_ACT + MEDT_T)* PDT_T[n]

Compare with equation *[eq_87]*

Use the results of equations *21.7.2.1 "Calculate trigger time stamps"* , *21.7.2.2 "Calculate DPLL_DT_T_ACT.DT_T_ACT (nominal value)"* , *21.7.2.5 "Calculate the error of last prediction"* and *21.7.2.6 "Calculate the weighted average error"* for the above calculation

21.8.3 Action calculations for STATE signal forwards

Valid for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 with p=DPLL_APS_1C2, pt=DPLL_APS_APS, pm= **DPLL_NA[i]** (part w) mb= **DPLL_NA[i]** (part b)/1024, DPLL_NUSC_NUSE – DPLL_NUSC_VSN= pq and DPLL_NUSC_NUSE=pn>pm

For **DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 calculate for *DIR2* =0 all NOAC Actions in forward direction, if requested; in the case **DPLL_CTRL_1.SMC** =1 calculate up to NOAC/2 Actions NOAC/2 to NOAC−1 in dependence of the *STATE* input.

Note:

All 5 steps of equations from chapter *21.8.3.1 "For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w)"* to chapter *21.8.3.5 "Calculate the duration value for an Action"* are only calculated in emergency mode or for **DPLL_CTRL_1.SMC** =1 in combination with **DPLL_CTRL_0_SHADOW_STATE.RMO** =1.

21.8.3.1 For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w)

[eq_101]
IF(DIR2=0 and pq>pm)

PDT_S[n] = (TSF_S[p+pm−pn] − TSF_S[p−pn] + mb* DT_Sx[pt−pq+1] * RDT_S[pt−pq]

with

[eq_102]
IF(DPLL_CTRL_1.TS0_HRS=0)

DT_Sx[pt−pq+1]= DPLL_DT_S[p].DT_S[pt−pq+1]

Compare with equations *[eq_105]* , *[eq_108]* or

[eq_103]
IF(DPLL_CTRL_1.TS0_HRS=1)

DT_Sx[pt−pq+1] = DPLL_DT_S[p].DT_S[pt−pq+1]/8

Compare with equations *[eq_106]* , *[eq_109]*

BOSCH

21.8.3.2 For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w) and DPLL_NUSC_NUSE= 2*(DPLL_SNU+1)

[eq_104]
IF(DPLL_STATUS.SYS=1 and DPLL_NUSC_NUSE=2*(DPLL_SNU+1) and pq>pm and DPLL_CTRL_1.SYSF=0 and DPLL_NUSC_VSN=2*DPLL_SYN_NS)

PDT_S[n] = (TSF_S[p+pm] – TSF_S[p]+ mb*DT_Sx[pt–pq+1]) * RDT_S[pt]

Note:

In this case DPLL_NUSC_NUSE – DPLL_NUSC_VSN = 2*(DPLL_SNU+1–DPLL_SYN_NS)

with

[eq_105]
IF(DPLL_CTRL_1.TS0_HRS=0)

DT_Sx[pt–pq+1] = DPLL_DT_S[p].DT_S[pt–pq+1]

Compare with equations *[eq_102]* , *[eq_108]* or

[eq_106]
IF(DPLL_CTRL_1.TS0_HRS=1)

DT_Sx[pt–pq+1] = DPLL_DT_S[p].DT_S[pt–pq+1]/8

Compare with equations *[eq_103]* , *[eq_109]*

21.8.3.3 For DPLL_NUSC_NUSE – DPLL_NUSC_VSN <= DPLL_NA[i] (part w) and DPLL_NUSC_NUSE > 1

[eq_107]
IF(DPLL_NUSC_NUSE–DPLL_NUTC.VTN=pq and pq <= pm and pn>1)

PDT_S[n] = (pm+mb) *DT_Sx[pt–pq+1] * RDT_S[pt–pq]

with

[eq_108]
IF(DPLL_CTRL_1.TS0_HRS=0)

DT_Sx[pt–pq+1] = DPLL_DT_S[p].DT_S[pt–pq+1]

Compare with equations *[eq_102]* , *[eq_105]* or

[eq_109]
IF(DPLL_CTRL_1.TS0_HRS=1)

DT_Sx[pt–pq+1] = DPLL_DT_S[p].DT_S[pt–pq+1]/8

Compare with equations *[eq_103]* , *[eq_109]*

21.8.3.4 For DPLL_NUSC_NUSE = 1

for pn=1

[eq_110]
IF(pn=1)

PDT_S[n] = (pm+mb) * DT_S_ax * RDT_S[pt–1]

with

[eq_111]
IF(DPLL_CTRL_1.TS0_HRS=0)

DT_S_ax = DPLL_DT_S_ACT.DT_S_ACT

Compare with equations *[eq_124]* , *[eq_145]* or

[eq_112]
IF(DPLL_CTRL_1.TS0_HRS=1)

DT_S_ax = DPLL_DT_S_ACT.DT_S_ACT/8

Compare with equations *[eq_125]* , *[eq_149]*

21.8.3.5 Calculate the duration value for an Action

[eq_113]

DTA[n] = (DT_S_ACT + MEDT_S)* PDT_S[n]

Compare with equation *[eq_126]*

Use the results of equations *[eq_31]* , *21.7.3.6 "Calculate the weighted average error"* or *21.7.5.3 "Calculate the weighted average error"* and one of the equations *[eq_101]* , *[eq_104]* , *[eq_107]* , *[eq_110]* for the above calculation.

21.8.4 Action calculations for STATE signal backwards

Valid for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 with p=DPLL_APS_1C2, pt=DPLL_APS_APS, pm= **DPLL_NA[i]** (part w) mb= **DPLL_NA[i]** (part b)/1024, DPLL_NUSC_NUSE – DPLL_NUSC_VSN = pq and DPLL_NUSC_NUSE=pn

For **DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 calculate for *DIR1* =1 all NOAC Actions in backwards mode for special purposes; in the case **DPLL_CTRL_1.SMC** =1 and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 calculate up to NOAC/2 Actions NOAC/2 to NOAC−1 in dependence of the *STATE* input.

Note:

All 5 steps of equations *21.8.4.1 "For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w)"* to *21.8.4.5 "Calculate the duration value until Action"* are only calculated in emergency mode or for **DPLL_CTRL_1.SMC** =1 in combination with **DPLL_CTRL_0_SHADOW_STATE.RMO** =1.

21.8.4.1 For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w)

[eq_114]
IF((DIR2= 1 and DPLL_CTRL_1.SMC=1) or (DIR1=1 and DPLL_CTRL_1.SMC=0)) and pq>pm)

PDT_S[n] = (TSF_S[p−pm+pn] − TSF_S[p+pn] + mb* DT_Sx[pt+pq−1]) * RDT_S[pt+pq]

with

[eq_115]
IF(DPLL_CTRL_1.TS0_HRS=0)

DT_Sx[pt+pq−1]= DPLL_DT_S[p].DT_S[pt+pq−1]

(Compare with equations *[eq_118]* , *[eq_121]*) or

[eq_116]
IF(DPLL_CTRL_1.TS0_HRS=1)

DT_Sx[pt+pq−1] = DPLL_DT_S[p].DT_S[pt+pq−1]/8

(Compare with equations *[eq_119]* , *[eq_122]*)

21.8.4.2 For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w) and DPLL_NUSC_NUSE= 2*(DPLL_SNU+1)

[eq_117]
IF(DPLL_STATUS.SYS=1 and DPLL_NUSC_NUSE=2*(DPLL_SNU+1) and pq>pm and DPLL_CTRL_1.SYSF=0 and DPLL_NUSC_VSN=2*DPLL_SYN_NS)

PDT_S[n] = (TSF_S[p−pm] − TSF_S[p]+ mb* DT_Sx[pt+pq−1]) * RDT_S[pt]

Note:

In this case DPLL_NUSC_NUSE – DPLL_NUSC_VSN= 2*(DPLL_SNU+1−DPLL_SYN_NS)

with

[eq_118]
IF(DPLL_CTRL_1.TS0_HRS=0)

DT_Sx[pt+pq−1]= DPLL_DT_S[p].DT_S[pt+pq−1]

(Compare with equations *[eq_115]* , *[eq_121]*) or

[eq_119]
IF(DPLL_CTRL_1.TS0_HRS=1)

DT_Sx[pt+pq−1] = DPLL_DT_S[p].DT_S[pt+pq−1]/8

(Compare with equations *[eq_116]* , *[eq_122]*)

21.8.4.3 For DPLL_NUSC_NUSE – DPLL_NUSC_VSN <= DPLL_NA[i] (part w) and DPLL_NUSC_NUSE > 1

[eq_120]
IF(DPLL_NUSC_NUSE−DPLL_NUSC_VSN=pq and pq <= pm and DPLL_NUSC_NUSE=pn and pn>1)

PDT_S[n] = m*DT_Sx[pt+pq-1] * RDT_S[pt+pq]

with

[eq_121]
IF(DPLL_CTRL_1.TS0_HRS=0)
DT_Sx[pt+pq-1] = DPLL_DT_S[p].DT_S[pt+pq-1]

(Compare with equations *[eq_115]* , *[eq_118]*) or

[eq_122]
IF(DPLL_CTRL_1.TS0_HRS=1)
DT_Sx[pt+pq-1] = DPLL_DT_S[p].DT_S[pt+pq-1]/8

(Compare with equations *[eq_116]* , *[eq_119]*)

21.8.4.4 For DPLL_NUSC_NUSE = 1

[eq_123]
IF(pn=1)
PDT_S[n] = (pm+mb) * DT_S_ax * RDT_S[pt+1]

with

[eq_124]
IF(DPLL_CTRL_1.TS0_HRS=0)
DT_S_ax = DPLL_DT_S_ACT.DT_S_ACT

Compare with equations *[eq_111]* , *[eq_145]* or

[eq_125]
IF(DPLL_CTRL_1.TS0_HRS=1)
DT_S_ax = DPLL_DT_S_ACT.DT_S_ACT/8

Compare with equations *[eq_112]* , *[eq_149]*

21.8.4.5 Calculate the duration value until Action

[eq_126]
DTA[n] = (DT_S_ACT + MEDT_S)* PDT_S[n]

Compare with equation *[eq_113]*

Use the results of equations *[eq_31]* , *21.7.3.6 "Calculate the weighted average error"* or *21.7.5.3 "Calculate the weighted average error"* and one of the equations *[eq_114]* , *[eq_117]* , *[eq_120]* , *[eq_123]* for the above calculation.

21.8.5 Update of RAM in Normal and Emergency Mode

After considering the calculations for up to all NOAC Actions according to the equations (*21.8.1.1 "For DPLL_NUTC.NUTE - DPLL_NUTC.VTN > NA[n] (part w)"* and similar equation *[eq_87]*), only when going back to state 1 or 21 (because of a new *TRIGGER* or *STATE* event, that means when no further **DPLL_ID_PMTR_[n].ID_PMTR** values are to be considered) set time stamp values and duration of increments in the RAM.

21.8.5.1 Update the time stamp values for TRIGGER signal

[eq_127]
TSF_T[ps]=TS_Tx

using the following equations for the determination of TS_Tx

[eq_128]
IF(DPLL_CTRL_1.TS0_HRT=0)
TS_Tx=TS_T

[eq_129]
DT_T_ax = DT_T_ACT

Compare with equations *[eq_85]* , *[eq_98]*

[eq_130]
IF(DPLL_CTRL_1.TS0_HRT=1)
TS_Tx[20:0]=TS_T/8

[eq_131]
IF(DPLL_TBU_TS0_T.DPLL_TBU_TS0_T[20:0] > or = TS_Tx[20:0])
TS_Tx[23:21] = DPLL_TBU_TS0_T.DPLL_TBU_TS0_T[23:21]

[eq_132]
IF(DPLL_TBU_TS0_T.DPLL_TBU_TS0_T[20:0] < TS_Tx[20:0])
TS_Tx[23:21]=DPLL_TBU_TS0_T.DPLL_TBU_TS0_T[23:21] −1
[eq_133]
DT_T_ax = DT_T_ACT/8

Compare with equations *[eq_86]* , *[eq_99]*

Note:
The combination of values *LOW_RES* =0 and **DPLL_CTRL_1.TS0_HRT** =1 is not possible.

Store the time stamp values in the time stamp field according to the address pointer **DPLL_APT.APT_2B** =ps, but make this update only after the calculation of Actions (see chapter *21.8 "Calculations for Actions"*) because the old **DPLL_TSF_T[p].TSF_T** values are still needed for these calculations. Please consider that the address pointer after a gap is still incremented by **DPLL_NUTC.SYN_T_OLD** in that case (see state machine step 1 in chapter *21.9.6 "Scheduling of the Calculation"*).

21.8.5.2 Extend the time stamp values for TRIGGER signal in forward direction

[eq_134]
IF(DPLL_STATUS.SYT=1 and DPLL_NUTC.SYN_T_OLD=pr>1 and DIR1=0)
TSF_T[ps−1] = TSF_T[ps] − DT_T_ax

Compare with equation *[eq_137]*
[eq_135]
IF(DPLL_STATUS.SYT=1 and DPLL_NUTC.SYN_T_OLD=pr>1 and DIR1=0)
TSF_T[ps−2] = TSF_T[ps−1] − DT_T_ax

Compare with equation *[eq_138]* until
[eq_136]
IF(DPLL_STATUS.SYT=1 and DPLL_NUTC.SYN_T_OLD=pr>1 and DIR1=0)
TSF_T[ps− pr+1] = TSF_T[ps− pr+2] − DT_T_ax

Compare with equation *[eq_139]* after the incrementation of the pointer **DPLL_APT.APT_2B** by **DPLL_NUTC.SYN_T_OLD**

21.8.5.3 Extend the time stamp values for trigger signal in backward direction

[eq_137]
IF(DPLL_STATUS.SYT=1 and DPLL_NUTC.SYN_T_OLD=pr>1 and DIR1=1)
TSF_T[ps+1] = TSF_T[ps] −DT_T_ax

Compare with equation *[eq_134]*
[eq_138]
IF(DPLL_STATUS.SYT=1 and DPLL_NUTC.SYN_T_OLD=pr>1 and DIR1=1)
TSF_T[ps+2] = TSF_T[ps+1] −DT_T_ax

Compare with equation *[eq_135]* until
[eq_139]
IF(DPLL_STATUS.SYT=1 and DPLL_NUTC.SYN_T_OLD=pr>1 and DIR1=1)
TSF_T[ps+pr−1] = TSF_T[ps+pr−2] −DT_T_ax

Compare with equation *[eq_136]* after the decrementation of the pointer **DPLL_APT.APT_2B** by **DPLL_NUTC.SYN_T_OLD**

21.8.5.4 Update of RAM by DT_T[p] and RDT_T[p] after calculation

[eq_140]
DT_T[p] = DT_T_ACT

save old reciprocal value from RAM before overwriting:
[eq_141]
RDT_T_FS1= RDT_T[p]

after that store new value in RAM
[eq_142]
RDT_T[p]= RDT_T_ACT

Store the increment duration and the reciprocal value in RAM region 2 after calculation of actions only when a new active *TRIGGER* slope is detected in normal mode and in emergency mode directly after the calculation of **DPLL_DT_T_ACT.DT_T_ACT** or **DPLL_RDT_T_ACT.RDT_T_-ACT** respectively.

21.8.5.5 Update the time stamp values for STATE signal

[eq_143]
TSF_S[ps]=TS_Sx

using the following equations for the determination of TS_Sx

[eq_144]
IF(DPLL_CTRL_1.TS0_HRS=0)
TS_Sx=TS_S

[eq_145]
DT_S_ax = DT_S_ACT

Compare with equations *[eq_111]* , *[eq_124]*

[eq_146]
IF(DPLL_CTRL_1.TS0_HRS=1)
TS_Sx[20:0]=TS_S/8

[eq_147]
IF(DPLL_TBU_TS0_S.DPLL_TBU_TS0_S[20:0] > or = TS_Sx[20:0])
TS_Sx[23:21]=DPLL_TBU_TS0_S.DPLL_TBU_TS0_S[23:21]

[eq_148]
IF(DPLL_TBU_TS0_S.DPLL_TBU_TS0_S[20:0] < TS_Sx[20:0])
TS_Sx[23:21]=DPLL_TBU_TS0_S.DPLL_TBU_TS0_S[23:21] −1

[eq_149]
DT_S_ax = DT_S_ACT/8

Compare with equations *[eq_112]* , *[eq_125]*

Note:
the combination of values *LOW_RES* =0 and **DPLL_CTRL_1.TS0_HRS** =1 is not possible.

Store the time stamp value in the time stamp field according to the address pointer DPLL_APS_1C2=s, but make this update only after the calculation of Actions (equations *[eq_104]* , chapter *21.8.3.2 "For DPLL_NUSC_NUSE − DPLL_NUSC_VSN > DPLL_NA[i] (part w) and DPLL_NUSC_NUSE= 2*(DPLL_SNU+1)"* or equation *[eq_117]* in chapter *21.8.4.2 "For DPLL_NUSC_NUSE − DPLL_NUSC_VSN > DPLL_NA[i] (part w) and DPLL_NUSC_NUSE= 2*(DPLL_SNU+1)"* , if applicable) because the old **DPLL_TSF_S[p].TSF_S** values are still needed for these calculations. Please consider, that the address pointer after a gap is still incremented by DPLL_NUSC_SYN_S_OLD in that case (see state machine step 21 in chapter *21.9.6 "Scheduling of the Calculation"*).

21.8.5.6 Extend the time stamp values for STATE signal

Let pr = DPLL_NUSC_SYN_S_OLD

[eq_150]
IF(DPLL_STATUS.SYS=1 and pr>1 and (DIR2=0 or DIR1=0))
TSF_S[ps−1] = TSF_S[ps] − DT_S_ax

(Compare with equation *[eq_153]*)

[eq_151]
IF(DPLL_STATUS.SYS=1 and pr>1 and (DIR2=0 or DIR1=0))
TSF_S[ps−2] = TSF_S[ps−1] − DT_S_ax

(Compare with equation *[eq_154]*) until

[eq_152]
IF(DPLL_STATUS.SYS=1 and pr>1 and (DIR2=0 or DIR1=0))
TSF_S[ps−pr+1] = TSF_S[ps−pr+2] − DT_S_ax

(Compare with equation *[eq_155]*)

after incrementation of the pointer APS_2b by DPLL_NUSC_SYN_S_OLD

21.8.5.7 Extend the time stamp values for STATE signal for backward direction

Let pr = DPLL_NUSC_SYN_S_OLD

[eq_153]
IF(DPLL_STATUS.SYS=1 and pr>1 and (DIR2=1 or DIR1=1))
TSF_S[ps+1] = TSF_S[ps] – DT_S_ax

(Compare with equation *[eq_150]*)

[eq_154]
IF(DPLL_STATUS.SYS=1 and pr>1 and (DIR2=1 or DIR1=1))
TSF_S[ps+2] = TSF_S[ps+1] – DT_S_ax

(Compare with equation *[eq_151]*) until

[eq_155]
IF(DPLL_STATUS.SYS=1 and pr>1 and (DIR2=1 or DIR1=1))
TSF_S[ps+pr–1] = TSF_S[ps+pr–2] – DT_S_ax

(Compare with equation *[eq_152]*)

after the incrementation of the pointer DPLL_APS_1C2 by DPLL_NUSC_SYN_S_OLD

21.8.5.8 Update of RAM by DT_S[p] and RDT_S[p] after calculation

[eq_156]
DT_S[p] = DT_S_ACT

(Compare with equations *[eq_31]* or *[eq_32]* for **DPLL_CTRL_11.ADS** =1)

save old reciprocal value from RAM before overwriting:

[eq_157]
RDT_S_FS1= DPLL_RDT_S[p].RDT_S

after that store new value in RAM

[eq_158]
DPLL_RDT_S[p].RDT_S = RDT_S_ACT

A new active *STATE* slope is detected in emergency mode or in normal mode directly after the calculation of the values (**DPLL_CTRL_1.SMC** =0, **DPLL_CTRL_0_SHADOW_STATE.RMO** =0).

Store the increment duration and the reciprocal value in RAM region 1c after calculation of actions only when a new active *STATE* slope is detected in emergency mode and in normal mode directly after calculation of **DPLL_DT_S_ACT.DT_S_ACT** or **DPLL_RDT_S_ACT.RDT_S_ACT** respectively.

21.8.6 Time and position stamps for Actions in Normal Mode

21.8.6.1 Calculate the Action time stamp

[eq_159]
IF(DPLL_DTA[n].DTA > DPLL_DLA[n] and DPLL_DTA[n].DTA – DPLL_DLA[n] < 0x800000)
DPLL_TSAC[n].TSAC= DPLL_DTA[i].DTA – DPLL_DLA[n] + TS_Tx

[eq_160]
IF(DTA[n] < DPLL_DLA[n])
TSAC[n]= TS_Tx

[eq_161]
IF(DTA[n] > DPLL_DLA[n] and DTA[n] – DPLL_DLA[n] > 0x7FFFFF)
TSAC[n]= 0x7FFFFF + TS_Tx

For TS_Tx see equations *[eq_127]* and following, chapter *21.8.5.1 "Update the time stamp values for TRIGGER signal "* .

The calculation is done after the calculation of the current expected duration value according to equation *[eq_100]* at chapter *21.8.2.5 "Calculate the duration value for an Action"* . The time stamp of the Action can be calculated as shown above in equations of this chapter using the delay value of the Action and the current time stamp.

21.8.6.2 Calculate the position stamp forwards

[eq_162]
IF(DIR1=0 and DPLL_CTRL_1.TS0_HRT=0)
PSAC[n] = PSA[n] – (DPLL_DLA[n]*RCDT_TX_NOM)*(MLT+1)

with

[eq_163]
RCDT_TX_NOM= (1/CDT_TX_NOM) * SYN_T

(Compare with equations *[eq_169]* , *[eq_166]* and *[eq_172]*) and

[eq_164]

RCDT_TX= 1/CDT_TX

(Compare with equations *[eq_167]* , *[eq_170]* , *[eq_173]*)

[eq_165]

IF(DIR1=0 and DPLL_CTRL_1.TS0_HRT=1)

PSAC[n] = PSA[n] – (8 * DPLL_DLA[n] * RCDT_TX_NOM) * (MLT+1)

with

[eq_166]

RCDT_TX_NOM= (1/CDT_TX_NOM) * SYN_T

(Compare with equations *[eq_166]* , *[eq_169]* and *[eq_172]*) and

[eq_167]

RCDT_TX= 1/CDT_TX

(Compare with equations *[eq_164]* , *[eq_170]* , *[eq_173]*)

replace (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) in equations *[eq_162]* and *[eq_165]* by **DPLL_MLS1.MLS1** for **DPLL_CTRL_1.SM-C** =1

Use the calculated value of equation *[eq_164]* also for the generation of *SUB_INC1* and *SUB_INC2* and serve the Action by transmission of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** to *ACT_D[n]* .
The action is to be updated for each new *TRIGGER* event until the calculated time stamp is in the past.
In this case the values of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** depend on the **DPLL_CTRL_11.ACBU** signal.
When **DPLL_CTRL_11.ACBU** = '0': Use the time stamp of the last input event instead of the calculated value and the calculated position stamp of the actual increment as target position value.
When **DPLL_CTRL_11.ACBU** = '1':
For **DPLL_ACB_[n].ACB_[n]** [1:1]= '1': is used as input signal to control if "Action in past" shall be checked based on position information. If the position has reached "past" use the calculated position stamp of the actual increment as target position value.
For **DPLL_ACB_[n].ACB_[n]** [1:1]= '0': In this case the **DPLL_PSAC[n].PSAC** is
used as calculated by the DPLL.
For **DPLL_ACB_[n].ACB_[n]** [0:0]= '1': is used as input signal to control if "Action in past" shall be checked based on time information. If the time has reached "past" use the time stamp of the last input event instead of the calculated **DPLL_TSAC[n].TSAC** value.
For **DPLL_ACB_[n].ACB_[n]** [0:0]= '0': In this case the **DPLL_TSAC[n].TSAC** is used as calculated by the DPLL.
Set the corresponding shadow bit in the **DPLL_ACT_STA** register. Because of the blocking of the read operation, the *ACT_D* values can be read only once.

21.8.6.3 Calculate the position stamp backwards

[eq_168]

IF(DIR1=1 and DPLL_CTRL_1.TS0_HRT=0)

PSAC[n] = PSA[n] + (DPLL_DLA[n] * RCDT_TX_NOM) * (MLT+1)

with

[eq_169]

RCDT_TX_NOM= (1/CDT_TX_NOM) * SYN_T

(Compare with equations *[eq_163]* and *[eq_166]* , *[eq_172]*) and

[eq_170]

RCDT_TX= 1/CDT_TX

(Compare with equations *[eq_164]* , *[eq_167]* , *[eq_173]*)

[eq_171]

IF(DIR1=1 and DPLL_CTRL_1.TS0_HRT=1)

PSAC[n] = PSA[n] + (8 * DPLL_DLA[n] * RCDT_TX_NOM) * (MLT+1)

with

[eq_172]

RCDT_TX_NOM= (1/CDT_TX_NOM) * SYN_T

(Compare with equations *[eq_163]* , *[eq_166]* and *[eq_169]*) and

[eq_173]

RCDT_TX= 1/CDT_TX

(Compare with equations *[eq_164]* , *[eq_167]* , *[eq_170]*)

replace (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) in equations *[eq_168]* and *[eq_171]* by **DPLL_MLS1.MLS1** for **DPLL_CTRL_1.SM-C** =1

Use the calculated value of equation *[eq_170]* also for the generation of *SUB_INC1* and *SUB_INC2* and serve the Action by transmission of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** to *ACT_D[n]* .
The action is to be updated for each new *TRIGGER* event until the calculated time stamp is in the past.
In this case the values of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** depend on the **DPLL_CTRL_11.ACBU** signal.
When **DPLL_CTRL_11.ACBU** = '0': Use the time stamp of the last input event instead of the calculated value and the calculated position stamp of the actual increment as target position value.
When **DPLL_CTRL_11.ACBU** = '1':
For **DPLL_ACB_[n].ACB_[n]** [1:1]= '1': is used as input signal to control if "Action in past" shall be checked based on position information. If the position has reached "past" use the calculated position stamp of the actual increment as target position value.
For **DPLL_ACB_[n].ACB_[n]** [1:1]= '0': In this case the **DPLL_PSAC[n].PSAC** is used as calculated by the DPLL.
For **DPLL_ACB_[n].ACB_[n]** [0:0]= '1': is used as input signal to control if "Action in past" shall be checked based on time information. If the time has reached "past" use the time stamp of the last input event instead of the calculated **DPLL_TSAC[n].TSAC** value.
For **DPLL_ACB_[n].ACB_[n]** [0:0]= '0': In this case the **DPLL_TSAC[n].TSAC** is used as calculated by the DPLL.
Set the corresponding shadow bit in the **DPLL_ACT_STA** register. Because of the blocking of the read operation the *ACT_D* values can be read only once.

21.8.7 The use of the RAM

The RAM is used to store the data of the last FULL_SCALE period. The use of single port RAMs is recommended. The data width of the RAM is usually 3 bytes, but could be extended to 4 bytes in future applications. There are 3 different RAMs, each with separate access ports. The RAM 1a is used to store the position minus time requests, got from the ARU. No CPU access is possible to this RAM during operation (when the DPLL is enabled).
RAM 1b is used for configuration parameters and variables needed for calculations. Within RAM 1c the values of the *STATE* events are stored. RAM 1b and RAM 1c have a common access port and are also marked as RAM 1bc in order to clarify this fact.
RAM 2 is used for values of the *TRIGGER* events.
Because of the access of the DPLL internal state machine on one side and the CPU on the other side, the access priority has to be controlled for both RAMs 1bc and 2. The access priority is defined as stated below. The CPU access procedure via AE–interface goes in a wait state (waiting for data valid) while it needs a colliding RAM access while serving a corresponding state machine RAM access. In order not to provoke unexpected behavior of the algorithms the writing of the CPU to the RAM regions 1b, 1c or 2 will be monitored and results in interrupt requests when enabled.

CPU access is specified as follows:
1. CPU has highest priority for a single read/write access. The DPLL algorithm is stalled during external bus RAM accesses.
2. After serving the CPU access to the RAM the DPLL gets the highest RAM access priority for 8 clock cycles. Afterwards continue with 1.

The RAM address space has to be implemented in the address space of the CPU.

21.8.8 Time and position stamps for Actions in Emergency Mode

21.8.8.1 Calculate the Action time stamp

[eq_174]
IF(DTA[n] > DPLL_DLA[n] and DTA[n] – DPLL_DLA[n] < 0x800000)
TSAC[n]= DTA[n] – DPLL_DLA[n] + TS_Sx
[eq_175]
IF(DTA[n] < DPLL_DLA[n])
TSAC[n]= TS_Sx
[eq_176]
IF(DTA[n] > DPLL_DLA[n] and DTA[n] – DPLL_DLA[n] > 0x7FFFFF)
TSAC[n]= 0x7FFFFF + TS_Sx

For TS_Sx see equations (*[eq_143]* and following), chapter *21.8.5.5 "Update the time stamp values for STATE signal"* .

The calculation is done after the calculation of the current expected duration value according to equation *[eq_113]* at chapter *21.8.3.5 "Calculate the duration value for an Action"* . The time stamp of the Action can be calculated as shown in this chapter *21.8.8.1 "Calculate the Action time stamp"* using the delay value of the Action and the current time stamp.

21.8.8.2 Calculate the position stamp forwards

[eq_177]
IF(*DIR2* =0 or *DIR1* =0 respectively and DPLL_CTRL_1.TS0_HRS=0)
DPLL_PSAC[n].PSAC = DPLL_PSA[n].PSA – (DPLL_DLA[n].DLA*DPLL_RCDT_SX_NOM.RCDT_SX_NOM)*DPLL_MLS1.MLS1

with
[eq_178]
RCDT_SX_NOM= (1/CDT_SX_NOM) * SYN_S

(Compare with equations *[eq_181]* , *[eq_184]* , *[eq_187]*) and
[eq_179]
RCDT_SX= 1/CDT_SX

(Compare with equations *[eq_182]* , *[eq_185]* and *[eq_188]*)

[eq_180]

IF(DIR2=0 or DIR1=0 respectively and DPLL_CTRL_1.TS0_HRS=1)

PSAC[n] = PSA[n] – (8 * DPLL_DLA[n] * RCDT_SX_NOM) * MLS1

with

[eq_181]

RCDT_SX_NOM= (1/CDT_SX_NOM) * SYN_S

(Compare with equations *[eq_178]* , *[eq_184]* , *[eq_187]*) and

[eq_182]

RCDT_SX= 1/CDT_SX

(Compare with equations *[eq_179]* , *[eq_185]* and *[eq_188]*)

replace **DPLL_MLS1.MLS1** in equations *[eq_177]* and *[eq_180]* by **DPLL_MLS2.MLS2** for (**DPLL_CTRL_1.SMC** =1 and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1)

Use the calculated value of equation *[eq_179]* also for the generation of *SUB_INC1* and *SUB_INC2* and serve the Action by transmission of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** to *ACT_D[n]* .
The Action is to be updated for each new *STATE* event until the calculated time stamp is in the past.
In this case the values of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** depend on the **DPLL_CTRL_11.ACBU** signal.
When **DPLL_CTRL_11.ACBU** = '0': Use the time stamp of the last input event instead of the calculated value and the calculated position stamp of the actual increment as target position value.
When **DPLL_CTRL_11.ACBU** = '1':
For **DPLL_ACB_[n].ACB_[n]** [1:1]= '1': is used as input signal to control if "Action in past" shall be checked based on position information. If the position has reached "past", use the calculated position stamp of the actual increment as target position value.
For **DPLL_ACB_[n].ACB_[n]** [1:1]= '0': In this case the **DPLL_PSAC[n].PSAC** is used as calculated by the DPLL.
For **DPLL_ACB_[n].ACB_[n]** [0:0]= '1': is used as input signal to control if "Action in past" shall be checked based on time information. If the time has reached "past" use the time stamp of the last input event instead of the calculated **DPLL_TSAC[n].TSAC** value.
For **DPLL_ACB_[n].ACB_[n]** [0:0]= '0': In this case the **DPLL_TSAC[n].TSAC** is used as calculated by the DPLL.
Set the corresponding shadow bit in the **DPLL_ACT_STA** register. Because of the blocking of the read operation, the *ACT_D* values can be read only once.

21.8.8.3 Calculate the position stamp backwards

[eq_183]

IF(DIR2=1 or DIR1=1 respectively and DPLL_CTRL_1.TS0_HRS=0)

PSAC[n] = PSA[n] + (DPLL_DLA[n] * RCDT_SX_NOM) * MLS1

with

[eq_184]

RCDT_SX_NOM= (1/CDT_SX_NOM) * SYN_S

(Compare with equations *[eq_178]* , *[eq_181]* , *[eq_187]*) and

[eq_185]

RCDT_SX= 1/CDT_SX

(Compare with equations *[eq_179]* , *[eq_182]* and *[eq_188]*)

For *DIR2* =1 or *DIR1* =1 respectively and **DPLL_CTRL_1.TS0_HRS** =1:

[eq_186]

IF(DIR2=1 or DIR1=1 respectively and DPLL_CTRL_1.TS0_HRS=1)

PSAC[n] = PSA[n] + (8 * DPLL_DLA[n] * RCDT_SX_NOM) * MLS1

with

[eq_187]

RCDT_SX_NOM= (1/CDT_SX_NOM) * SYN_S

(Compare with equations *[eq_178]* , *[eq_181]* , *[eq_184]*) and

[eq_188]

RCDT_SX= 1/CDT_SX

(Compare with equations *[eq_179]* , *[eq_182]* and *[eq_185]*)

replace **DPLL_MLS1.MLS1** in equations *[eq_183]* and *[eq_186]* by **DPLL_MLS2.MLS2** for (**DPLL_CTRL_1.SMC** =1 and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1)

Use the calculated value of equation *[eq_179]* also for the generation of *SUB_INC1* and *SUB_INC2* and serve the Action by transmission of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** to *ACT_D* .
The Action is to be updated for each new *STATE* event until the event is in the past. In this case use the time stamp of the last input event

instead of the calculated value and the calculated position stamp of the actual increment as target position value. Set the corresponding shadow bit in the **DPLL_ACT_STA** register. Because of the blocking read operation the *ACT_D* values can be read only once.

Use the calculated value of equation *[eq_179]* also for the generation of *SUB_INC1* and *SUB_INC2* and serve the Action by transmission of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** to *ACT_D[n]* .
The Action is to be updated for each new *STATE* event until the calculated time stamp is in the past.
In this case the values of **DPLL_TSAC[n].TSAC** and **DPLL_PSAC[n].PSAC** depend on the **DPLL_CTRL_11.ACBU** signal.
When **DPLL_CTRL_11.ACBU** = '0': Use the time stamp of the last input event instead of the calculated value and the calculated position stamp of the actual increment as target position value.
When **DPLL_CTRL_11.ACBU** = '1':
For **DPLL_ACB_[n].ACB_[n]** [1:1]= '1': is used as input signal to control if "Action in past" shall be checked based on position information. If the position has reached "past", use the calculated position stamp of the actual increment as target position value.
For **DPLL_ACB_[n].ACB_[n]** [1:1]= '0': In this case the **DPLL_PSAC[n].PSAC** is used as calculated by the DPLL.
For **DPLL_ACB_[n].ACB_[n]** [0:0]= '1': is used as input signal to control if "Action in past" shall be checked based on time information. If the time has reached "past" use the time stamp of the last input event instead of the calculated **DPLL_TSAC[n].TSAC** value.
For **DPLL_ACB_[n].ACB_[n]** [0:0]= '0': In this case the **DPLL_TSAC[n].TSAC** is used as calculated by the DPLL.
Set the corresponding shadow bit in the **DPLL_ACT_STA** register. Because of the blocking of the read operation, the *ACT_D* values can be read only once.

21.9 Signal processing

21.9.1 Time stamp processing

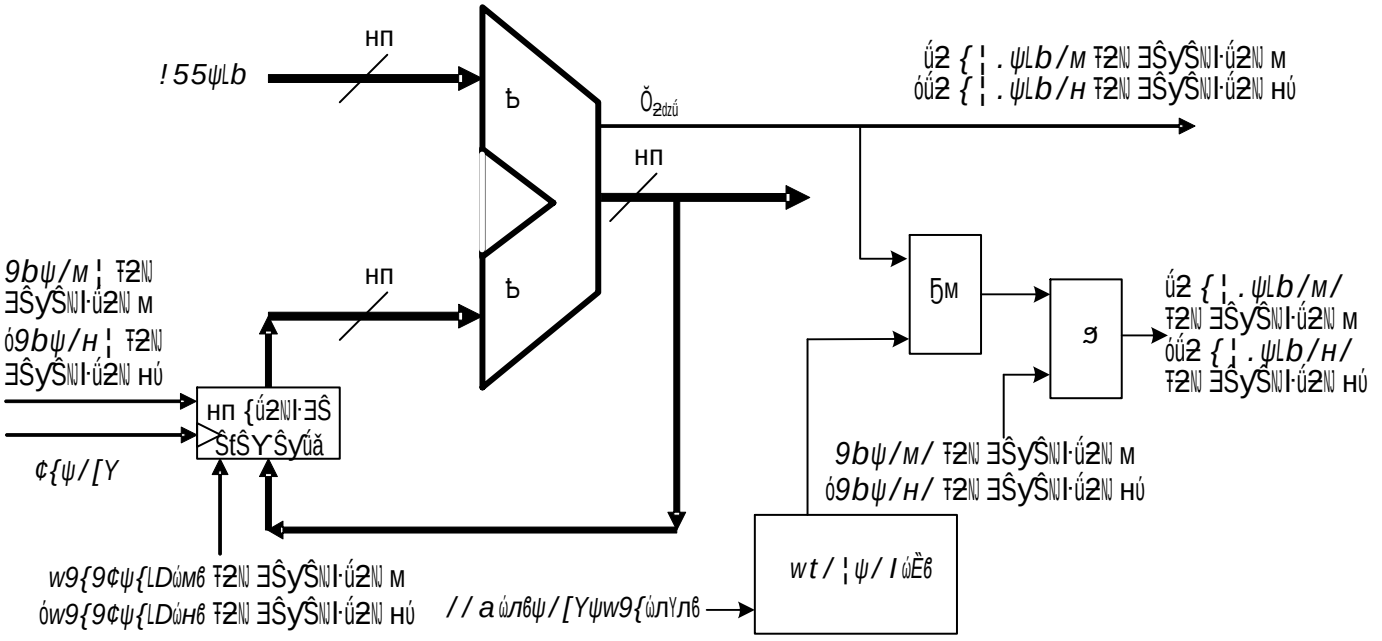
Signal processing means the computation of the time stamps in order to calculate at which time the outputs have to appear. For such purposes the time stamp values have to be stored in the RAM and by simply calculating the difference between old and new values the duration of the last time interval is determined. This difference should also be stored in the RAM in order to see the changes between the intervals by changing the conditions and the speed of the observed process.

21.9.2 Count and compare unit

The count and compare unit processes all input signals by taking the configuration values into account. It uses a state machine and provides the output signals as described above.

21.9.3 Sub pulse generation for DPLL_CTRL_1.SMC=0

Figure 154 Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out.



The *SUB_INC1* and *SUB_INC2* generation by the circuit above has the advantage, that the resolution for higher speed values is better as for a simple down counter. *RESET_SIG[1]* and *RESET_SIG[2]* reset the storage lements to zero on the arrival of an active input slope for generator 1 and generator 2, respectively. After reset and after *EN_C1U* / *EN_C2U* the storage elements should have a zero value. *EN_C1U* / *EN_C2U* has to be zero until reliable *ADD_IN* values are available and the pulse generation starts. This is controlled by the configuration bits **DPLL_CTRL_1.SGE1** and **DPLL_CTRL_1.SGE2** . The calculated values for the increment prediction using equations *[eq_21]* in chapter *21.7.2.7 "Calculate the current increment value"* , equation *[eq_58]* in chapter *21.7.4.4 "Calculate the current increment value"* , equation *[eq_45]* in chapter *21.7.3.7 "Calculate the current increment (nominal value)"* or equation *[eq_71]* in chapter *21.7.5.4 "Calculate the current increment value"* respectively are valid only when at least **DPLL_NUTC.NUTE** >1 *TRIGGER* values or at least DPLL_NUSC_NUSE>1 *STATE* values are available. For **DPLL_NUTC.NUTE** =1 or DPLL_NUSC_NUSE=1 respectively the equations *[eq_193]* in chapter *21.9.3.3*

"Calculate ADD_IN in normal mode for DPLL_CTRL_1.SMC=0" and equation *[eq_196]* in chapter *21.9.3.5 "Calculate ADD_IN in emergency mode for DPLL_CTRL_1.SMC=0"* use the actual increment value subtracted by the weighted average error.

The generation of *SUB_INC1* pulses depends on the configuration of the DPLL.
In automatic end mode the counter **DPLL_INC_CNT1.INC_CNT1** resets the enable signal *EN_C1C* when the desired number of pulses is reached. In this case only the uncompensated output *SUB_INC1* remains active in order to provide pulses for the input filter unit. In the case of acceleration missing pulses can be determined at the next *TRIGGER* / *STATE* event in normal/emergency mode easily. For the correction strategy **DPLL_CTRL_1.COA** = 0 those missing pulses are sent out with *CCM[0]_CLK_RES* [0:0] frequency as soon they are determined. The new pulses are sent out afterwards, when **DPLL_INC_CNT1.INC_CNT1** is set to the desired value, maybe by adding **DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1 or **DPLL_MLS1.MLS1** respectively for the new *TRIGGER* / *STATE* event.

Because the used DIV procedure of the algorithms results only in integer values, a systematic failure could appear. The pulse generation at *SUB_INC1* will stop in automatic end mode when the **DPLL_INC_CNT1.INC_CNT1** register reaches zero or all remaining pulses at a new increment will be considered in the next calculation. In this way the loss of pulses can be avoided.
When a new *TRIGGER* / *STATE* appears the value of **DPLL_NUTC.SYN_T** *(**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) or DPLL_NUSC_SYN_S* **DPLL_MLS1.MLS1** respectively is added to **DPLL_INC_CNT1.INC_CNT1** , when **DPLL_CTRL_1.SGE1** =1. Therefore for FULL_SCALE 2*(**DPLL_CTRL_0.TNU** +1)*(**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) pulses *SUB_INC1* is generated, when **DPLL_INC_CNT1.INC_CNT1** reaches the zero value. The generation of *SUB_INC1* pulses has to be done as fast as possible. The calculations for the *ADD_IN* value must be done first. Therefore, all values needed for calculation are to be fetched in a forecast.

21.9.3.1 Calculate the number of pulses to be sent in normal mode using the automatic end mode condition

[eq_189]
IF(DPLL_CTRL_0_SHADOW_TRIGGER.RMO=0, DPLL_CTRL_1.SMC=0 and DPLL_CTRL_1_SHADOW_TRIGGER.DMO=0)
NMB_T = ((MLT+1) + PD_store) *SYN_T + MP + MPVAL1

with

PD_store = **DPLL_ADT_T[p]** [12:0], prefetched during last increment,

SYN_T = ADT_T[18:16], prefetched during last increment,

and **DPLL_MPVAL1.MPVAL1** = pulse correction value for **DPLL_CTRL_1_SHADOW_TRIGGER.PCM1** =1

while the value for PD_store is zero for **DPLL_CTRL_0_SHADOW_TRIGGER.AMT** =0
and the value of **DPLL_MP_T.MP_T** is zero for **DPLL_CTRL_1_SHADOW_TRIGGER.COA** =0.

In order to get a higher resolution for higher speed a generator for the sub-pulses is chosen using an adder. All missing pulses **DPLL_MP_T.MP_T** are considered using equation *[eq_189]* and are determined by counting the number of pulses of the last increment. The value **DPLL_NUTC.SYN_T** is stored from the last increment using **DPLL_ADT_T[p].NT** value at RAM region 2c.

21.9.3.2 Calculate the number of pulses to be sent in emergency mode using the automatic end mode condition for DPLL_CTRL_1.SMC=0

[eq_190]
IF(DPLL_CTRL_0_SHADOW_STATE.RMO=1 and DPLL_CTRL_1.SMC=0 and DPLL_CTRL_1_SHADOW_TRIGGER.DMO=0)
NMB_S = (MLS1 + PD_S_store) *SYN_S+ MP

Note:
The value for PD_S_store is zero for **DPLL_CTRL_0_SHADOW_STATE.AMS** =0

with
[eq_191]
MLS1= (MLT+1) * (TNU+1) / (SNU+1)

and
[eq_192]
PD_S_store = ADT_S[15:0]

prefetched during last increment.

DPLL_NUSC_SYN_S = **DPLL_ADT_S[p]** [21:16], prefetched during last increment

DPLL_MPVAL1.MPVAL1 = pulse correction value for **DPLL_CTRL_1_SHADOW_STATE.PCM1** =1

while the value for PD_S_store is zero for **DPLL_CTRL_0_SHADOW_STATE.AMS** =0
and
the value of **DPLL_MP_S.MP_S** is zero for **DPLL_CTRL_1_SHADOW_STATE.COA** =0

Note:

These calculations above in equations chapters of *21.9.3.1 "Calculate the number of pulses to be sent in normal mode using the automatic end mode condition"* and *21.9.3.2 "Calculate the number of pulses to be sent in emergency mode using the automatic end mode condition for DPLL_CTRL_1.SMC=0"* are only valid for an automatic end mode (**DPLL_CTRL_1_SHADOW_TRIGGER.DMO** = 0).
For calculation of the number of generated pulses a value of 0.5 is added as shown in equations *[eq_193]* , *[eq_194]* , *[eq_196]* or *[eq_197]* respectively in order to compensate rounding down errors at the succeeding arithmetic operations. Because in automatic end mode the number of pulses is limited by **DPLL_INC_CNT1.INC_CNT1** it is guaranteed, that not more pulses as needed are generated and in the same way missing pulses are caught up for the next increment.

21.9.3.3 Calculate ADD_IN in normal mode for DPLL_CTRL_1.SMC=0

[eq_193]
IF(DPLL_CTRL_0_SHADOW_TRIGGER.RMO=0 and LOW_RES=DPLL_CTRL_1.TS0_HRT)
ADD_IN_CALN= (NMB_T+0.5) *RCDT_TX

where **DPLL_RCDT_TX.RCDT_TX** is the 2^{32} time value of the quotient in equation *[eq_164]* .

In normal mode (for **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0) calculate
in the case *LOW_RES* =1 and **DPLL_CTRL_1.TS0_HRT** =0

[eq_194]
IF(DPLL_CTRL_0_SHADOW_TRIGGER.RMO=0 and LOW_RES=1 and DPLL_CTRL_1.TS0_HRT=0)
ADD_IN_CALN= (NMB_T+0.5) *(RCDT_TX /8)

where **DPLL_RCDT_TX.RCDT_TX** is the 2^{32} time value of the quotient in equation *[eq_164]* .

[eq_195]
IF(DPLL_CTRL_0_SHADOW_TRIGGER.RMO=0 and DPLL_CTRL_1.SMC=0)
ADD_IN_CAL1 = ADD_IN_CALN

LOW_RES =0 and **DPLL_CTRL_1.TS0_HRT** =1 is not possible. For such a configuration the **DPLL_STATUS.RCT** bit is set together with the **DPLL_STATUS.ERR** bit.

In the automatic end mode (**DPLL_CTRL_1_SHADOW_TRIGGER.DMO** =0) missing pulses should be sent to the input *RPCU_CH[x]* (rapid pulse catch up on) in chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "* , to be caught up on with *CCM[0]_CLK_RES* [0:0] (for **DPLL_CTRL_1.COA** =0).
When normal and rapid pulses are generated simultaneously, the *SUB_INC1* and *SUB_INC2* frequency is doubled at this moment in order to count two pulses at the **TBU_CH[x]_BASE.BASE** register. In order to make the frequency doubling possible, the *CCM[0]_CLK_RES* [0:0] should be having a frequency which does not exceed half the frequency of *CLS[0]_CLK* . In addition, the *ADD_IN* value should never exceed the value 0x800000. This limitation is only necessary for **DPLL_CTRL_1_SHADOW_TRIGGER.DMO** =0 and **DPLL_CTRL_1.COA** =0.
For the normal mode replace *ADD_IN* of the ADDER (see Figure in chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*) by **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** (when calculated, **DPLL_CTRL_1.DLM1** =0) or **DPLL_ADD_IN_LD1.ADD_IN_LD1** (when provided by the CPU, **DPLL_CTRL_1.DLM1** =1).
The sub-pulse generation in this case is done by the following calculations using a 24 bit adder with a carry out c_{out} and the following inputs:
- *ADD_IN*
- the second input is the output of the adder, stored one time stamp clock before

In order to not complicate the calculation procedure, use a Multiplier with a sufficient bit width at the output and use the corresponding shifted output bits.

21.9.3.4 Enabling of the compensated output for pulses

The c_{out} of the adder directly influences the *SUB_INC1* output of the DPLL (see Figure in chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*). The compensated output *SUB_INC1C* and *SUB_INC2C* is in automatic end mode only enabled by *EN_C1C* / *EN_C2C* when **DPLL_INC_CNT1.INC_CNT1** and **DPLL_INC_CNT2.INC_CNT2** >0.

21.9.3.5 Calculate ADD_IN in emergency mode for DPLL_CTRL_1.SMC=0

[eq_196]
IF(DPLL_CTRL_0_SHADOW_STATE.RMO=1 and LOW_RES=DPLL_CTRL_1.TS0_HRS)
ADD_IN_CALE= (NMB_S+0.5)* RCDT_SX

where **DPLL_RCDT_SX.RCDT_SX** is the 2^{32} time value of the quotient in equation *[eq_179]* in chapter *21.8.8.2 "Calculate the position stamp forwards"* .

[eq_197]
IF(DPLL_CTRL_0_SHADOW_STATE.RMO=1 and LOW_RES=1 and DPLL_CTRL_1.TS0_HRS=0)
ADD_IN_CALE= (NMB_S+0.5)* RCDT_SX /8

where **DPLL_RCDT_SX.RCDT_SX** is the 2^{32} time value of the quotient in equation *[eq_179]* in chapter *21.8.8.2 "Calculate the position stamp forwards"* .

[eq_198]
IF(DPLL_CTRL_0_SHADOW_STATE.RMO=1 and DPLL_CTRL_1.SMC=0)
ADD_IN_CAL1 = ADD_IN_CALE

LOW_RES =0 and **DPLL_CTRL_1.TS0_HRS** =1 is not possible. For such a configuration the **DPLL_STATUS.RCS** bit is set together with the **DPLL_STATUS.ERR** bit.

In the automatic end mode (**DPLL_CTRL_1_SHADOW_TRIGGER.DMO** =0) missing pulses should be sent to the input *RPCU_CH[x]* (rapid pulse catch up on) in chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "* , to be caught up on with *CCM[0]_CLK_RES* [0:0] (for **DPLL_CTRL_1_SHADOW_TRIGGER.COA** =0).
When normal and rapid pulses are generated simultaneously, the *SUB_INC1* and *SUB_INC2* frequency is doubled at this moment in order to count two pulses at the **TBU_CH[x]_BASE.BASE** register. In order to make the frequency doubling possible, the *CCM[0]_CLK_RES* [0:0] should be having a frequency which does not exceed half the frequency of the system clock. In addition, the *ADD_IN* value should never exceed the value 0x800000. This limitation is only necessary for **DPLL_CTRL_1_SHADOW_TRIGGER.DMO** =0 and **DPLL_CTRL_1_SHADOW_TRIGGER.COA** =0.
For the emergency mode replace *ADD_IN* of the Adder (see Figure in chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*) by **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** (when calculated, **DPLL_CTRL_1.DLM1** =0) or **DPLL_ADD_IN_LD1.ADD_IN_LD1** (when provided by the CPU, **DPLL_CTRL_1.DLM1** =1).
The sub-pulse generation in this case is done by the following calculations using a 24 bit adder with a carry out c_{out} and the following inputs:
- *ADD_IN*
- the second input is the output of the adder, stored one time stamp clock before.

In order to not complicate the calculation procedure, use a Multiplier with a sufficient bit width at the output and use the corresponding shifted output bits.

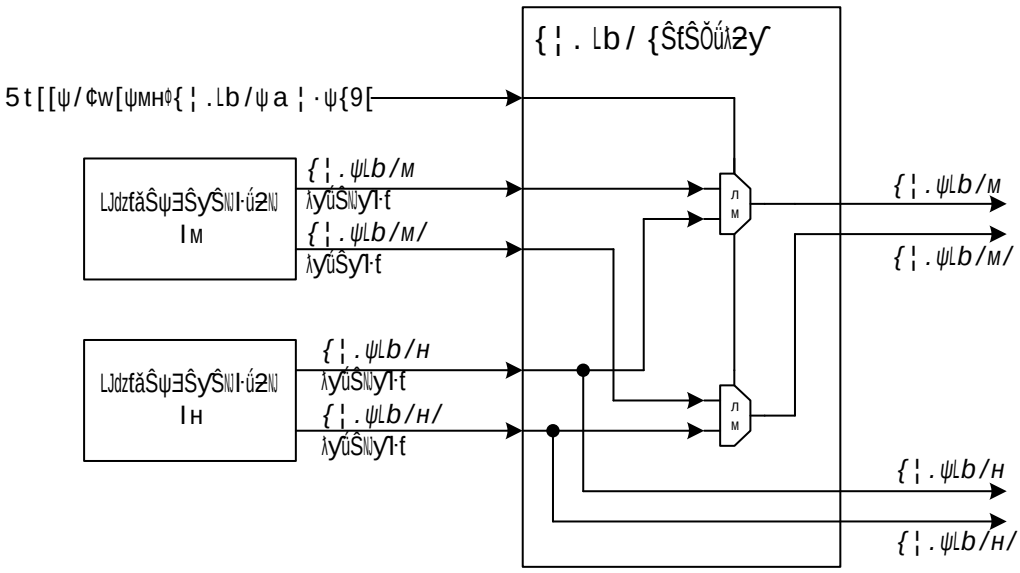
21.9.4 Sub pulse generation for DPLL_CTRL_1.SMC=1

21.9.4.1 Necessity of two pulse generators

The adder of the picture in chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "* must be implemented twice in the case of **DPLL_CTRL_1.SMC** =1: one for *SUB_INC1* controlled by the *TRIGGER* input and (while **DPLL_CTRL_0.RMO** =1) one for *SUB_INC2* , controlled by the *STATE* input. In the case described in the chapter above for **DPLL_CTRL_1.SMC** =0, only one adder is used to generate *SUB_INC1* controlled by the *TRIGGER* in normal mode or by *STATE* in emergency mode.

21.9.4.2 Multiplexing of the pulse generators

Figure 155 Subinc1/subinc2 multiplexer



It is possible to multiplex the outputs of pulse generator 2 (*SUB_INC2* , *SUB_INC2C*) to the DPLL outputs *SUB_INC1* , *SUB_INC1C* . This is shown in figure *155 "Subinc1/subinc2 multiplexer"* . If both the signal processing channels *TRIGGER* and *STATE* are operating for reasons of redundancy in DPLL mode **DPLL_CTRL_1.SMC** =1 on the same system (engine) this can be used to switch the pulse source immediately between the two signal sources. The multiplexer is controlled by the bit field **DPLL_CTRL_12.SUBINC_MUX_SEL** . See chapter **DPLL_CTRL_12** .

This multiplexer can be used alternatively to the control signal **DPLL_CTRL_0.RMO** to switch the source of pulse generation for *CCM[0]_TBU_TS1* . By using this multiplexer the pulse generation is continuously ongoing when switching instead of stopping the pulse generation while the input signal on e.g. *TRIGGER* is lost, until the next active input signal (e.g. on *STATE*) in the new context occurs when using **DPLL_CTRL_0.RMO** .

21.9.4.3 Calculate the number of pulses to be sent for the first device using the automatic end mode condition

[eq_199]
IF(DPLL_CTRL_1.SMC=1 and DPLL_CTRL_1_SHADOW_TRIGGER.DMO=0)
NMB_T = (MLS1 + PD_store) *SYN_T + MP + MPVAL1

with PD_store = **DPLL_ADT_T[p]** [12:0], prefetched during last increment,

SYN_T = **DPLL_ADT_T[p][18:16]** , prefetched during last increment, and

DPLL_MPVAL1.MPVAL1 = pulse correction value for **DPLL_CTRL_1_SHADOW_TRIGGER.PCM1** =1
while the value for PD_store is zero for **DPLL_CTRL_0_SHADOW_TRIGGER.AMT** =0
and
for **DPLL_CTRL_1_SHADOW_TRIGGER.COA** =0 use zero instead of the value of MP

21.9.4.4 Calculate the number of pulses to be sent for the second device using the automatic end mode condition

for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1, **DPLL_CTRL_1.SMC** =1 and **DPLL_CTRL_1_SHADOW_STATE.DMO** =0:

[eq_200]
IF(DPLL_CTRL_0_SHADOW_STATE.RMO=1, DPLL_CTRL_1.SMC=1 and DPLL_CTRL_1_SHADOW_STATE.DMO=0)
NMB_S = (MLS2 + PD_S_store) *SYN_S+ MP + MPVAL2

with PD_S_store = **DPLL_ADT_S[p]** [15:0], prefetched during last increment,

DPLL_NUSC_SYN_S = **DPLL_ADT_S[p]** [21:16], prefetched during last increment, and

DPLL_MPVAL2.MPVAL2 = pulse correction value for **DPLL_CTRL_1_SHADOW_STATE.PCM2** =1

while the value for PD_S_store is zero for **DPLL_CTRL_0_SHADOW_STATE.AMS** =0
and
for **DPLL_CTRL_1_SHADOW_STATE.COA** =0 use zero instead of the value of MP

Note:

These calculations above in equations *[eq_199]* and *[eq_200]* are only valid for an automatic end mode (**DPLL_CTRL_1_SHADOW_STATE.DMO** = 0). In addition, the number of generated pulses is added by 0.5 as shown in equations *[eq_116]* , *[eq_118]* or *[eq_203]* , *[eq_204]* respectively in order to compensate rounding down errors at the succeeding division operation. Because in automatic end mode the number of pulses is limited by **DPLL_INC_CNT1.INC_CNT1** and **DPLL_INC_CNT2.INC_CNT2** it is guaranteed, that not more pulses than needed are generated and in the same way missing pulses are made up for the next increment.

21.9.4.5 Calculate ADD_IN for the first device for DPLL_CTRL_1.SMC=1

The sub-pulse generation in this case is done by the following calculations using a 24 bit adder with a carry out c_{out} and the following inputs:
- *ADD_IN*
- the second input is the (delayed) output of the adder, stored with each time stamp clock.

Replace *ADD_IN* by **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** (when calculated, **DPLL_CTRL_1_SHADOW_TRIGGER.DLM1** =0) or **DPLL_ADD_IN_LD1.ADD_IN_LD1** (when provided by the CPU, **DPLL_CTRL_1_SHADOW_TRIGGER.DLM1** =1) respectively while:

[eq_201]
IF(DPLL_CTRL_1.SMC=1 and LOW_RES=DPLL_CTRL_1.TS0_HRT)
ADD_IN_CAL1= (NMB_T+0.5) * RCDT_TX

When **DPLL_RCDT_TX.RCDT_TX** is the 2^{32} time value of the quotient in equation *[eq_164]* .

[eq_202]
IF(DPLL_CTRL_1.SMC=1 and LOW_RES= 1 and DPLL_CTRL_1.TS0_HRT=0)
ADD_IN_CAL1= (NMB_T+0.5) * (RCDT_TX /8)

When **DPLL_RCDT_TX.RCDT_TX** is the 2^{32} time value of the quotient in equation *[eq_164]* .

In order to not complicate the calculation procedure, use a Multiplier with a sufficient bit width at the output and use the corresponding shifted output bits.

ADD_IN_CAL1 is a 24 bit integer value. The **DPLL_CDT_TX.CDT_TX** is the expected duration of current *TRIGGER* increment.
The c_{out} of the adder directly influences the *SUB_INC1* output of the DPLL (see chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*). The *SUB_INC1* output is in automatic end mode only enabled by *EN_C1C* when **DPLL_INC_CNT1.INC_CNT1** >0.

21.9.4.6 Calculate ADD_IN for the second device for DPLL_CTRL_1.SMC=1

Replace *ADD_IN* by **DPLL_ADD_IN_CAL2.ADD_IN_CAL2** (when calculated, **DPLL_CTRL_1_SHADOW_STATE.DLM2** =0) or **DPLL_ADD_IN_LD2.ADD_IN_LD2** (when provided by the CPU, **DPLL_CTRL_1_SHADOW_STATE.DLM2** =1) respectively while:

[eq_203]
IF(DPLL_CTRL_1.SMC=1 and DPLL_CTRL_0_SHADOW_STATE.RMO=1 and LOW_RES=DPLL_CTRL_1.TS0_HRS)
ADD_IN_CAL2= (NMB_S+0.5)* RCDT_SX

When **DPLL_RCDT_SX.RCDT_SX** is the 2^{32} time value of the quotient in equation *[eq_179]* in chapter *21.8.8.2 "Calculate the position stamp forwards"* .

[eq_204]
IF(DPLL_CTRL_1.SMC=1 and DPLL_CTRL_0_SHADOW_STATE.RMO=1 and LOW_RES=1 and DPLL_CTRL_1.TS0_HRS=0)
ADD_IN_CAL2= (NMB_S+0.5)* (RCDT_SX /8)

When **DPLL_RCDT_SX.RCDT_SX** is the 2^{32} time value of the quotient in equation *[eq_179]* in chapter *21.8.8.2 "Calculate the position stamp forwards"* .

In order to not complicate the calculation procedure, use a Multiplier with a sufficient bit width at the output and use the corresponding shifted output bits.

The c_{out} of the adder2 directly influences the *SUB_INC2* output of the DPLL (see chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*).
The *SUB_INC2* output is in automatic end mode only enabled by *EN_C2C* when **DPLL_INC_CNT2.INC_CNT2** >0.

After RESET and after *EN_C1C* =0/ *EN_C2C* =0 (after stopping in automatic end mode) the storage elements have a zero value and also *EN_C1U* / *EN_C2U* has to be zero until reliable *ADD_IN* values are available and the pulse generation starts. The calculated values for the increment prediction using equations *[eq_21]* in chapter *21.7.2.7 "Calculate the current increment value"* , equation *[eq_58]* in chapter *21.7.4.4 "Calculate the current increment value"* , equation *[eq_45]* in chapter *21.7.3.7 "Calculate the current increment (nominal value)"* or equation *[eq_71]* in chapter *21.7.5.4 "Calculate the current increment value"* respectively are valid only when **DPLL_NUTC.NUTE** >1 or DPLL_NUSC_NUSE>1 respectively. For **DPLL_NUTC.NUTE** =1 or DPLL_NUSC_NUSE=1 respectively the equations *[eq_116]* or *[eq_118]* (see chapter *21.9.4.5 "Calculate ADD_IN for the first device for DPLL_CTRL_1.SMC=1"*) and equation *[eq_203]* or equation *[eq_204]* (see chapter *21.9.4.6 "Calculate ADD_IN for the second device for DPLL_CTRL_1.SMC=1"*) use the actual increment value subtracted by the weighted average error.

The generation of *SUB_INC1* and *SUB_INC2* pulses depends on the configuration of the DPLL.
In automatic end mode the counter **DPLL_INC_CNT1.INC_CNT1** and **DPLL_INC_CNT2.INC_CNT2** resets the enable signal *EN_C1C* / *EN_C2C* when the number of pulses desired is reached. In this case only the uncompensated outputs *SUB_INC1* and *SUB_INC2* remain active in order to provide pulses for the input filter units. A new *TRIGGER* or *STATE* input respectively can reset the storage elements and also *ADD_IN* , especially when *EN_C1C* / *EN_C2C* was zero before. In the case of acceleration missing pulses can be determined at the next *TRIGGER* / *STATE* event easily. For the correction strategy **DPLL_CTRL_1.COA** = 0 those missing pulses are sent out with *CCM[0]_CLK_RES* [0:0] frequency as soon they are determined. After that the pulse counter **DPLL_INC_CNT1.INC_CNT1** and **DPLL_INC_CNT2.INC_CNT2** should always be zero and the new pulses are sent out afterwards, when **DPLL_INC_CNT1.INC_CNT1** and **DPLL_INC_CNT2.INC_CNT2** is set to the desired value by adding **DPLL_MLS1.MLS1** or **DPLL_MLS2.MLS2** for the new *TRIGGER* or *STATE* event respectively.

Because the used DIV procedure of the algorithms results only in integer values, a systematic failure could appear. The pulse generation will stop when the **DPLL_INC_CNT1.INC_CNT1** and **DPLL_INC_CNT2.INC_CNT2** register reaches zero or all remaining pulses at a new increment will be considered in the next calculation. In this way the loss of pulses can be avoided.

When a new *TRIGGER* appears the value of **DPLL_NUTC.SYN_T** * **DPLL_MLS1.MLS1** is added to **DPLL_INC_CNT1.INC_CNT1** . Therefore for FULL_SCALE 2*(**DPLL_CTRL_0.TNU** +1)* **DPLL_MLS1.MLS1** pulses *SUB_INC1* generated, when **DPLL_INC_CNT1.INC_CNT1** reaches the value zero. The generation of *SUB_INC1* pulses has to be done as fast as possible.

When a new *STATE* appears the value of DPLL_NUSC_SYN_S* **DPLL_MLS2.MLS2** is added to **DPLL_INC_CNT2.INC_CNT2** . Therefore for FULL_SCALE 2*(DPLL_SNU+1)* **DPLL_MLS2.MLS2** pulses *SUB_INC2* is generated, when **DPLL_INC_CNT2.INC_CNT2** reaches the zero value. The generation of *SUB_INC2* pulses has to be done as fast as possible.

21.9.5 Calculation of the Accurate Position Values

All appearing *TRIGGER* and *STATE* signals have a time stamp and a position stamp assigned after the input filter procedure. For the calculation of the exact time stamp the filter values are considered in the calculations of equations *21.7.2.1 "Calculate trigger time stamps"* or *21.7.3.1 "Calculate STATE signal time stamps"* respectively. A corresponding calculation is to be performed for the calculation of position values.
The **DPLL_PSTC.PSTC** and **DPLL_PSSC.PSSC** values can be corrected by the CPU, when needed.

After reset, while **DPLL_STATUS.FTD** =0 and no active *TRIGGER* slope is detected:
DPLL_PSTC.PSTC = 0

Calculate the new position value for each active *TRIGGER* event:

[eq_205]
IF(DPLL_STATUS.FTD=1 and DPLL_CTRL_1_SHADOW_TRIGGER.SGE1=1)
DPLL_PSTC.PSTC= PSTC_old + DPLL_NMB_T_TAR_OLD.NMB_T_TAR_OLD

with
PSTC_old is the last **DPLL_PSTC.PSTC** value and
NMB_T_old is the number of pulses which are calculated
and provided for sending out in the last increment.

After reset, while **DPLL_STATUS.FSD** =0 and no active *STATE* slope is detected:
DPLL_PSSC.PSSC = 0

Calculate the new position value for each *STATE* event:

[eq_206]
IF((DPLL_STATUS.FSD=1 and DPLL_CTRL_1_SHADOW_STATE.SGE1=1 and DPLL_CTRL_1.SMC=0) or (DPLL_CTRL_1_SHADOW_STATE.SGE2=1 and DPLL_CTRL_1.SMC=1))

DPLL_PSSC.PSSC= PSSC_old + DPLL_NMB_S_TAR_OLD.NMB_S_TAR_OLD

Note:

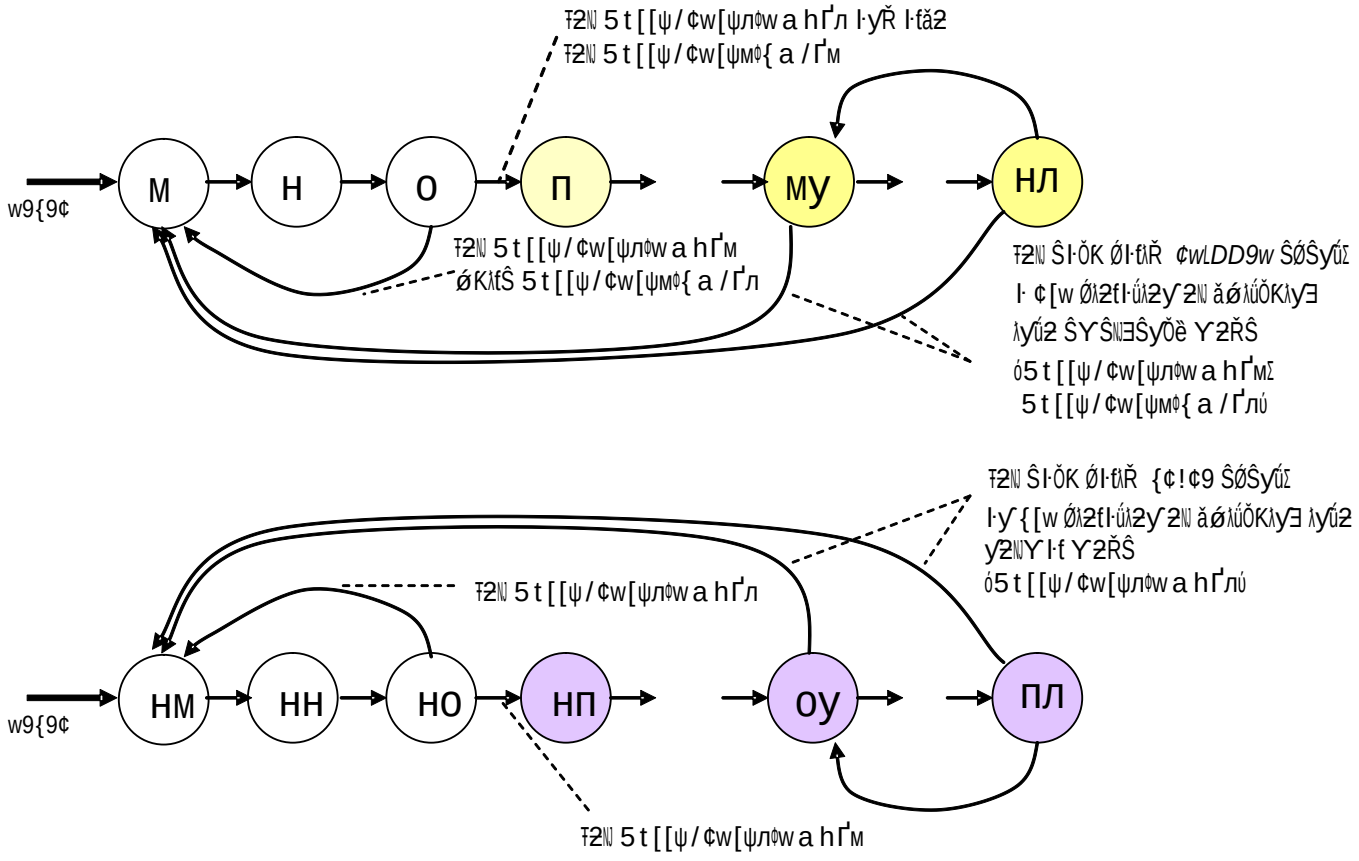
With
(**DPLL_PSSC.PSSC**)_old is the last **DPLL_PSSC.PSSC** value and
(**DPLL_NMB_S.NMB_S**)_old is the number of pulses which are calculated
and provided for sending out in the last increment.

21.9.6 Scheduling of the Calculation

After enabling the DPLL with each active *TRIGGER* or *STATE* event respectively a cycle of operations is performed to calculate all the results shown in detail in the table of chapter *51 "State description of the State Machine Table"* below. A state machine controls this procedure and consists of two parts, the first is triggered by an active slope of the signal *TRIGGER* , begins at step 1 and ends at step 20 (in normal mode and for **DPLL_CTRL_1.SMC** =1). The second state machine is controlled by an active slope of the signal *STATE* , begins at step 21 and ends at step 40 (in emergency mode and also for **DPLL_CTRL_1.SMC** = **DPLL_CTRL_0.RMO** =1). Depending on the mode used all 20 steps are executed or already after 2 steps the jump into the initial state is performed, as shown in the state machine descriptions below. For each new extended cycle (without this jump) all prediction values for Actions in the case **DPLL_CTRL_1.SMC** =0 are calculated once more (with maybe improved accuracy because of better parameters) and all pending decisions are made using these new values when transmitted to the decision device.

In chapter *51 "State description of the State Machine Table"* the steps of the state machine are described. The elaboration of the steps depends on the configuration bits described in the comments. The steps 4 to 17 are only calculated in normal mode (in the state machine, explanation below circles yellow marked in chapter *156 "State machine partitioning for normal and emergency mode."*), but steps 24 to 37 are only calculated in emergency mode (in the state machine, explanation below circles marked in magenta in chapter *156 "State machine partitioning for normal and emergency mode."*) when **DPLL_CTRL_1.SMC** =0.

Figure 156 State machine partitioning for normal and emergency mode.



BOSCH

21.9.6.1 Synchronization description

TRIGGER :
The **DPLL_APT.APT** (address pointer for duration and reciprocal duration values of *TRIGGER* increments) is initially set to zero and incremented with each active *TRIGGER* event. Therefore, data are stored in the RAM beginning from the first available value. The actual duration of the last increment is stored at **DPLL_DT_T_ACT.DT_T_ACT** . For the prediction of the next increment it is assumed, that the same value is valid as long as **DPLL_NUTC.NUTE** is one.

A missing *TRIGGER* is assumed, when at least after **DPLL_TOV** * **DPLL_DT_T_ACT.DT_T_ACT** no active *TRIGGER* event occurs.

The data of equations *[eq_140]* and *[eq_142]* of chapter *21.8.5.4 "Update of RAM by DT_T[p] and RDT_T[p] after calculation"* are written in the corresponding RAM regions and **DPLL_APT.APT** is incremented accordingly up to 2* **DPLL_CTRL_0.TNU** −2* **DPLL_CTRL_1.SYN_NT** +1.

The **DPLL_APT.APT_2B** (address pointer for the time stamp field of *TRIGGER*) is initially set to zero and incremented with each active *TRIGGER* event. When no gap is detected because of the incomplete synchronization process at the beginning, for all *TRIGGER* events the time stamp values are written in the RAM up to 2*(**DPLL_CTRL_0.TNU** +1) entries, although only 2*(**DPLL_CTRL_0.TNU** +1− **DPLL_CTRL_1.SYN_NT**) events in FULL_SCALE appear. When the current position is detected, the synchronization procedure can be performed as described below:
Before the CPU sets the **DPLL_APT_2C.APT_2C** address pointer in order to synchronize to the profile, it writes the corresponding increment value for the necessary extension of the RAM region 2b value **DPLL_APT_SYNC.APT_2B_EXT** into the register **DPLL_APT.APT_2B** _sync and sets the status bit **DPLL_APT_SYNC.APT_2B_STATUS** . This value can be e.g. 2* **DPLL_CTRL_1.SYN_NT** , when all gaps in FULL_SCALE already passed the input data stream of *TRIGGER* , or less then this value, when up to now e.g. only a single gap is to be considered in the data stream stored already in the RAM region 2b. The number of virtual increments to be considered depends on the number of inputs already got. After writing **DPLL_APT_2C.APT_2C** by the CPU, with the next *TRIGGER* event the **DPLL_APT.APT_2B** address pointer is incremented (as usual) and then the additional offset value **DPLL_APT_SYNC.APT_2B_EXT** is added to it once (while **DPLL_APT_SYNC.APT_2B_STATUS** =1 and for forward direction). For that reason the **DPLL_APT_SYNC.APT_2B_STATUS** bit is reset after it. The old **DPLL_APT.APT_2B** value before adding the offset is stored in the **DPLL_APT_SYNC.APT_2B_OLD** register as information for the CPU where to start the extension procedure. In the following the CPU fills in the time stamp field around the **DPLL_APT_SYNC.APT_2B_OLD** position taking into account the corresponding number of virtual entries stored in the **DPLL_APT_SYNC.APT_2B_EXT** value and the corresponding **DPLL_ADT_T[p].NT** values in the profile. The extension procedure ends when all gaps considered in the **DPLL_APT_SYNC.APT_2B_EXT** value are treated once. In the consequence all storage locations of RAM region 2b up to now do have the corresponding entries. Future gaps are treated by the DPLL.
For a backward direction the APT_2C_ext value is subtracted accordingly.

When the CPU writes the **DPLL_APT_2C.APT_2C** address pointer the **DPLL_STATUS.SYT** bit is set simultaneously. For **DPLL_STATUS.SYT** =1 in normal mode (**DPLL_CTRL_1.SMC** =0) the **DPLL_STATUS.LOCK1** bit is set with the system clock, when the right number of increments between two synchronization gaps is detected by the DPLL. An unexpected missing *TRIGGER* or an additional *TRIGGER* between two synchronization gaps does reset the **DPLL_STATUS.LOCK1** bit in normal mode. In that case the CPU must correct the *SUB_INC1* and *SUB_INC2* pulse number and maybe correct the **DPLL_APT_2C.APT_2C** pointer. For this purpose the **DPLL_IRQ_NOTIFY.LL1I** interrupt can be used.

When **DPLL_STATUS.SYT** is set the calculations of equations in chapters *21.7.2.1 "Calculate trigger time stamps"* to *21.7.2.7 "Calculate the current increment value"* or chapter *21.7.4.4 "Calculate the current increment value"* are performed accordingly and the values are stored in (and distributed to) the right RAM positions.
This includes the multiple time stamp storage by the DPLL for a gap according to equations *[eq_134]* to *[eq_136]* forwards (chapter *21.8.5.2 "Extend the time stamp values for TRIGGER signal in forward direction"*) or backwards (chapter *21.8.5.3 "Extend the time stamp values for trigger signal in backward direction"*). The **DPLL_APT.APT_2B** pointer is for that reason incremented or decremented before this operation considering the virtual increments in addition.
For the **DPLL_APT.APT** and **DPLL_APT_2C.APT_2C** pointers the gap is considered as a single increment.

STATE :
The DPLL_APS_APS (address pointer for duration and reciprocal duration values of *STATE*) is initially set to zero and incremented with each active *STATE* event. Therefore, data are stored in the RAM field beginning at the first location. The actual duration of the last increment is stored at **DPLL_DT_S_ACT.DT_S_ACT** . For the prediction of the next increment it is assumed, that the same value is valid as long as DPLL_NUSC_NUSE is one.

A missing *STATE* is assumed, when at least after SOV* **DPLL_DT_S_ACT.DT_S_ACT** no active *STATE* event appears.

The data of equations in chapter *21.8.5.8 "Update of RAM by DT_S[p] and RDT_S[p] after calculation"* is written in the corresponding RAM regions and DPLL_APS_APS is incremented accordingly up to 2*DPLL_SNU−2*DPLL_SYN_NS+1 (for **DPLL_CTRL_1.SYSF** =0).

The DPLL_APS_1C2 (address pointer for the time stamp field of *STATE*) is initially set to zero and incremented with each active *STATE* event. When no gap is detected because of the incomplete synchronization process at the beginning, for all *STATE* events the time stamp values are written in the RAM up to 2*(DPLL_SNU+1) entries, although (e.g. for **DPLL_CTRL_1.SYSF** =0) only 2*(DPLL_SNU+1−DPLL_SYN_NS) events in FULL_SCALE appear. When the current position is detected, the synchronization procedure can be performed as described below:
Before the CPU sets the DPLL_APS_1C3_1C3 address pointer in order to synchronize to the profile, it writes the corresponding increment value DPLL_APS_SYNC_APS_1C2_EXT for the necessary extension of the RAM region 1c2 into the register **DPLL_APS_SYNC_EXT** and sets the DPLL_APS_SYNC_APS_1C2_STATUS bit there. This value can be e.g. 2*DPLL_SYN_NS (for **DPLL_CTRL_1.SYSF** =0) or DPLL_SYN_NS (for **DPLL_CTRL_1.SYSF** =1), when all gaps in FULL_SCALE already passed the input data stream of *STATE* . Also less then this value can be considered, when up to now only a single gap is to be considered in the data stream stored already in the RAM region 1c2. The number of increments to be considered depends on the number of inputs already got. After writing DPLL_APS_1C3_1C3 by the CPU, with the next active *STATE* slope the DPLL_APS_1C2 address pointer is incremented (as usual) and then the additional offset value DPLL_APS_SYNC_APS_1C2_EXT is added to it once (while DPLL_APS_SYNC_APS_1C2_STATUS=1 and forward direction). For that reason the DPLL_APS_SYNC_APS_1C2_STATUS bit is reset after it. The old DPLL_APS_1C2 value is stored in the DPLL_APS_SYNC_APS_1C2_OLD register as information for the CPU where to start the extension procedure. In the following the CPU extends the time stamp field beginning from the DPLL_APS_SYNC_APS_1C2_OLD position taking into account the corresponding number of virtual entries according to the DPLL_APS_SYNC_APS_1C2_EXT value and also the corresponding NS values in the profile. The extension procedure ends when all gaps considered in the DPLL_APS_SYNC_APS_1C2_EXT value are treated once. In the consequence all storage locations of RAM region 1c2 up to now do have the corresponding entries. Future gaps are

treated by the DPLL.
For a backward direction the DPLL_APS_SYNC_APS_1C2_EXT value is subtracted accordingly.

When the CPU writes the DPLL_APS_1C3_1C3 address pointer the **DPLL_STATUS.SYS** bit is set simultaneously. For **DPLL_STATUS.SYS** =1 in emergency mode (**DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0.RMO** =1) the **DPLL_STATUS.LOCK1** bit is set with the system clock, when the right number of increments between two synchronization gaps is detected by the DPLL. An unexpected missing *STATE* or an additional *STATE* between two synchronization gaps does reset the **DPLL_STATUS.LOCK1** bit in emergency mode. In that case the CPU must correct the *SUB_INC1* pulse number and maybe correct the DPLL_APS_1C3_1C3 pointer. For this purpose the **DPLL_IRQ_NOTIFY.LL1I** interrupt can be used.

When **DPLL_STATUS.SYS** is set the calculations of equations *21.7.3.1 "Calculate STATE signal time stamps"* to *21.7.3.7 "Calculate the current increment (nominal value)"* or *21.7.5.4 "Calculate the current increment value"* are performed accordingly and the values are stored in (and distributed to) the right RAM positions.
This includes the multiple time stamp storage by the DPLL for a gap according to equations *21.8.5.6 "Extend the time stamp values for STATE signal"* forward or in backward direction equation *21.8.5.7 "Extend the time stamp values for STATE signal for backward direction"* . Therefore, the DPLL_APS_1C2 pointer is incremented or decremented additionally before this operation considering the virtual increments.
For the DPLL_APS_APS and DPLL_APS_1C2 pointers the gap is considered as a single increment.

DPLL_CTRL_1.SMC =1:
For **DPLL_CTRL_1.SMC** =1 it is assumed, that the starting position is known by measuring the characteristic of the device. In this way the **DPLL_APT.APT** and **DPLL_APT_2C.APT_2C** as well the DPLL_APS_APS and DPLL_APS_1C3_1C3 values are set properly, maybe with an unknown repetition rate. When no gap is to be considered for *TRIGGER* or *STATE* signals the **DPLL_APT.APT_2B** and DPLL_APS_1C2 address pointers are set equal to **DPLL_APT.APT** or DPLL_APS_APS respectively. It is assumed, that all missing *TRIGGER* and missing *STATE* can be also considered from the beginning, when a valid profile with the corresponding adapted values is written in the RAM regions 1c3 and 2c respectively. In that case the **DPLL_TSF_T[p].TSF_T** and **DPLL_TSF_S[p].TSF_S** must be extended by the DPLL according to the profile. . Thus the **DPLL_STATUS.SYT** and **DPLL_STATUS.SYS** bits could be set from the beginning and the **DPLL_STATUS.LOCK1** and **DPLL_STATUS.LOCK2** bits are set after recognition of the corresponding gaps accordingly. When no gap exists (**DPLL_CTRL_1.SYN_NT** =0 or DPLL_SYN_NS=0), the LOCK bits are set immediately. The CPU can correct the **DPLL_APT_2C.APT_2C** and DPLL_APS_1C3_1C3 pointer according to the recognized repetition rate later once more without the loss of **DPLL_STATUS.LOCK1** and **DPLL_STATUS.LOCK2** .

21.9.6.2 Operation for direction change in normal and emergency mode (DPLL_CTRL_1.SMC=0)

When for **DPLL_CTRL_1.SMC** =0 in normal mode a backwards condition is detected for the *TRIGGER* input signal (e.g. when THMI is not violated), the **DPLL_STATUS.LOCK1** bit is reset, the **DPLL_NUTC.NUTE** value is set to 1 (the same for DPLL_NUSC_NUSE). The address pointers **DPLL_APT_2C.APT_2C** as described below (and after that decremented for each following active slope of *TRIGGER* as long as the *DIR1* bit shows the backward direction).
Please note, that in the case of the change in the direction, the **DPLL_STATUS.ITN** and **DPLL_STATUS.ISN** bit are reset.

For the transition to the backward direction, no change of address pointer **DPLL_APT.APT** and **DPLL_APT.APT_2B** is necessary.

Profile update for *TRIGGER* when changing direction
The profile address pointer **DPLL_APT_2C.APT_2C** is changed step by step in order to update the profile information in **DPLL_NUTC.SYN_T** , **DPLL_NUTC.SYN_T_OLD** and PD_store:

- decrement **DPLL_APT_2C.APT_2C** , load **DPLL_NUTC.SYN_T**
- decrement **DPLL_APT_2C.APT_2C** , load **DPLL_NUTC.SYN_T**
- decrement **DPLL_APT_2C.APT_2C** , load **DPLL_NUTC.SYN_T** , PD_store, update **DPLL_NUTC.SYN_T_OLD**
- decrement **DPLL_APT_2C.APT_2C** , make calculations, load **DPLL_NUTC.SYN_T** and PD_store, update **DPLL_NUTC.SYN_T_OLD** and PD_store_old and wait for a new *TRIGGER* event.

Note:

The update of **DPLL_NUTC.SYN_T_OLD** and the loading of PD_store can be performed in all steps above. The value of **DPLL_APT.APT_2B** needs not to be corrected. In order to change direction from backward to forward, make the same corrections by incrementing **DPLL_APT_2C.APT_2C** .

Making calculations means the operation of the state machine starts with the calculations of **DPLL_NMB_T.NMB_T** and **DPLL_INC_CNT1.INC_CNT1** using the actual **DPLL_APT_2C.APT_2C** address pointer value, see table in chapter *51 "State description of the State Machine Table"* .

The *CCM[0]_TBU_TS1* value is to be corrected by the number of pulses sent out in the wrong direction mode during the last and current increment. This correction is done by sending out *SUB_INC1* pulses for decrementing *CCM[0]_TBU_TS1* (while *DIR1* =1).

Save **DPLL_INC_CNT1.INC_CNT1** value at direction change to inc_cnt1_save.
Calculate the new **DPLL_INC_CNT1.INC_CNT1** value as follows:

1. Stop sending pulses and save **DPLL_INC_CNT1.INC_CNT1** as **DPLL_INC_CNT1.INC_CNT1** _save at the time of change in direction.
2. Set **DPLL_INC_CNT1.INC_CNT1** to the target value of the last increment **DPLL_NMB_T_TAR_OLD.NMB_T_TAR_OLD**
3. Add the target number of *TRIGGER* pulses, which was calculated for the current increment when this value was already added to **DPLL_INC_CNT1.INC_CNT1** before the change in direction is detected + **DPLL_NMB_T_TAR.NMB_T_TAR**
4. Subtract the value of the pulses not sent yet (remaining value at **DPLL_INC_CNT1.INC_CNT1** _save) – **DPLL_INC_CNT1.INC_CNT1** _save

5. Calculate the new target pulses to be sent considering the new values of **DPLL_NUTC.SYN_T** and **DPLL_ADT_T[p].PD** _store and add them: + **DPLL_NMB_T_TAR.NMB_T_TAR** _new

This results in the following equation:
DPLL_INC_CNT1.INC_CNT1 = **DPLL_NMB_T_TAR_OLD.NMB_T_TAR_OLD** + **DPLL_NMB_T_TAR.NMB_T_TAR**
– **DPLL_INC_CNT1.INC_CNT1** _save + **DPLL_NMB_T_TAR.NMB_T_TAR** _new
All pulses summarized at **DPLL_INC_CNT1.INC_CNT1** are sent out by the maximum possible frequency, because no speed information is available for the first increment after changing the direction. Please note that no pulse correction using **DPLL_CTRL_1_SHADOW_TRIGGER.PCM1** is possible during the change in direction.
When **DPLL_PSTC.PSTC** was incremented/decremented at the active slope and after that the direction change was detected at the same input event, correct **DPLL_PSTC.PSTC** once by
– **DPLL_NMB_T_TAR_OLD.NMB_T_TAR_OLD** when changed backward
+ **DPLL_NMB_T_TAR_OLD.NMB_T_TAR_OLD** when changed forward
in order to compensate the former operation. When the direction information is known before an intended change of **DPLL_PSTC.PSTC** , do not change them.
Store the new calculated value **DPLL_NMB_T_TAR.NMB_T_TAR** _new at **DPLL_NMB_T_TAR.NMB_T_TAR** for the correct calculation of **DPLL_PSTC.PSTC** at the next input event.

consequences for *STATE*
With the next active *STATE* event the direction information is already given. The profile pointer DPLL_APS_1C3_1C3 is to be corrected by a two times decrement in order to point to the profile of the next increment. In the following it is decremented with each *STATE* event while *DIR1* =1. The DPLL_NUSC_SYN_S and PD_S_store values must be updated accordingly, including DPLL_NUSC_SYN_S_OLD and PD_S_store_old.

Because the right direction is already known when an input event appears, make the following corrections:
– decrement DPLL_APS_1C3_1C3, load DPLL_NUSC_SYN_S and PD_S_store, update DPLL_NUSC_SYN_S_OLD and PD_S_store_old
– decrement DPLL_APS_1C3_1C3, make calculations, load DPLL_NUSC_SYN_S and PD_S_store, update DPLL_NUSC_SYN_S_OLD and PD_S_store_old and wait for a new *STATE* event.

Note:
The update of DPLL_NUSC_SYN_S_OLD and the loading of PD_S_store can be performed in all steps above. The value of DPLL_APS_1C2 need not be corrected.

When a new *STATE* event occurs, all address pointers are decremented accordingly as long as *DIR1* =1.

In emergency mode the pulses are corrected as follows:
Save **DPLL_INC_CNT1.INC_CNT1** value at direction change to inc_cnt1_save.
Calculate the new **DPLL_INC_CNT1.INC_CNT1** value as follows:
1. Stop sending pulses and save **DPLL_INC_CNT1.INC_CNT1** at the moment of direction change as inc_cnt1_save.
2. Set **DPLL_INC_CNT1.INC_CNT1** to the target value of the current increment
nmb_s_tar
In contrast to the normal mode, nmb_s_tar is to be used instead of nmb_s_tar_old, because direction information in emergency mode is only given from the *TRIGGER* input and occurs independent of a *STATE* event.
That means: The calculations at the last *STATE* event were done for the correct former direction. In addition, pulse calculations are still not performed for the current increment, because the direction change is known at the time of the recent *STATE* event. Further changes in the direction are considered at the next *STATE* event.
3. Do not add the calculated number of STATE pulses because no new *STATE* event occurred.
4. Subtract the value of the target pulses (remaining value at inc_cnt1_save) that are still not sent
– inc_cnt1_save
5. Add the new calculated target pulses for the current increment
+ nmb_s_tar_new
when for the calculation all new conditions of PD_S_store and DPLL_NUSC_SYN_S are considered.
inc_cnt1 = nmb_s_tar_old – inc_cnt1_save + nmb_s_tar_new
All pulses summarized at **DPLL_INC_CNT1.INC_CNT1** are sent out by the maximum possible frequency, because no speed information is available for the first increment after changing the direction. Please note that no pulse correction using **DPLL_CTRL_1_SHADOW_STATE.PCM1** is possible during direction change.
Do not change **DPLL_PSSC.PSSC** and suppress incrementing/decrementing of **DPLL_PSSC.PSSC** at the event that directly follows the information about the change in direction.
Store the new calculated value nmb_s_tar_new at nmb_s_tar for the correct calculation of **DPLL_PSTC.PSTC** at the next input event.

Repeated change to forward direction for *TRIGGER*
The *DIR1* bit remains set as long as the **DPLL_THMI.THMI** value is not violated for the following *TRIGGER* events and is reset when for an inactive *TRIGGER* slope the **DPLL_THMI.THMI** is violated.

Resetting the *DIR1* to 0 results (after repeated reset of **DPLL_STATUS.LOCK1** , **DPLL_STATUS.ITN** , **DPLL_STATUS.ISN**) the opposite correction of the profile address pointer considered.

This means two increment operations of the address pointer DPLL_APS_1C3_1C3 including the update of DPLL_NUSC_SYN_S and PD_S_store with the automatic update of DPLL_NUSC_SYN_S_OLD and PD_S_store_old for *STATE* and
four increment operations of the address pointer **DPLL_APT_2C.APT_2C** including the update of **DPLL_NUTC.SYN_T** and PD_store with the automatic update of **DPLL_NUTC.SYN_T_OLD** and PD_store_old for *TRIGGER* .

The correction of TBU_CH1 is done by sending out the correction pulses with the highest possible frequency at *SUB_INC1* while *DIR1* =0. The number of pulses is calculated as shown above.

Consequences for *STATE*
see corrections above. After that the address pointers are incremented again with each following active *STATE* event as long as *DIR1* =0.

21.9.6.3 Operation for direction change for TRIGGER signal (DPLL_CTRL_1.SMC=1)

When for **DPLL_CTRL_1.SMC** =1 a backward condition is detected for the *TRIGGER* input signal (*TDIR* =1, resulting in *DIR1* =1), the **DPLL_STATUS.LOCK1** bit in the **DPLL_STATUS** register is reset, the **DPLL_NUTC.NUTE** value is set to 1. The address pointers **DPLL_APT.APT** and **DPLL_APT_2C.APT_2C** as well as **DPLL_APT.APT_2B** are decremented for each active slope of *TRIGGER* as long as the *DIR1* bit shows the backward direction.
Please note, that in the case of change in the direction the **DPLL_STATUS.ITN** bit is reset.

Profile update for *TRIGGER*
Make the same update steps for the profile address pointer as shown in chapter *21.9.6.2 "Operation for direction change in normal and emergency mode (DPLL_CTRL_1.SMC=0)"* : Decrement **DPLL_APT_2C.APT_2C** for 2 times with the update of the **DPLL_NUTC.SYN_T** and PD_store values at each step with an automatic update of **DPLL_NUTC.SYN_T_OLD** and PD_store_old:

1. decrement **DPLL_APT_2C.APT_2C** , load **DPLL_NUTC.SYN_T** , PD_store, update **DPLL_NUTC.SYN_T_OLD**
2. decrement **DPLL_APT_2C.APT_2C** , make calculations, load **DPLL_NUTC.SYN_T** and PD_store, update **DPLL_NUTC.SYN_T_OLD** and PD_store_old and wait for a new *TRIGGER* event.

In the normal case no correction of wrong pulses sent is necessary, because the change in direction is detected immediately by the pattern. However, a correction is necessary as shown below. In the other case: see treatment of pulses **TBU_CH[1]_BASE** in normal mode in chapter *21.9.6.2 "Operation for direction change in normal and emergency mode (DPLL_CTRL_1.SMC=0)"* .

Save inc_cntx value at direction change to inc_cnt1_save.
Calculate the new **DPLL_INC_CNT1.INC_CNT1** value as follows:
1. Clear **DPLL_INC_CNT1.INC_CNT1** .
2. Set **DPLL_INC_CNT1.INC_CNT1** to the target value of the last increment
nmb_t_tar
In contrast to the normal mode, nmb_t_tar is to be used instead of nmb_t_tar_old, because the direction information is known before the calculation takes place.
3. Do not add the calculated number of *TRIGGER* pulses because it is not calculated until the direction change information is known.
4. Subtract the value of still not sent pulses (remaining value at inc_cnt1_save)
– inc_cnt1_save
5. Add the new calculated target pulses for the current increment
+ nmb_t_tar_new
when for the calculation all new conditions of PD_S_store and DPLL_NUSC_SYN_S are considered.
inc_cnt1 = nmb_t_tar_old – inc_cnt1_save + nmb_t_tar_new
All pulses summarized at **DPLL_INC_CNT1.INC_CNT1** are sent out by the maximum possible frequency, because no speed information is available for the first increment after changing the direction. Please note that no pulse correction using **DPLL_CTRL_1_SHADOW_STATE.PCM1** is possible during direction change.
Suppress changing of **DPLL_PSTC.PSTC** for the *TRIGGER* event when a direction change is detected.
Store the new calculated value nmb_t_tar_new at nmb_t_tar for the correct calculation of **DPLL_PSTC.PSTC** at the next input event.

repeated change to forward direction for *TRIGGER*
The *DIR1* bit remains set as long as the *TDIR* bit is set for the following *TRIGGER* events and is reset when for an active *TRIGGER* slope the *TDIR* is zero.

Resetting the *DIR1* to 0 results (after repeated reset of **DPLL_STATUS.LOCK1** and **DPLL_STATUS.ITN**) the opposite correction of the address pointer use.

This means there are two increment operations of the address pointer including the update of **DPLL_NUTC.SYN_T** and PD_store.

A complex correction of **TBU_CH[1]_BASE.BASE** and **DPLL_INC_CNT1.INC_CNT1** is in the normal case not necessary, when all increments are equal (**DPLL_CTRL_1.SYN_NT** =0) and no adapt information is used. In this case only the **DPLL_MLS1.MLS1** value is added to **DPLL_INC_CNT1.INC_CNT1** in order to back count the value for the last increment. In the other case: see treatment of pulses **TBU_CH[1]_BASE.BASE** and **DPLL_INC_CNT1.INC_CNT1** in normal mode in chapter *21.9.6.2 "Operation for direction change in normal and emergency mode (DPLL_CTRL_1.SMC=0)"* .

21.9.6.4 Operation for direction change for STATE signal (DPLL_CTRL_1.SMC=1)

When for **DPLL_CTRL_1.SMC** =1 a backward condition is detected for the *STATE* input signal (*SDIR* =1, resulting in *DIR2* =1), the **DPLL_STATUS.LOCK2** bit is reset, the DPLL_NUSC_NUSE value is set to 1 and the address pointers DPLL_APS_APS and DPLL_APS_1C3_1C3 and DPLL_APS_1C2 are decremented for each active slope of *STATE* as long as the *DIR2* bit shows the backward direction.

Note:

in case of the change of the direction the **DPLL_STATUS.ISN** bit is reset.

For this transition to the backward direction no change of address pointer DPLL_APS_APS and DPLL_APS_1C2 is necessary.

profile update for *STATE*
Make the same update steps for the profile address pointer as shown in chapter *21.9.6.2 "Operation for direction change in normal and*

emergency mode (DPLL_CTRL_1.SMC=0)" : Decrement DPLL_APS_1C3_1C3 for 2 times with the update of the DPLL_NUSC_SYN_S, DPLL_NUSC_SYN_S_OLD, PD_S_store and PD_S_store_old values at each step:

- decrement DPLL_APS_1C3_1C3, load DPLL_NUSC_SYN_S, PD_S_store, update DPLL_NUSC_SYN_S_OLD
- decrement DPLL_APS_1C3_1C3, make calculations, load DPLL_NUSC_SYN_S and PD_S_store, update DPLL_NUSC_SYN_S_OLD and PD_S_store_old and wait for a new *STATE* event.

A complex correction of **TBU_CH[2]_BASE.BASE** and **DPLL_INC_CNT2.INC_CNT2** is in the normal case not necessary, when all increments are equal (DPLL_SYN_NS=0) and no adapt information is used. In this case only the **DPLL_MLS2.MLS2** value is added to **DPLL_INC_CNT2.INC_CNT2** in order to back count the value for the last increment. In the other case: see treatment of pulses **TBU_CH[1]_BASE.BASE** and **DPLL_INC_CNT1.INC_CNT1** in normal mode at chapter *21.9.6.2 "Operation for direction change in normal and emergency mode (DPLL_CTRL_1.SMC=0)"* .

For the second PMSM the pulses are corrected as follows:
Save **DPLL_INC_CNT2.INC_CNT2** value at direction change to inc_cnt2_save.
Calculate the new **DPLL_INC_CNT2.INC_CNT2** value as follows:
1. Clear **DPLL_INC_CNT2.INC_CNT2** .
2. Set **DPLL_INC_CNT2.INC_CNT2** to the target value of the last increment
nmb_s_tar
In contrast to the normal mode, nmb_s_tar is to be used instead of nmb_s_tar_old, because no new calculation is performed so far.
3. Do not add the calculated number of STATE pulses because it is not calculated until the direction change information is known.
4. Subtract the value of the pulses not sent yet (remaining value at inc_cnt2_save)
– inc_cnt2_save
5. Add the new calculated target pulses for the current increment
+ nmb_s_tar_new
when for the calculation all new conditions of PD_S_store and DPLL_NUSC_SYN_S are considered.
inc_cnt2 = nmb_s_tar_old – inc_cnt2_save + nmb_s_tar_new
All pulses summarized at **DPLL_INC_CNT2.INC_CNT2** are sent out by the maximum possible frequency, because no speed information is available for the first increment after changing the direction. Please note that no pulse correction using **DPLL_CTRL_1.PCM2** is possible during direction change.
Do not change **DPLL_PSSC.PSSC** for a *STATE* event when a change in direction is detected.
Store the new calculated value nmb_s_tar_new at nmb_s_tar for the correct calculation of **DPLL_PSTC.PSTC** at the next input event.

repeated change to forward direction for *STATE*
The *DIR2* bit remains set as long as the *SDIR* bit is set for the following *STATE* events and is reset when for an active *STATE* slope *SDIR* is zero.

Resetting the *DIR2* to 0 results (after repeated reset of **DPLL_STATUS.LOCK2** and **DPLL_STATUS.FSD**) in the opposite correction of the address pointer use.

After a last decrementing of all address pointers the DPLL_APS_1C3_1C3 is incremented 2 times with a repeated update of DPLL_NUSC_SYN_S, DPLL_NUSC_SYN_S_OLD and PD_S_store after each increment.

21.9.6.5 DPLL reaction in the case of non plausible input signals

When the DPLL is synchronized concerning the *TRIGGER* signal by setting the **DPLL_STATUS.FTD** , **DPLL_STATUS.SYT** and **DPLL_STATUS.LOCK1** bits, the number of active *TRIGGER* events between the gaps is to be checked continuously.

When additional events occur while a gap is expected, the **DPLL_STATUS.LOCK1** bit is reset and the **DPLL_STATUS.ITN** bit is set.

When an unexpected gap appears (missing *TRIGGER* S), the **DPLL_NUTC.NUTE** value is set to 1, the **DPLL_STATUS.LOCK1** bit is reset and the **DPLL_STATUS.ITN** bit is set. The address pointers are incremented with the next active *TRIGGER* slope accordingly.

The **DPLL_STATUS.TOR** bit is set, when the time from the last valid *TRIGGER* event to the next active *TRIGGER* slope exceeds the value of the last nominal *TRIGGER* duration multiplied by the value of the **DPLL_TLR** register (see chapter **DPLL_TLR**). In this case also the **DPLL_IRQ_NOTIFY.TORI** interrupt is generated, when enabled.

When in the following the direction *DIR1* changes as described in the chapters above the **DPLL_STATUS.ITN** bit is reset, the use of the address pointers **DPLL_APT_2C.APT_2C** is switched and the pulse correction takes place as described above.

In all other cases the CPU can interact to leave the instable state. This can be done by setting the **DPLL_APT_2C.APT_2C** address pointer which results in a reset of the **DPLL_STATUS.ITN** bit. In the following **DPLL_NUTC.NUTE** can also be set to higher values.

When the DPLL is synchronized with regard to the *STATE* signal by setting the **DPLL_STATUS.FSD** , **DPLL_STATUS.SYS** and **DPLL_STATUS.LOCK1** (for **DPLL_CTRL_1.SMC** =0) or **DPLL_STATUS.LOCK2** (for **DPLL_CTRL_1.SMC** =1) bits, the number of active *STATE* events between the gaps is to be checked continuously.

When additional events appear while a gap is expected or while an unexpected missing *STATE* event appears, the **DPLL_STATUS.LOCK1** and **DPLL_STATUS.LOCK2** bit is reset and the **DPLL_STATUS.ISN** bit is set.

When an unexpected gap appears for **DPLL_CTRL_0.RMO** = **DPLL_CTRL_1.SMC** =1 (missing *STATE* signals for synchronous motor control), the DPLL_NUSC_NUSE value is set to 1, the **DPLL_STATUS.LOCK2** bit is reset and the **DPLL_STATUS.ISN** bit is set. The address pointers are incremented with the next active *STATE* slope accordingly.

When the *STATE* locking range **DPLL_SLR.SLR** is violated, the state machine 2 will remain in state 21 and the address pointer DPLL_APS_APS, DPLL_APS_1C2 and DPLL_APS_1C3_1C3 will remain unchanged until the CPU sets the DPLL_APS_1C3_1C3 accordingly. In this case also the DPLL_NUSC_NUSE value is set to 1. The DPLL stops the generation of the *SUB_INC1* / *SUB_INC2* pulses respectively and will perform no other Actions – remaining in step21 of the second state machine (see table in chapter *51 "State description of the State Machine Table"*).
The *STATE* locking range **DPLL_SLR.SLR** is violated when the time from the last valid *STATE* event to the next active *STATE* slope exceeds the value of the last nominal *STATE* duration multiplied with the value of the **DPLL_SLR** register (see chapter **DPLL_SLR**). In this case the **DPLL_STATUS.SOR** bit is set and the **DPLL_IRQ_NOTIFY.SORI** interrupt is generated, when enabled.

When in the following, the direction *DIR2* changes as described in the chapters above the **DPLL_STATUS.ISN** bit is reset, the use of the address pointers DPLL_APS_1C3_1C3 is switched and the pulse correction takes place as described above. In all other cases the CPU must interact to leave the instable state. This can be done by setting the DPLL_APS_1C3_1C3 address pointers which results in a reset of the **DPLL_STATUS.ISN** bit. In the following DPLL_NUSC_NUSE can also be set to higher values.

21.9.6.6 State description of the State Machine.

Table 51 State description of the State Machine Table

Step	Description	Comments

always
for
DPLL_CTRL_1.DEN =1
	for each inactive *TRIGGER* slope with **DPLL_CTRL_0.TEN** =1:
	check, if the last active *TRIGGER* slope was passing the **DPLL_PVT.PVT** check; only in this case perform the following tasks:
	calculate the time stamp difference Δ T to the last active event, store this value at **DPLL_THVAL2.THVAL** . This value is stored to **DPLL_THVAL.THVAL** at the upcoming active slope;
	when 0xFFFF > **DPLL_THMI.THMI** >0 is violated (Δ T < **DPLL_THMI.THMI**):
	generate **DPLL_IRQ_NOTIFY.TINI** interrupt,
	set *DIR1* =0 (forwards)
	set **DPLL_STATUS.BWD1** =0 (see **DPLL_STATUS** register)
	else (only for 0xFFFF > **DPLL_THMI.THMI** >0):
	set *DIR1* = 1 (backwards);
	set **DPLL_STATUS.BWD1** =1 (see **DPLL_STATUS** register)
	after changing the direction correct the WP pulses sent with wrong direction information and send the pulses for the actual increment in addition to highest possible frequency: WP= **DPLL_NMB_T.NMB_T** – **DPLL_INC_CNT1.INC_CNT1** ;
	correct **DPLL_INC_CNT1.INC_CNT1** by addition of 2*WP before sending the correction pulses;
	generate the **DPLL_IRQ_NOTIFY.TISI** interrupt;
	check **DPLL_THMA.THMA** , when **DPLL_THMA.THMA** is violated, generate the **DPLL_IRQ_NOTIFY.TAXI** interrupt; go to step 1
	for each inactive *STATE* slope with **DPLL_CTRL_0.SEN** =1:
	set *DIR2* = *DIR1*
		for **DPLL_CTRL_1.SMC** =0;
		set *DIR1* always after inc./ decr. the address pointers **DPLL_APT.APT** , APT_x;
		go to step 1;
		stop output of *SUB_INC1* and correct pulses after changing *DIR1* after incr./ decr. of A-PS_x
		set *DIR2* always after incr./decr. the address pointers DPLL_APS_APS, APS_x;
		go to step 1
		Processing of the inactive slope (calculation of ΔT and updating **DPLL_THVAL2.THVAL**) is performed parallel to the processing of the active slope.
		Updating the direction on ports (*DIR1* and *DIR2*) and the respective bit fields (**DPLL_STATUS.BWD1** and **DPLL_STATUS.BWD2**) takes place only after the processing of the previous active slope (after step 11).

always for **DPLL_CTRL_1.DEN** =1 and (**DPLL_CTRL_0.TEN** =1 or **DPLL_CTRL_0.SEN** =1, respectively)	set *DIR1* = **DPLL_STATUS.BWD1** = *TDIR* , set *DIR2* = **DPLL_STATUS.BWD2** = *SDIR* ; for each change of *TDIR* go to step 1 after performing the following calculations: correct **DPLL_INC_CNT1.INC_CNT1** correct the pulses (WP, see above) sent with wrong direction information and send the pulses for the actual increment in addition to highest possible frequency. For each change of *SDIR* go to step 21 after performing the following calculations: update of DPLL_NUSC_SYN_S, PD_S_store according to chapter *21.9.6.2 "Operation for* ▽	for **DPLL_CTRL_1.SMC** =1; set the direction bits always after incr./decr. the corresponding address pointers;

Step	Description	Comments
	△ *direction change in normal and emergency mode (DPLL_CTRL_1.SMC=0)"* correct **DPLL_INC_CNT1.INC_CNT1** and **DPLL_INC_CNT2.INC_CNT2** correct the pulses sent with wrong direction information and send the pulses for the actual increment in addition to highest possible frequency.	
1	When **DPLL_CTRL_1.DEN** = 0 or **DPLL_CTRL_0.TEN** =0: stay in step 1 until **DPLL_CTRL_1.DEN** =1, **DPLL_CTRL_0.TEN** =1 and at least one active *TRIGGER* has been detected (**DPLL_STATUS.FTD** =1); the following steps are always performed (not necessarily in step 1, but also in steps 18 to 20 (when waiting for new **DPLL_ID_PMTR_[n].ID_PMTR** values to be calculated): compare *TRIGGER_S* with **DPLL_CTRL_1.TSL** (active slope); When no active *TRIGGER* slope appears and when TS_T_CHECK time is reached: send missing *TRIGGER* INT, also when a gap is expected according to the profile; set **DPLL_STATUS.MT** =1 (missing *TRIGGER* bit) in the **DPLL_STATUS** register; do not leave the active step, until a valid active *TRIGGER* appears. When an active *TRIGGER* slope appears check **DPLL_PVT.PVT** – when the **DPLL_PVT.PVT** value is violated: generate the **DPLL_IRQ_NOTIFY.PWI** interrupt, ignore the *TRIGGER* input and wait for the next active *TRIGGER* slope (ignore each inactive slope); do not store any value – When the **DPLL_PVT.PVT** value is fulfilled: store the actual position stamp at **DPLL_PSTM.PSTM** (value at the *TRIGGER* event) update the RAM region 2 by equations in chapter *21.8.5 "Update of RAM in Normal and Emergency Mode"* . store the actual **DPLL_INC_CNT1.INC_CNT1** value at MP1 as missing pulses (instead of calculation in step 5) store all relevant configuration bits X of the **DPLL_CTRL_0** and **DPLL_CTRL_1** Registers in shadow registers and consider them for all corresponding calculations of steps 1 to 20 accordingly; the relevant bits are explained in the registers itself generate the **DPLL_IRQ_NOTIFY.TASI** interrupt; for **DPLL_STATUS.FTD** =0: set **DPLL_PSTC.PSTC** = **DPLL_PSTM.PSTM** set **DPLL_STATUS.FTD** (first *TRIGGER* detected) do not change **DPLL_PSTC.PSTC** , **DPLL_APT.APT** , **DPLL_APT.APT_2B** for (**DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0 or **DPLL_CTRL_1.SMC** =1) and **DPLL_CTRL_1_SHADOW_TRIGGER.SGE1** =1: increment **DPLL_INC_CNT1.INC_CNT1** by (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) ▽	Depending on **DPLL_CTRL_1.TSL** , **DPLL_CTRL_0.TEN** , **DPLL_CTRL_1.DEN** step one is leaving with the next *TRIGGER* input; Step 1 is also left in emergency mode when an active *TRIGGER* event appears in order to make a switch back to normal mode possible; _old – values are values valid at the last but one active *TRIGGER* event; for the whole table: use always **DPLL_MLS1.MLS1** instead of (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) for the case **DPLL_CTRL_1.SMC** =1 ; dir_crement means: increment for *DIR1* =0 decrement for *DIR1* =1 replace (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) by **DPLL_MLS1.MLS1** for **DPLL_CTRL_1.SMC** =1 **DPLL_NMB_T_TAR.NMB_T_TAR** is the target value of **DPLL_NMB_T.NMB_T** of the last increment (see step 5 ff.) add **DPLL_MPVAL1.MPVAL1** once to **DPLL_INC_CNT1.INC_CNT1** , that means only when **DPLL_CTRL_1_SHADOW_TRIGGER.PCM1** =1 SGE1_delay is the value of **DPLL_CTRL_1_SHADOW_TRIGGER.SGE1** delayed by one active *TRIGGER* event PD_store = 0 for **DPLL_CTRL_0_SHADOW_TRIGGER.AMT** =0

Step	Description	Comments
	△ + **DPLL_MPVAL1.MPVAL1** send *SUB_INC1* pulses with highest possible frequency when **DPLL_CTRL_1_SHADOW_TRIGGER.SGE1** =1 and **DPLL_CTRL_11.SIP1** = 0. for **DPLL_STATUS.SYT** =0 and **DPLL_STATUS.FTD** =1: dir_crement **DPLL_APT.APT** and **DPLL_APT.APT_2B** by one; dir_crement for SGE1_delay=1: **DPLL_PSTC.PSTC** by **DPLL_NMB_T_TAR.NMB_T_TAR** for (**DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0 or **DPLL_CTRL_1.SMC** =1) and **DPLL_CTRL_1_SHADOW_TRIGGER.SGE1** =1: increment **DPLL_INC_CNT1.INC_CNT1** by (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) + **DPLL_MPVAL1.MPVAL1** for **DPLL_STATUS.SYT** =1 : dir_crement **DPLL_APT.APT** , **DPLL_APT_2C.APT_2C** , dir_crement **DPLL_APT.APT_2B** by **DPLL_NUTC.SYN_T_OLD** dir_crement for SGE1_delay=1 **DPLL_PSTC.PSTC** by **DPLL_NMB_T_TAR.NMB_T_TAR** for (**DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0 or **DPLL_CTRL_1.SMC** =1) and **DPLL_CTRL_1_SHADOW_TRIGGER.SGE1** =1: increment **DPLL_INC_CNT1.INC_CNT1** by **DPLL_NUTC.SYN_T** *((**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) + PD_store) + **DPLL_MPVAL1.MPVAL1** PD_store is 0 for **DPLL_CTRL_0_SHADOW_TRIGGER.AMT** =0 within the **DPLL_STATUS** register: set **DPLL_STATUS.LOCK1** bit accordingly;	
1	When **DPLL_CTRL_1.DEN** = 0 or **DPLL_CTRL_0.TEN** =0: stay in step 1 until **DPLL_CTRL_1.DEN** =1, **DPLL_CTRL_0.TEN** =1 and at least one active *TRIGGER* has been detected (**DPLL_STATUS.FTD** =1); the following steps are performed always (not necessarily in step 1, but also in steps 18 to 20 (when waiting for new PMTR values to be calculated): compare *TRIGGER_S* with **DPLL_CTRL_1.TSL** (active slope); When no active *TRIGGER* slope appears and when TS_T_CHECK time is reached: send missing *TRIGGER* INT, also when a gap is expected according to the profile; set **DPLL_STATUS.MT** =1 (missing *TRIGGER* bit) in the **DPLL_STATUS** register; do not leave the active step, until a valid active *TRIGGER* appears. When an active *TRIGGER* slope appears check **DPLL_PVT.PVT** – when the **DPLL_PVT.PVT** value is violated: generate the **DPLL_IRQ_NOTIFY.PWI** interrupt, ignore the *TRIGGER* input and wait for the next active *TRIGGER* slope (ignore each inactive slope); do not store any value – When the **DPLL_PVT.PVT** value is fulfilled: store the actual position stamp at **DPLL_PSTM.PSTM** (value at the *TRIGGER* event) update the RAM region 2 by equations of chapter *21.8.5 "Update of RAM in Normal* ▽	Depending on **DPLL_CTRL_1.TSL** , **DPLL_CTRL_0.TEN** , **DPLL_CTRL_1.DEN** step one is leaving with the next *TRIGGER* input; Step 1 is left in emergency mode as well when an active *TRIGGER* event appears in order to make a switch back to normal mode possible; _old – values are values valid at the last but one active *TRIGGER* event ; for the whole table: use always **DPLL_MLS1.MLS1** instead of (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) for the case **DPLL_CTRL_1.SMC** =1 ; dir_crement means: increment for *DIR1* =0 decrement for *DIR1* =1 replace (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) by **DPLL_MLS1.MLS1** for **DPLL_CTRL_1.SMC** =1 **DPLL_NMB_T_TAR.NMB_T_TAR** is the target value of **DPLL_NMB_T.NMB_T** of the last increment (see step 5 ff.) add **DPLL_MPVAL1.MPVAL1** once to **DPLL_INC_CNT1.INC_CNT1** , that means only when **DPLL_CTRL_1_SHADOW_TRIGGER.PCM1** =1 SGE1_delay is the value of **DPLL_CTRL_1_SHADOW_TRIGGER.SGE1** delayed by one active *TRIGGER* event PD_store = 0 for **DPLL_CTRL_0_SHADOW_TRIGGER.AMT** =0

Step	Description	Comments
	△ *and Emergency Mode"* including subchapters *21.8.5.1 "Update the time stamp values for T-RIGGER signal "* , *21.8.5.2 "Extend the time stamp values for TRIGGER signal in forward direction"* , *21.8.5.3 "Extend the time stamp values for trigger signal in backward direction"* , *21.8.5.4 "Update of RAM by DT_T[p] and R-DT_T[p] after calculation"* . Update **DPLL_NUTC.SYN_T** and PD_store; store the actual **DPLL_INC_CNT1.INC_C-NT1** value at MP1 as missing pulses (instead of calculation in step 5) store all relevant configuration bits X of the **DPLL_CTRL_0** and **DPLL_CTRL_1** registers in shadow registers and consider them for all corresponding calculations of steps 2 to 20 accordingly; the relevant bits are explained in the registers itself generate the **DPLL_IRQ_NOTIFY.TASI** inter-rupt; for **DPLL_STATUS.FTD** =0: set **DPLL_PSTC.PSTC** = **DPLL_PSTM.PS-TM** set **DPLL_STATUS.FTD** (first *TRIGGER* de-tected) do not change **DPLL_PSTC.PSTC** , **DPLL_A-PT.APT** , **DPLL_APT.APT_2B** for (**DPLL_CTRL_0_SHADOW_TRIGGER.-RMO** =0 or **DPLL_CTRL_1.SMC** =1) and **D-PLL_CTRL_1_SHADOW_TRIGGER.SGE1** =1: increment **DPLL_INC_CNT1.INC_CNT1** by (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) + **DPLL_MPVAL1.MPVAL1** send *SUB_INC1* pulses with highest possible frequency when **DPLL_CTRL_1_SHADOW_T-RIGGER.SGE1** =1 for **DPLL_STATUS.SYT** =0 and **DPLL_STATU-S.FTD** =1: dir_crement **DPLL_APT.APT** and **DPLL_A-PT.APT_2B** by one; dir_crement for SGE1_delay=1: **DPLL_PST-C.PSTC** by **DPLL_NMB_T_TAR.NMB_T_TAR** for (**DPLL_CTRL_0_SHADOW_TRIGGER.-RMO** =0 or **DPLL_CTRL_1.SMC** =1) and **D-PLL_CTRL_1_SHADOW_TRIGGER.SGE1** =1: increment **DPLL_INC_CNT1.INC_CNT1** by (**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) + **DPLL_MPVAL1.MPVAL1** for **DPLL_STATUS.SYT** =1 : dir_crement **DPLL_APT.APT** , **DPLL_APT_-2C.APT_2C** , dir_crement **DPLL_APT.APT_-2B** by **DPLL_NUTC.SYN_T_OLD** dir_crement for SGE1_delay=1 **DPLL_PSTC.-PSTC** by **DPLL_NMB_T_TAR.NMB_T_TAR** for (**DPLL_CTRL_0_SHADOW_TRIGGER.-RMO** =0 or **DPLL_CTRL_1.SMC** =1) and **D-PLL_CTRL_1_SHADOW_TRIGGER.SGE1** =1: increment **DPLL_INC_CNT1.INC_CNT1** by **D-PLL_NUTC.SYN_T** *((**DPLL_CTRL_0_SHADO-W_TRIGGER.MLT** +1) + PD_store) + **DPLL_-MPVAL1.MPVAL1** PD_store is 0 for **DPLL_CTRL_0_SHADOW_T-RIGGER.AMT** =0 within the **DPLL_STATUS** register: set **DPLL_STATUS.LOCK1** bit accordingly;	

Step	Description	Comments
2	calculate TS_T according to equations *21.7.2.1 "Calculate trigger time stamps"* ; calculate **DPLL_DT_T_ACT.DT_T_ACT** = TS_T – TS_T_OLD calculate **DPLL_RDT_T_ACT.RDT_T_ACT** calculate QDT_TX according to equation in chapter *21.7.2.4 "Calculate QDT_T_ACT"* (forward) and chapter *21.7.4.1 "Calculate Q-DT_T_ACT backwards"* (backward).	
3	send **DPLL_IRQ_NOTIFY.CDTI** interrupt when **DPLL_NTI_CNT.NTI_CNT** is zero or decrement **DPLL_NTI_CNT.NTI_CNT** when not zero; calculate **DPLL_EDT_T.EDT_T** and **DPLL_-MEDT_T.MEDT_T** according to equations *21.7.2.5 "Calculate the error of last prediction"* and *21.7.2.6 "Calculate the weighted average error"* for (**DPLL_CTRL_0_SHADOW_TRIGGER.-RMO** =1 and **DPLL_CTRL_1.SMC** =0): Wait until a new active event occurs and go then back to step 1.	
4	calculate **DPLL_CDT_TX.CDT_TX** according to equation *[eq_17]* and equation *[eq_24]* ;	for **DPLL_CTRL_0_SHADOW_TRIGGER.RM-O** =0 or **DPLL_CTRL_1.SMC** =1;

5	calculate missing pulses:
	MP1 = **DPLL_INC_CNT1.INC_CNT1** (at the moment of an active *TRIGGER* slope)
	calculate target pulses:
	DPLL_NMB_T_TAR.NMB_T_TAR = ((**DPLL_-CTRL_0_SHADOW_TRIGGER.MLT** +1) *+ P-D_store) * **DPLL_NUTC.SYN_T** + **DPLL_MPV-AL1.MPVAL1**
	(instead of PD_store use zero in the case **D-PLL_CTRL_0_SHADOW_TRIGGER.AMT** =0)
		for **DPLL_CTRL_0_SHADOW_TRIGGER.RM-O** =0 or **DPLL_CTRL_1.SMC** =1;
		replace (**DPLL_CTRL_0_SHADOW_TRIGG-ER.MLT** +1) by **DPLL_MLS1.MLS1** for **DPL-L_CTRL_1.SMC** =1;
		add **DPLL_MPVAL1.MPVAL1** only for **DPL-L_CTRL_1_SHADOW_TRIGGER.PCM1** =1 and reset **DPLL_CTRL_1.PCM1** and **DPLL_CTR-L_1_SHADOW_TRIGGER.PCM1** after that;

6	sent MP with highest possible frequency and set **DPLL_NMB_T.NMB_T** = **DPLL_NMB_T_TAR.-NMB_T_TAR**	for **DPLL_CTRL_0_SHADOW_TRIGGER.RM-O** =0 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1_SHADOW_TRIGGER.DMO** =0 and **DPLL_CTRL_1_SHADOW_TRIGGER.CO-A** =0
7	calculate the number of pulses to be sent **DPLL_NMB_T.NMB_T** = **DPLL_NMB_T_TAR.-NMB_T_TAR** + MP (see equations *21.9.3.1 "Calculate the number of pulses to be sent in normal mode using the automatic end mode condition"* or *[eq_199]* respectively)	for **DPLL_CTRL_0_SHADOW_TRIGGER.RM-O** =0 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1_SHADOW_TRIGGER.DMO** =0 and **DPLL_CTRL_1_SHADOW_TRIGGER.CO-A** =1
8	**DPLL_NMB_T.NMB_T** = **DPLL_NUTC.SYN_-T** * **DPLL_CNT_NUM_1.CNT_NUM_1**	for **DPLL_CTRL_0_SHADOW_TRIGGER.RM-O** =0 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1_SHADOW_TRIGGER.DMO** =1
9	State not used .	
10	calculate **DPLL_ADD_IN_CAL1.ADD_IN_C-AL1** according to equation *[eq_193]* or equation *[eq_194]* and equation *[eq_195]* or equations *[eq_196]* , *[eq_197]* , *[eq_1-98]* and store this value in RAM use **DPLL_A-DD_IN_CAL1.ADD_IN_CAL1** as *ADD_IN* value for the case **DPLL_CTRL_1.DLM1** =0 use **DPLL_ADD_IN_LD1.ADD_IN_LD1** as *A-DD_IN* for the case DLM=1, but do this upda-te immediately (without waiting for this step 10); for **DPLL_CTRL_1_SHADOW_TRIGGER.-DMO** = **DPLL_CTRL_1.DLM1** =0 and *EN_-C1U* =0: reset the storage elements in the *SUB_I-NC1* generator; start sending *SUB_INC1* ;	for **DPLL_CTRL_0_SHADOW_TRIGGER.RM-O** =0 or **DPLL_CTRL_1.SMC** =1 for **DPLL_CTRL_1.DLM1** =0 for **DPLL_CTRL_1.DLM1** =1

Step	Description	Comments
11	calculate TS_T_CHECK = TS_T + **DPLL_DT_T_ACT.DT_T_ACT** *(**DPLL_TOV**) ;	for **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =0 or **DPLL_CTRL_1.SMC** =1;
12	automatic setting of Actions masking bits in the **DPLL_STATUS** register: for **DPLL_CTRL_1.SMC** =0: set **DPLL_STATUS.CAIP1** = **DPLL_STATUS.CAIP2** =1 for **DPLL_CTRL_1.SMC** =1: set only **DPLL_STATUS.CAIP1** =1	steps 12 to 16 are not valid for the combination: (**DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** =1)
13	for all corresponding Actions with **DPLL_ACT_STA.ACT_N[n]** =1 calculate: **DPLL_NA[n]** = (**DPLL_PSA[n].PSA** – **DPLL_PSTC.PSTC**)/(**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) for forward direction with w= integer part and b = remainder of the division (fractional part); for backward direction use **DPLL_NA[n]** = (**DPLL_PSTC.PSTC** – **DPLL_PSA[n].PSA**)/(**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) and consider in both cases the time base overflow in order to get a positive difference	Actions 0...(NOAC/2–1) for **DPLL_CTRL_1.SMC** =1 Actions 0...(NOAC–1) for **DPLL_CTRL_1.SMC** =0 depending on **DPLL_ACT_STA.ACT_N[n]** in **DPLL_ACT_STA** register; replace **DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1 by **DPLL_MLS1.MLS1** for **DPLL_CTRL_1.SMC** =1
14	calculate PDT_T[n] and **DPLL_DTA[n].DTA** for up to NOAC Action values according to equation *21.8.1.1 "For DPLL_NUTC.NUTE -- DPLL_NUTC.VTN > NA[n] (part w)"* and equation *[eq_87]* ;	Actions 0...(NOAC/2–1) for **DPLL_CTRL_1.SMC** =1 Actions 0...(NOAC–1) for **DPLL_CTRL_1.SMC** =0
15	calculate **DPLL_TSAC[n].TSAC** according to equation in chapter *[eq_159]* and **DPLL_PSAC[n].PSAC** according to equations in chapter *21.8.6.2 "Calculate the position stamp forwards"* or chapter *21.8.6.3 "Calculate the position stamp backwards"* .	Actions 0...(NOAC/2–1) for **DPLL_CTRL_1.SMC** =1 Actions 0...(NOAC–1) for **DPLL_CTRL_1.SMC** =0
16	automatic resetting of Actions masking bits in the **DPLL_STATUS** register: for **DPLL_CTRL_1.SMC** =0: set **DPLL_STATUS.CAIP1** = **DPLL_STATUS.CAIP2** =0 for **DPLL_CTRL_1.SMC** =1: set only **DPLL_STATUS.CAIP1** =0; set the corresponding **DPLL_ACT_STA.ACT_N[n]** bits in the **DPLL_ACT_STA** register	Set **DPLL_ACT_STA.ACT_N[n]** for all enabled Actions concerned: 0...(NOAC/2–1) for **DPLL_CTRL_1.SMC** =1 0...(NOAC–1) for **DPLL_CTRL_1.SMC** =0
17	check the relation of the last increment to its predecessor according to the profile and taking into account **DPLL_TOV** : set the **DPLL_STATUS.ITN** status bit and reset the corresponding **DPLL_STATUS.LOCK1** or **DPLL_STATUS.LOCK2** bit, when not plausible; go to step 18, when no active *TRIGGER* appears for all following steps 18 to 20: go immediately back to step 1, when an active *TRIGGER* event occurs, interrupt all calculations there and reset all **DPLL_STATUS.CAIP2** or **DPLL_STATUS.CAIP1** in that case; when going back to step 1: store **DPLL_TS_T** in RAM 2b according to **DPLL_APT.APT_2B** ; update RAM 2a and RAM 2d	for all conditions
18	wait for a new **DPLL_ID_PMTR_[n].ID_PMTR** value; set the corresponding **DPLL_STATUS.CAIP1** and **DPLL_STATUS.CAIP2** values and go to step 19 in that case	go immediately to step 1 and update the RAM according to step 17 when an active *TRIGGER* event occurs
19	make the requested Action calculation according to new **DPLL_ID_PMTR_[n].ID_PMTR** values	go immediately to step 1 and update the RAM according to step 17 when an active *TRIGGER* event occurs

Step	Description	Comments
20	reset **DPLL_STATUS.CAIP1** and **DPLL_STATUS.CAIP2** and go back to step 18	go immediately to step 1 and update the RAM according to step 17 when an active *TRIGGER* event occurs

21	When **DPLL_CTRL_1.DEN** = 0 or **DPLL_CTRL_0.SEN** =0:
	make sure that the first active slope of *STATE* is detected;
	stay in step 1 until **DPLL_CTRL_1.DEN** =1, **DPLL_CTRL_0.SEN** =1 and at least one active *STATE* has been detected (**DPLL_STATUS.FSD** =1);
	
	the following steps are always performed (not necessarily in step 21, but also in steps 38 to 40 (when waiting for new **DPLL_ID_PMTR_[n].ID_PMTR** values to be calculated):
	compare *STATE_S* with **DPLL_CTRL_1.SSL** (active slope); for each inactive slope: generate a **DPLL_IRQ_NOTIFY.SISI** interrupt;
	send missing *STATE* INT when **DPLL_TS_S** _CHECK time is reached and set **DPLL_STATUS.MS** =1 (missing *STATE* bits) in that case; do not leave step 21 while no active *STATE* appears.
	When an active *STATE* slope appears:
	store the actual position stamp at **DPLL_PSSM.PSSM** (value at the *STATE* event);
	update RAM by equations of chapter *21.8.5 "Update of RAM in Normal and Emergency Mode"* , especially subchapters *21.8.5.5 "Update the time stamp values for STATE signal"* , *21.8.5.6 "Extend the time stamp values for STATE signal"* , *21.8.5.7 "Extend the time stamp values for STATE signal for backward direction"* , *21.8.5.8 "Update of RAM by DT_S[p] and RDT_S[p] after calculation"* ;
	Update DPLL_NUSC_SYN_S and PD_S_store using the current **DPLL_ADT_S[p]** values.
	store the actual **DPLL_INC_CNT1.INC_CNT1** / **DPLL_INC_CNT2.INC_CNT2** at MP1/MP2 respectively as missing pulses (instead of calculations in step 25)
	store all relevant configuration bits x of the **DPLL_CTRL_0** and **DPLL_CTRL_1** Registers in shadow registers and consider them for all corresponding calculations of steps 21 to 37 accordingly; the relevant bits are explained in the registers itself
	DPLL_CTRL_1.PCM2 is cleared to zero after it is written to **DPLL_CTRL_1_SHADOW_STATE** . If **DPLL_CTRL_0.RMO** = 1 and **DPLL_CTRL_1.SMC** = 0, **DPLL_CTRL_1.PCM1** is cleared to zero after it is written to **DPLL_CTRL_1_SHADOW_STATE** .
	for **DPLL_STATUS.FSD** =0:
	set **DPLL_PSSC.PSSC** = **DPLL_PSSM.PSSM**
	set **DPLL_STATUS.FSD** (first *STATE* detected)
	do not increment **DPLL_PSSC.PSSC**
	for (**DPLL_CTRL_0_SHADOW_STATE.RMO** =1 and **DPLL_CTRL_1.SMC** =0) and **DPLL_CTRL_1_SHADOW_STATE.SGE1** =1: increment **DPLL_INC_CNT1.INC_CNT1** by **DPLL_MLS1.MLS1** + **DPLL_MPVAL1.MPVAL1**
	for (**DPLL_CTRL_0_SHADOW_STATE.RMO** =1 and **DPLL_CTRL_1.SMC** =1) and **DPLL_CTRL_1_SHADOW_STATE.SGE2** =1: incre- ▽
		Depending on **DPLL_CTRL_1.SSL** , **DPLL_CTRL_0.SEN** , **DPLL_CTRL_1.DEN** step 21 is leaving with the next *STATE* input;
		for the steps 22–37: for **DPLL_CTRL_1.SMC** =1 replace:
		DPLL_MLS1.MLS1 by **DPLL_MLS2.MLS2** , **DPLL_STATUS.LOCK1** by **DPLL_STATUS.LOCK2** ;
		SUB_INC1 by *SUB_INC2* ;
		DPLL_CNT_NUM_1.CNT_NUM_1 by **DPLL_CNT_NUM_2.CNT_NUM_2** ;
		DPLL_MPVAL1.MPVAL1 by **DPLL_MPVAL2.MPVAL2** ;
		EN_C1U by *EN_C2U* ;
		dir_crement means:
		increment for *DIR2* =0
		decrement for *DIR2* =1
		or *DIR1* respectively
		target number of pulses of the last increment (see step 25 ff.)
		add **DPLL_MPVAL1.MPVAL1** or **DPLL_MPVAL2.MPVAL2** only once, that means as long as **DPLL_CTRL_1_SHADOW_STATE.PCM1** or **DPLL_CTRL_1_SHADOW_STATE.PCM2** is set respectively
		SGE1_delay is the value of **DPLL_CTRL_1_SHADOW_STATE.SGE1** delayed by one active *STATE* event
		SGE2_delay is the value of **DPLL_CTRL_1_SHADOW_STATE.SGE2** delayed by one active *STATE* event PD_S_store = 0 for **DPLL_CTRL_0_SHADOW_STATE.AMS** =0

Step	Description	Comments

	△
	ment **DPLL_INC_CNT2.INC_CNT2** by **DPLL_MLS2.MLS2** + **DPLL_MPVAL2.MPVAL2**
	for **DPLL_STATUS.SYS** =0, **DPLL_STATUS.FSD** =1:
	dir_crement **DPLL_PSSC.PSSC** by **DPLL_NMB_S_TAR.NMB_S_TAR** for (**DPLL_CTRL_1.SMC** =0 and SGE1_delay=1) or (**DPLL_CTRL_1.SMC** =1 and SGE2_delay=1)
	increment **DPLL_INC_CNT1.INC_CNT1** by **DPLL_MLS1.MLS1** + **DPLL_MPVAL1.MPVAL1** (for **DPLL_CTRL_1.SMC** =0, **DPLL_CTRL_1_SHADOW_STATE.SGE1** =1 and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1);
	increment **DPLL_INC_CNT2.INC_CNT2** by **DPLL_MLS2.MLS2** + **DPLL_MPVAL2.MPVAL2** (for **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1_SHADOW_STATE.SGE2** =1 and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1);
	dir_crement DPLL_APS_APS and DPLL_APS_1C2
	for **DPLL_STATUS.SYS** =1 :
	dir_crement DPLL_APS_APS and DPLL_APS_1C3_1C3
	dir_crement DPLL_APS_1C2 by DPLL_NUSC_SYN_S_OLD
	for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 and **DPLL_CTRL_1.SMC** =0: for SGE1_delay=1 dir_crement **DPLL_PSSC.PSSC** by **DPLL_NMB_S_TAR.NMB_S_TAR** ; for **DPLL_CTRL_1_SHADOW_STATE.SGE1** =1 increment **DPLL_INC_CNT1.INC_CNT1** by DPLL_NUSC_SYN_S*(**DPLL_MLS1.MLS1** + PD_S_store) + **DPLL_MPVAL1.MPVAL1**
	for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 and **DPLL_CTRL_1.SMC** =1: for SGE2_delay=1 dir_crement **DPLL_PSSC.PSSC** by **DPLL_NMB_S_TAR.NMB_S_TAR** ; for **DPLL_CTRL_1_SHADOW_STATE.SGE2** =1 increment **DPLL_INC_CNT2.INC_CNT2** by DPLL_NUSC_SYN_S*(**DPLL_MLS2.MLS2** + PD_S_store) + **DPLL_MPVAL2.MPVAL2**
	within the **DPLL_STATUS** register:
	set **DPLL_STATUS.LOCK1** or **DPLL_STATUS.LOCK2** bit accordingly;	

22	calculate TS_S according to equations in chapter *21.7.3.1 "Calculate STATE signal time stamps"* ; calculate **DPLL_DT_S_ACT.DT_S_ACT** = TS_S – TS_S_OLD calculate **DPLL_RDT_S_ACT.RDT_S_ACT** calculate QDT_SX	
23	send **DPLL_IRQ_NOTIFY.CDSI** interrupt; calculate **DPLL_EDT_S.EDT_S** and **DPLL_MEDT_S.MEDT_S** according to equations *[eq_37]* and *[eq_40]* for **DPLL_CTRL_0_SHADOW_STATE.RMO** =0: Wait until a new active event occurs and go then back to step 21 for **DPLL_CTRL_0_SHADOW_STATE.RMO** =0;	
24	calculate **DPLL_CDT_SX.CDT_SX** according to equation *[eq_41]* and equation *[eq_47]* ;	only for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1

Step	Description	Comments
25	calculate missing pulses – for TBU Channel 1: MP1 = **DPLL_INC_CNT1.INC_CNT1** (active *STATE* slope) – for TBU Channel 2: MP2 = **DPLL_INC_CNT2.INC_CNT2** (active *STATE* slope) calculate target number of pulses: **DPLL_NMB_S_TAR.NMB_S_TAR** = (**DPLL_MLS1.MLS1** + PD_S_store)*DPLL_NUSC_SYN_S + PD_S_store + **DPLL_MPVAL1.MPVAL1** (for **DPLL_CTRL_1.SMC** =0) **DPLL_NMB_S_TAR.NMB_S_TAR** = **DPLL_MLS2.MLS2** * (DPLL_NUSC_SYN_S + PD_S_store) + **DPLL_MPVAL2.MPVAL2** (for **DPLL_CTRL_1.SMC** =1) (instead of PD_S_store use zero in the case **DPLL_CTRL_0_SHADOW_STATE.AMS** =0)	only for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 for **DPLL_CTRL_1.SMC** =0 instead of **DPLL_MPVAL1.MPVAL1** use zero for **DPLL_CTRL_1_SHADOW_STATE.PCM1** =0 for **DPLL_CTRL_1.SMC** =1 instead of **DPLL_MPVAL2.MPVAL2** use zero for **DPLL_CTRL_1_SHADOW_STATE.PCM2** =0; add **DPLL_MPVAL1.MPVAL1** / **DPLL_MPVAL2.MPVAL2** once to **DPLL_INC_CNT1.INC_CNT1** / **DPLL_INC_CNT2.INC_CNT2** and reset **DPLL_CTRL_1_SHADOW_STATE.PCM1** / **DPLL_CTRL_1_SHADOW_STATE.PCM2** after that
26	sent MPx with highest possible frequency and set **DPLL_NMB_S.NMB_S** = **DPLL_NMB_S_TAR.NMB_S_TAR**	only for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1, **DPLL_CTRL_1_SHADOW_STATE.DMO** =0 and **DPLL_CTRL_1_SHADOW_STATE.COA** =0
27	calculate number of pulses to be sent according to equation *[eq_190]* or **DPLL_NMB_S.NMB_S** = **DPLL_NMB_S_TAR.NMB_S_TAR** + MP1 calculate number of pulses to be sent according to equation *[eq_190]* or **DPLL_NMB_S.NMB_S** = **DPLL_NMB_S_TAR.NMB_S_TAR** + MP2	only for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1, **DPLL_CTRL_1_SHADOW_STATE.DMO** =0 and **DPLL_CTRL_1_SHADOW_STATE.COA** =1
28	**DPLL_NMB_S.NMB_S** = DPLL_NUSC_SYN_S* **DPLL_CNT_NUM_1.CNT_NUM_1** (**DPLL_CTRL_1.SMC** =0) **DPLL_NMB_S.NMB_S** = DPLL_NUSC_SYN_S* **DPLL_CNT_NUM_2.CNT_NUM_2** (**DPLL_CTRL_1.SMC** =1)	only for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1, **DPLL_CTRL_1_SHADOW_STATE.DMO** =1
29	State not used.	
30	calculate **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** (**DPLL_CTRL_1.SMC** =0) according to equation *[eq_196]* and equation *[eq_198]* or calculate **DPLL_ADD_IN_CAL2.ADD_IN_CAL2** with equation *[eq_203]* or equation *[eq_204]* respectively and store this value in RAM use **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** / **DPLL_ADD_IN_CAL2.ADD_IN_CAL2** as *ADD_IN* value for the case **DPLL_CTRL_1.DLM1** / **DPLL_CTRL_1.DLM2** =0 use **DPLL_ADD_IN_LD2.ADD_IN_LD2** as *ADD_IN* for the case **DPLL_CTRL_1.DLM2** =1, but do this update immediately (without waiting for this step 30); for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1, **DPLL_CTRL_1_SHADOW_STATE.DMO** =DLM=0 and *EN_C1U* =0 (*EN_C1U* =0): reset the storage elements in the *SUB_INC1* or *SUB_INC2* generator respectively; start sending *SUB_INC1* / *SUB_INC2* ;	only for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 for **DPLL_CTRL_1.DLM1** =0 for **DPLL_CTRL_1.DLM1** =1
31	calculate TS_S_CHECK = TS_S + DT_S _ACT * (SOV);	only for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1;
32	automatic setting of Actions masking bits in the **DPLL_STATUS** register: **DPLL_STATUS.CAIP1** and **DPLL_STATUS.CAIP2** for **DPLL_CTRL_1.SMC** =0 only **DPLL_STATUS.CAIP2** for **DPLL_CTRL_1.SMC** =1	for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1

Step	Description	Comments
33	for all Actions with **DPLL_ACT_STA.ACT_N[n]** =0 calculate: **DPLL_NA[n]** = (**DPLL_PSA[n].PSA** – **DPLL_PSSC.PSSC**)/ **DPLL_MLS1.MLS1** for forward direction with w = integer part and b = remainder of the division (fractional part) for backward direction use **DPLL_NA[n]** = (**DPLL_PSSC.PSSC** – **DPLL_PSA[n].PSA**)/(**DPLL_MLS1.MLS1**) and consider in both cases the time base overflow in order to get a positive difference use **DPLL_MLS2.MLS2** as divider in the case of **DPLL_CTRL_1.SMC** =1	for **DPLL_CTRL_1.SMC** =0: NOAC Actions, for **DPLL_CTRL_1.SMC** =1: NOAC/2 Actions; depending on **DPLL_ACT_STA.ACT_N[n]** in **DPLL_ACT_STA** register
34	calculate PDT_S[n] and **DPLL_DTA[n].DTA** for up to NOAC Action values according to equations in chapters *21.8.3.1 "For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w)"* , *21.8.3.2 "For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w) and DPLL_NUSC_NUSE= 2*(DPLL_SNU+1)"* , *21.8.3.3 "For DPLL_NUSC_NUSE – DPLL_NUSC_VSN <= DPLL_NA[i] (part w) and DPLL_NUSC_NUSE > 1"* , *21.8.3.4 "For DPLL_NUSC_NUSE = 1"* , *21.8.3.5 "Calculate the duration value for an Action"* in forward direction, chapters *21.8.4.1 "For DPLL_NUSC_NUSE –- DPLL_NUSC_VSN > DPLL_NA[i] (part w)"* , *21.8.4.2 "For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w) and DPLL_NUSC_NUSE= 2*(DPLL_SNU+1)"* , *21.8.4.3 "For DPLL_NUSC_NUSE – DPLL_NUSC_VSN <= DPLL_NA[i] (part w) and DPLL_NUSC_NUSE > 1"* , *21.8.4.4 "For DPLL_NUSC_NUSE = 1"* , *21.8.4.5 "Calculate the duration value until Action"* in backward direction;	only for **DPLL_CTRL_0_SHADOW_STATE.RMO** =1; for **DPLL_CTRL_1.SMC** =0 Actions 0...(NOAC–1) for **DPLL_CTRL_1.SMC** =1 Actions (NOAC/2)...(NOAC–1)
35	calculate **DPLL_TSAC[n].TSAC** according to equations in chapter *21.8.8.1 "Calculate the Action time stamp"* and **DPLL_PSAC[n].PSAC** according to equations in chapters *21.8.8.2 "Calculate the position stamp forwards"* and *21.8.8.3 "Calculate the position stamp backwards"* .	for the relevant Actions (see above) and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1
36	automatic reset of the Actions masking bit in the **DPLL_STATUS** register: **DPLL_STATUS.CAIP1** = **DPLL_STATUS.CAIP2** =0 for **DPLL_CTRL_1.SMC** =0 and only **DPLL_STATUS.CAIP2** =0 for **DPLL_CTRL_1.SMC** =1 set the corresponding **DPLL_ACT_STA.ACT_N[n]** bits in the **DPLL_ACT_STA** register	for the relevant Actions (see above) and **DPLL_CTRL_0_SHADOW_STATE.RMO** =1 Set **DPLL_ACT_STA.ACT_N[n]** and reset ACT_WRn for all enabled Actions
37	check the duration of the last increment to its predecessor according to the profile and taking into account **DPLL_TOV_S** : set the **DPLL_STATUS.ISN** status bit and reset the corresponding **DPLL_STATUS.LOCK1** or **DPLL_STATUS.LOCK2** bit, when not plausible; go to step 38, when no active *STATE* appears for all following steps 38 to 40: go immediately back to step 21, when an active *STATE* event occurs, interrupt all calculations there and reset all **DPLL_STATUS.CAIP1** / **DPLL_STATUS.CAIP2** in that case; when going back to step 21: store **DPLL_TS_S** in RAM 1c2 according to DPLL_APS_1C2; update RAM 1c1 and RAM 1c4	for all conditions

Step	Description	Comments
38	wait for a new **DPLL_ID_PMTR_[n].ID_PMTR** value; set the corresponding **DPLL_STATUS.CAIP1** / **DPLL_STATUS.CAIP2** values and go to step 39 in that case	go immediately to step 21 and update the RAM according to step 37 when an active *STATE* event occurs
39	make the requested Action calculation according to new **DPLL_ID_PMTR_[n].ID_PMTR** values	go immediately to step 21 and update the RAM according to step 37 when an active *STATE* event occurs
40	reset **DPLL_STATUS.CAIP1** and **DPLL_STATUS.CAIP2** and go back to step 38	go immediately to step 21 and update the RAM according to step 37 when an active *STATE* event occurs

21.10 DPLL Interrupt Signals

The DPLL provides 27 interrupt lines. These interrupts are shown in DPLL Port Description (*21.16.1 "DPLL Interrupt ports"*).

Note:

DPLL_TE0_IRQ , *DPLL_TE1_IRQ* , *DPLL_TE2_IRQ* , *DPLL_TE3_IRQ* , *DPLL_TE4_IRQ* depends on the **DPLL_ADT_T[p].TINT** value (see RAM region 2 explanations in chapter *21.14 " DPLL RAM Region 2 value description"*) and is only active when **DPLL_STATUS.SYT** =1. The **DPLL_ADT_T[p].TINT** value is corresponding to the **DPLL_ADT_T[p]** section of RAM region 2.

21.11 DPLL Registers Description

21.11.1 DPLL_CTRL_0

Description	Control Register 0
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_0
Address	$0x8000$
C-Name	GTM.CLS[0].DPLL.CTRL_0

Interface: MCS[i]

Name	DPLL_CTRL_0
Address	$0x8000$
C-Name	

MLT	
Description	Multiplier for TRIGGER.
Loop	-
Bit Range	$[9:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

MLT	
Condition	–
Initial value	$0x257$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	DPLL_CTRL_0_SHADOW_TRIGGER.MLT +1 is number of *SUB_INC1* pulses between two *TRIGGER* events in normal mode (1...1024); **Note:** For emergency mode the number of *SUB_INC1* pulses between two *STATE* events is calculated by the CPU using the formula **DPLL_MLS1.MLS1** =(**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1)* (**DPLL_CTRL_0.TNU** +1) / (DPLL_SNU+1) in order to get the same number of *SUB_INC1* pulses for FULL_SCALE. This value is stored in RAM at 0x05C0. Change of **DPLL_CTRL_0.MLT** by the CPU must result in the corresponding change of **DPLL_MLS1.MLS1** by the CPU for **DPLL_CTRL_1.SMC** =0. **Note:** The number of **DPLL_CTRL_0_SHADOW_TRIGGER.MLT** events is the binary value plus 1. The value **DPLL_CTRL_0.MLT** +1 is replaced by **DPLL_MLS1.MLS1** in the case of **DPLL_CTRL_1.SMC** =1 (see **DPLL_CTRL_1** register) for all relevant calculations.

IFP	
Description	Input filter position.
Loop	–
Bit Range	$[10 : 10]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : **DPLL_FTV_T.TRIGGER_FT** and **DPLL_FTV_S.STATE_FT** mean time related values, that means the number of time stamp clocks 1 : **DPLL_FTV_T.TRIGGER_FT** and **DPLL_FTV_S.STATE_FT** mean position related values, that means the number of *SUB_INC1* (or *SUB_INC2* in the case **DPLL_CTRL_1.SMC** =1) pulses respectively
	Value contains position or time related information (See notes **DPLL_CTRL_0.RMO**).

SNU	
Description	STATE number.
Loop	–
Bit Range	$[15 : 11]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

SNU	
Initial value	$0x17$
Protect Enable Cond	DPLL_CTRL_11.STATE_EXT == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	DPLL_CTRL_0.SNU +1 is number of nominal *STATE* events in HALF_SCALE (1...32) (for explanation see note at **DPLL_CTRL_0.RMO**). **Note:** The number of nominal *STATE* events is the decimal value plus 1. This value can only be written when (**DPLL_CTRL_0.RMO** =0 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.DEN** =0. To make sure that this signal is not changed during a mode change **DPLL_CTRL_0.RMO** =0 means that the status of **DPLL_CTRL_0.RMO** =0 must be given before and during writing to the register. Set **DPLL_CTRL_1.SSL** =00 before changing this value and set **DPLL_CTRL_0.RMO** =1 only after FULL_SCALE with **DPLL_CTRL_1.SSL** >0. **Note:** This register can only be written when **DPLL_CTRL_11.STATE_EXT** is not set. If **DPLL_CTRL_11.STATE_EXT** is set, the signal cannot be written (*AEI_STATUS* =0b10) . If **DPLL_CTRL_11.STATE_EXT** is set the read value is equal to **DPLL_CTRL_EXT.SNU** .

TNU	
Description	TRIGGER number.
Loop	–
Bit Range	$[24 : 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0x3B$
Protect Enable Cond	DPLL_CTRL_1.DEN == 1 || !(DPLL_CTRL_0.RMO == 1 && DPLL_CTRL_1.SMC == 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	DPLL_CTRL_0.TNU +1 is number of nominal *TRIGGER* events in HALF_SCALE (1...512) (for explanation see note at **DPLL_CTRL_0.RMO**). **Note:** The number of nominal *TRIGGER* events is the decimal value plus 1.This value can only be written when (**DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.DEN** =0. To make sure that this bit field is not changed during a mode change, **DPLL_CTRL_0.RMO** =0 must be at zero before and during the writing to the bit field. Set **DPLL_CTRL_1.TSL** =00 before changing this value and set **DPLL_CTRL_0.RMO** =0 only after FULL_SCALE with **DPLL_CTRL_1.TSL** >0.

AMS	
Description	Adapt mode STATE.
Loop	–
Bit Range	$[25 : 25]$
Access Type	RW
Volatile	False

AMS	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : No adaptation information is used for *STATE* 1 : Immediate adapting mode; the values for physical deviation **DPLL_ADT_S[p].PD_S** are considered to calculate *SUB_INC1* pulses in emergency mode (**DPLL_CTRL_1.SMC** =0) or *SUB_INC2* pulses for **DPLL_CTRL_1.SMC** =1
	Use of adaptation information of *STATE* (for explanation see note at **DPLL_CTRL_0.RMO**).

AMT	
Description	Adapt mode TRIGGER.
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : No adaptation information for *TRIGGER* is used 1 : Immediate adapting mode; the values for physical deviation **DPLL_ADT_T[p].PD** are considered to calculate the *SUB_INC1* pulses in normal mode and for **DPLL_CTRL_1.SMC** =1
	Use of adaptation information of *TRIGGER* (for explanation see note at **DPLL_CTRL_0.RMO**).

IDS	
Description	Input delay STATE.
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

IDS	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Delay information is not used 1 : Up to 24 bits of the FT part contain the delay value of the input signal, concerning the corresponding edge
	Use of input delay information transmitted in FT part of the *STATE* signal (for explanation see note at **DPLL_CTRL_0.RMO**).

IDT	
Description	Input delay TRIGGER.
Loop	–
Bit Range	[28 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Delay information is not used 1 : Up to 24 bits of the FT part contain the delay value of the input signal, concerning the corresponding edge
	Use of input delay information transmitted in FT part of the *TRIGGER* signal (for explanation see note at **DPLL_CTRL_0.RMO**).

SEN	
Description	STATE enable.
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : *STATE* signal is not enabled (no signal considered) 1 : *STATE* signal is enabled

TEN	
Description	TRIGGER enable.

TEN	
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : *TRIGGER* signal is not enabled (no signal considered) 1 : *TRIGGER* signal is enabled

RMO	
Description	Reference mode.
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Normal mode; the signal *TRIGGER* is used to generate the *SUB_INC1* signals 1 : Emergency mode when **DPLL_CTRL_1.SMC** =0; signal *STATE* is used to generate the *SUB_INC1* signals ; Double synchronous mode when **DPLL_CTRL_1.SMC** =1: signal *TRIGGER* is used to generate the *SUB_INC1* signals and *STATE* is used to generate the *SUB_INC2* signals
	Selection of the relevant input signal for generation of *SUB_INC1* (for explanation see note at **DPLL_CTRL_0.RMO**). **Note:** For **DPLL_CTRL_1.SMC** =0: *TRIGGER* and *STATE* are prepared to calculate *SUB_INC1* . The **DPLL_CTRL_0.RMO** bit gives a decision only, which of them is used. For changing from normal mode to emergency mode at the following *STATE* slope (according to **DPLL_CTRL_0_SHADOW_STATE.RMO**) the **DPLL_PSSC.PSSC** value is calculated by **DPLL_PSSC.PSSC** = **DPLL_PSSM.PSSM** + correction value (forward direction) or **DPLL_PSSC.PSSC** = **DPLL_PSSM.PSSM** – correction value (backward direction) with the correction value = **DPLL_INC_CNT1.INC_CNT1** – **DPLL_NMB_S.NMB_S** . For changing from emergency mode to normal mode at the following *TRIGGER* slope (according to **DPLL_CTRL_0_SHADOW_TRIGGER.RMO**) the **DPLL_PSTC.PSTC** value is calculated by **DPLL_PSTC.PSTC** = **DPLL_PSTM.PSTM** + correction value (forward direction) or **DPLL_PSTC.PSTC** = **DPLL_PSTM.PSTM** – correction value (backward direction) with the correction value = **DPLL_INC_CNT1.INC_CNT1** – **DPLL_NMB_T.NMB_T** . In case no further *TRIGGER* or *STATE* events the CPU has to perform the above corrections.

Note:
DPLL_CTRL_0.RMO is stored to **DPLL_CTRL_0_SHADOW_STATE.RMO** on encountering an active *STATE* event if **DPLL_CTRL_1.DEN** = 1.

Note:

DPLL_CTRL_0.RMO is stored to **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** on encountering an active *TRIGGER* event if **DPLL_CTRL_1.DEN** = 1.

Note:

The time between two active *STATE* or *TRIGGER* events must always be greater than 23.4 µs; in addition, the *TS_CLK* and the resolution must be chosen such that for each nominal increment the time stamps at the beginning and the end of the increment differ at least in the value of 257.

Note:

For **DPLL_CTRL_0.IFP** =1 the time between two active *TRIGGER* or *STATE* events must always be greater than 2.34 ms and the value x of **DPLL_CTRL_0.MLT** , **DPLL_MLS1.MLS1** or **DPLL_MLS2.MLS2** must be chosen such that the number of time stamp pulses between two *SUB_INC1* and *SUB_INC2* events must be less than 65536. This is fulfilled when x is greater than 256.

21.11.2 DPLL_CTRL_1

Description	Control Register 1
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_1
Address	$0x8004$
C-Name	GTM.CLS[0].DPLL.CTRL_1

Interface: MCS[i]

Name	DPLL_CTRL_1
Address	$0x8004$
C-Name	

DMO	
Description	DPLL mode select.
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Automatic end mode; if the number of pulses for an increment is reached, no further pulse is generated until the next active *TRIGGER* / *STATE* is received; in the case of getting a new active *TRIGGER* / *STATE* before the defined number of pulses is reached, the pulse frequency is changed according to the conditions described below (**DPLL_CTRL_1.COA**). 1 : Continuous mode; in this mode a difference between the predefined number of pulses and the actual number of generated pulses can influence the pulse frequency by writing a corresponding pulse number into **DPLL_CNT_NUM_1.CNT_NUM_1** or **DPLL_CNT_NUM_2.CNT_NUM_2** respectively in RAM region 1b.

DMO	
	For explanation see note at **DPLL_CTRL_1.TSL** .

DEN	
Description	DPLL enable.
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : The DPLL is not enabled; Disabling the DPLL will result in a reset state of the **DPLL_STATUS** register which remains in this state until **DPLL_CTRL_1.DEN** =1. No DPLL related interrupt will be generated in that case. 1 : The DPLL is enabled.
	Note: The bits 31 down to 0 of the **DPLL_STATUS** register are cleared, when the DPLL is disabled. Some bits of the control registers can be set only when **DPLL_CTRL_1.DEN** =0. The protected bits in the **DPLL_CTRL_1** register cannot be written when simultaneously **DPLL_CTRL_1.DEN** is set to 1.

IDDS	
Description	Input direction detection strategy in the case of DPLL_CTRL_1.SMC=0.
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : The input direction is detected comparing the **DPLL_THMI.THMI** value with the duration between active and inactive slope of *TRIGGER* . 1 : The input direction is detected using *TDIR* input signal also in the case **DPLL_CTRL_1.SMC** =0.
	Note: This bit can only be written when the DPLL is disabled and set to zero, when not needed for an implementation. Independent of the value of **DPLL_CTRL_1.IDDS** , the direction information for *TRIGGER* in the case of **DPLL_CTRL_1.SMC** =0 is always considered at the moment when the inactive slope appears.

COA	
Description	Correction strategy in automatic end mode (DPLL_CTRL_1.DMO=0).
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The pulse frequency of the *CCM[0]_CLK_RES* [0:0] will be used to make up for missing pulses from last increment; the output of the calculated new pulses will start after resetting the storage elements in the pulse generation unit. The frequency of *CCM[0]_CLK_RES* [0:0] should not exceed half the frequency of the system clock (see *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*) 1 : Missing pulses of the last increment are distributed evenly to the next increment, calculations are done when the next active input event occurs. The number of missing sub-pulses will be determined by the pulse counter difference between the last two active *TRIGGER* / *STATE* events respectively; the storage elements in the pulse generation unit are not reset before sending new pulses.
	For explanation see note at **DPLL_CTRL_1.TSL** . **Note:** For **DPLL_CTRL_1.SMC** = **DPLL_CTRL_0.RMO** =1: **DPLL_CTRL_1.COA** is used for *SUB_INC1* and *SUB_INC2*

PIT	
Description	Plausibility value DPLL_PVT.PVT to next active TRIGGER is time related.
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The plausibility value is position related (**DPLL_PVT.PVT** contains the number of *SUB_INC1* pulses) 1 : The plausibility value is time related (the **DPLL_PVT.PVT** value is to be multiplied with the duration of the last increment **DPLL_DT_T_ACT.DT_T_ACT** and divided by 1024)
	For explanation see note at **DPLL_CTRL_1.TSL** .

SGE1	
Description	SUB_INC1 generator enable.
Loop	–

SGE1	
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The *SUB_INC1* generator is not enabled 1 : The *SUB_INC1* generator is enabled
	For explanation see note at **DPLL_CTRL_1.TSL** .

DLM1	
Description	Direct Load Mode for SUB_INC1 generation.
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The DPLL uses the calculated **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** value for the *SUB_INC1* generation 1 : The **DPLL_ADD_IN_LD1.ADD_IN_LD1** value is used for the *SUB_INC1* generation and is provided by the CPU; the value remains valid until the CPU writes a new one; the calculated *ADD_IN* values are stored as **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** in the RAM at different locations for normal and emergency mode
	For explanation see note at **DPLL_CTRL_1.TSL** .

PCM1	
Description	Pulse Correction Mode for SUB_INC1 generation.
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

PCM1	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : The DPLL does not use the correction value stored in **DPLL_MPVAL1.MPVAL1** 1 : The DPLL uses the correction value stored in **DPLL_MPVAL1.MPVAL1** in normal and emergency mode
	For explanation see note at **DPLL_CTRL_1.TSL** . PCM1 is cleared to zero after it is written to the relevant shadow register depending on the configuration. PCM1 is written to **DPLL_CTRL_1_SHADOW_TRIGGER** in normal mode (**DPLL_CTRL_0.RMO** = 0 and **DPLL_CTRL_1.SMC** = 0) or synchronous engine mode (**DPLL_CTRL_1.SMC** = 1) at the active *TRIGGER* slope when **DPLL_CTRL_1.DEN** = 1 and **DPLL_CTRL1.SGE1** = 1. PCM1 is written to **DPLL_CTRL_1_SHADOW_STATE** in emergency mode (**DPLL_CTRL_0.RMO** = 1 and **DPLL_CTRL_1.SMC** = 0) at the active *STATE* slope when **DPLL_CTRL_1.DEN** = 1 and **DPLL_CTRL1.SGE1** = 1.

SGE2	
Description	SUB_INC2 generator enable.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : The *SUB_INC2* generator is not enabled. 1 : The *SUB_INC2* generator is enabled
	For explanation see note at **DPLL_CTRL_1.TSL** .

DLM2	
Description	Direct Load Mode for SUB_INC2 generation.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

DLM2	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The DPLL uses the calculated **DPLL_ADD_IN_CAL2.ADD_IN_CAL2** value for the *SUB_INC2* generation. 1 : The **DPLL_ADD_IN_LD2.ADD_IN_LD2** value is used for the *SUB_INC2* generation and is provided by the CPU; the value remains valid until the CPU writes a new one; the calculated *ADD_IN* values are stored as **DPLL_ADD_IN_CAL2.ADD_IN_CAL2** in the RAM at different locations for normal and emergency mode
	For explanation see note at **DPLL_CTRL_1.TSL** .

PCM2	
Description	Pulse Correction Mode for SUB_INC2 generation.
Loop	-
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The DPLL does not use the correction value stored in **DPLL_MPVAL2.MPVAL2** 1 : The DPLL uses the correction value stored in **DPLL_MPVAL2.MPVAL2**
	For explanation see note at **DPLL_CTRL_1.TSL** .

SYN_NS	
Description	Synchronization number of DPLL STATE events. summarized number of virtual increments in Half Scale Mode.
Loop	-
Bit Range	[15 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1) || !(DPLL_CTRL_0.RMO == 0 && DPLL_CTRL_1.SMC == 0) || (DPLL_CTRL_11.STATE_EXT == 1)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

SYN_NS	
	Sum of all systematic missing *STATE* events in HALF_SCALE (for **DPLL_CTRL_1.SYSF** =0) or FULL SCALE (for **DPLL_CTRL_1.SYSF** =1) ; the **DPLL_CTRL_1.SYN_NS** missing *STATE* ports can be divided up to an arbitrary number of blocks. The pattern of events and missing events in FULL_SCALE is shown in RAM region 1c3 as value **DPLL_ADT_S[p].NS** in addition to the adapted values. The number of stored increments in FULL_SCALE must be equal to 2*(**DPLL_CTRL_0.SNU** +1− **DPLL_CTRL_1.SYN_NS**) for **DPLL_CTRL_1.SYSF** =0 or 2*(**DPLL_CTRL_0.SNU** +1)− **DPLL_CTRL_1.SYN_NS** for **DPLL_CTRL_1.SYSF** =1 . This pattern is written by the CPU beginning from a fixed reference point (maybe beginning of the FULL_SCALE region). The relation to the actual increment is established by setting of the profile RAM pointer **DPLL_APS_1C3.APS_1C3** in an appropriate relation to the RAM pointer **DPLL_APS.APS** of the actual increment by the CPU. **Note:** This value can only be written when (**DPLL_CTRL_0.RMO** =0 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.DEN** =0. Set **DPLL_CTRL_1.SSL** =00 before changing this value and set **DPLL_CTRL_0.RMO** =1 only after FULL_SCALE with **DPLL_CTRL_1.SSL** >0. To make sure that this bit field is not changed during a mode change, **DPLL_CTRL_1.SMC** must be at zero before and during the writing to the bit field. **Note:** This register can only be written when **DPLL_CTRL_11.STATE_EXT** is not set. If **DPLL_CTRL_11.STATE_EXT** is set, the signal cannot be written. If **DPLL_CTRL_11.STATE_EXT** is set the read value is equal to **DPLL_CTRL_EXT.SYN_NS** .

SYN_NT	
Description	Synchronization number of DPLL TRIGGER events. summarized number of virtual increments in Half Scale Mode.
Loop	–
Bit Range	[21 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1) || !(DPLL_CTRL_0.RMO == 1 && DPLL_CTRL_1.SMC == 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Sum of all systematic missing *TRIGGER* events in HALF_SCALE; the **DPLL_CTRL_1.SYN_NT** missing *TRIGGER* can be divided up to an arbitrary number of blocks. The pattern of events and missing events in FULL_SCALE is shown in RAM region 2c as value **DPLL_ADT_T[p].NT** in addition to the adapted values. The number of stored increments in FULL_SCALE must be equal to 2*(**DPLL_CTRL_0.TNU** − **DPLL_CTRL_1.SYN_NT**). This pattern is written by the CPU starting with a fixed reference point (maybe beginning of the FULL_SCALE region). The relation to the actual increment is established by setting of the profile RAM pointer **DPLL_APT_2C.APT_2C** in an appropriate relation to the RAM pointer **DPLL_APT.APT** of the actual increment by the CPU. **Note:** This value can only be written when (**DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.DEN** =0. Set **DPLL_CTRL_1.TSL** =00 before changing this value and set **DPLL_CTRL_0.RMO** =0 only after FULL_SCALE with **DPLL_CTRL_1.TSL** >0. To make sure that this bit field is not changed during a mode change, **DPLL_CTRL_1.SMC** must be at zero before and during writing to the register.

LCD	
Description	Locking condition definition
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False

LCD	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Locking condition definition is one times missing *TRIGGER* s as expected by the profile in HALF_SCALE (one gap). 1 : Locking condition definition is n-1 times missing *TRIGGER* s as expected by the profile in HALF_SCALE (one additional tooth)
	Note: This bit can only be written when the DPLL is disabled and be set to zero, when not needed for an implementation.

SWR	
Description	Software reset
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : No software reset enabled 1 : Software reset enabled
R-Coding	0 : No status
	resets all register and internal states of the DPLL **Note:** Setting the **DPLL_CTRL_1.SWR** bit results only in a software reset when the DPLL is not enabled (**DPLL_CTRL_1.DEN** =0).

SYSF	
Description	DPLL_SYN_NS for FULL_SCALE
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

SYSF	
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1) || !(DPLL_CTRL_0.RMO == 0 && DPLL_CTRL_1.SMC == 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The DPLL_SYN_NS value is valid for HALF_SCALE 1 : The DPLL_SYN_NS value is valid for FULL_SCALE
	The value DPLL_SYN_NS means the sum of all systematic missing *STATE* events in HALF_SCALE (for **DPLL_CTRL_1.SYSF** =0) or FULL SCALE (for **DPLL_CTRL_1.SYSF** =1). **Note:** This value can only be written when (**DPLL_CTRL_0.RMO** =0 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.DEN** =0. Set **DPLL_CTRL_1.SSL** =00 before changing this value and set **DPLL_CTRL_0.RMO** =1 only after FULL_SCALE with **DPLL_CTRL_1.SSL** >0. To make sure that this bit field is not changed during a mode change, **DPLL_CTRL_1.SMC** must be at zero before and during writing to the register.

TS0_HRS	
Description	Time stamp high resolution STATE
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The resolution of the used DPLL input *CCM[0]_TBU_TS0* bits is equal to the *STATE* input time stamp resolution 1 : The *STATE* input time stamps have 8 times higher resolution than the *CCM[0]_TBU_TS0* DPLL input
	Note: This bit can only be written when the DPLL is disabled.

TS0_HRT	
Description	Time stamp high resolution TRIGGER
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

TS0_HRT	
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The resolution of the used DPLL input *CCM[0]_TBU_TS0* bits is equal to the *TRIGGER* input time stamp resolution 1 : The *TRIGGER* input time stamps have a 8 times higher resolution than the *CCM[0]_TBU_TS0* input
	Note: This bit can only be written when the DPLL is disabled.

SMC	
Description	Synchronous Motor Control
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : *TRIGGER* and *STATE* inputs are used for a control different to **DPLL_CTRL_1.SMC** . 1 : The *TRIGGER* input reflects a combined sensor signal for **DPLL_CTRL_1.SMC** and in the case of **DPLL_CTRL_0.RMO** =1 also *STATE* reflects a different combined sensor signal
	Note: This bit can only be written when the DPLL is disabled.

SSL	
Description	State slope select
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	3
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1) || !(DPLL_CTRL_0.RMO == 0 && DPLL_CTRL_1.SMC == 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

SSL	
RW-Coding	DPLL_STATUS.FSD == 1 0b00 : No slope of *STATE* will be used; this value makes only sense in normal mode 0b01 : Low high slope will be used as active slope, only inputs with a signal value of "1" will be considered 0b10 : High low slope will be used as active slope, only inputs with a signal value of "0" will be considered 0b11 : Both slopes will be used as active slopes
RW-Coding	DPLL_STATUS.FSD == 0 0b00 : No input signal of *STATE* will be used; this value makes only sense in normal mode 0b01 : "High" input signal level will be used as active slope, only inputs with a signal value of "1" will be considered 0b10 : "Low" input signal level will be used as active slope, only inputs with a signal value of "0" will be considered 0b11 : Both input signal levels will be used as active slopes
	Note: Definition of active slope for signal *STATE* each active slope is an event defined by DPLL_SNU. Set by **DPLL_CTRL_1.DEN** =0 only. "slope sensitive after detection of first *STATE* input signal" "level sensitive for first *STATE* input signal edge" **Note:** This value can only be written when (**DPLL_CTRL_0.RMO** =0 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.DEN** =0. To make sure that this signal is not changed during a mode change, **DPLL_CTRL_1.SMC** must be at zero before and during writing to the register.

TSL	
Description	Trigger slope select
Loop	–
Bit Range	[31 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	2
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1) || !(DPLL_CTRL_0.RMO == 1 && DPLL_CTRL_1.SMC == 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	DPLL_STATUS.FTD == 1 0b00 : No slope of *TRIGGER* will be used; this value makes only sense in emergency mode 0b01 : Low high slope will be used as active slope, only inputs with a signal value of "1" will be considered 0b10 : High low slope will be used as active slope, only inputs with a signal value of "0" will be considered 0b11 : Both slopes will be used as active slopes
RW-Coding	DPLL_STATUS.FTD == 0 0b00 : No input signal of *TRIGGER* will be used; this value makes only sense in normal mode 0b01 : "High" input signal level will be used as active slope, only inputs with a signal value of "1" will be considered 0b10 : "Low" input signal level will be used as active slope, only inputs with a signal value of "0" will be considered 0b11 : Both input signal levels will be used as active slopes

TSL	
	Note: Definition of active slope for signal TRIGGER each active slope is an event defined by DPLL_CTRL_0.TNU. Set by DPLL_CTRL_1.DEN=0 only. "slope sensitive after detection of first *TRIGGER* input signal" "level sensitive for first *TRIGGER* input signal edge" **Note:** This value can only be written when (**DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.-DEN** =0. To make sure that this signal is not changed during a mode change, **DPLL_CTRL_1.SMC** must be at zero before and during writing to the register. **Note:** Stored in an independent shadow register for an active *TRIGGER* event and for **DPLL_CTRL_1.DEN** = 1. **Note:** Stored in an independent shadow register for an active *STATE* event and for **DPLL_CTRL_1.DEN** = 1. **Note:** Bit is cleared, when transmitted to shadow register

21.11.3 DPLL_CTRL_2

Description	Action Enable Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_2
Address	$0x8008$
C-Name	GTM.CLS[0].DPLL.CTRL_2

Interface: MCS[i]

Name	DPLL_CTRL_2
Address	$0x8008$
C-Name	

AEN[n]	
Description	Action [n] enable.
Loop	$n = \{n : 0 <= n <= 7\}$
Bit Range	$[n+8 : n+8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

AEN[n]	
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, n + 16, n + 16) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The corresponding Action [n] is not enabled 1 : The corresponding Action [n] is enabled

WAD[m]	
Description	Write control bit of Action [m].
Loop	$m = \{n : 0 <= n <= 7\}$
Bit Range	$[m + 16 : m + 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : The corresponding **DPLL_CTRL_2.AEN[n]** bit is not writeable (n=m) 1 : The corresponding **DPLL_CTRL_2.AEN[n]** bit is writeable (n=m)

Note:

DPLL_CTRL_2.AEN[n] can be written only if the corresponding **DPLL_CTRL_2.WAD[m]** is set in the same access. It can be set for debug purposes by CPU also, when DPLL is disabled. The enable bit becomes active only when the DPLL is in operation (**DPLL_CTRL_1.DEN** =1).

Note:

For **DPLL_CTRL_2.WAD[m]** =1 only the corresponding **DPLL_CTRL_2.AEN[n]** bits are writable. The **DPLL_CTRL_2.AEN[n]** bits remain unchanged when the corresponding **DPLL_CTRL_2.WAD[m]** =0.

Note:

The bitfields **DPLL_CTRL_2.AEN[n]** , **DPLL_CTRL_3.AEN[n]** , **DPLL_CTRL_4.AEN[n]** and **DPLL_CTRL_5.AEN[n]** construct the signal *AEN[n]* , where *AEN[n]* is constructed out of the concatenation of the bitfields **DPLL_CTRL_2.AEN[n]** , (n∈{0, 1,..., 7}) and **DPLL_CTRL_3.AEN[n]** , (n∈{8, 9,..., 15}) and **DPLL_CTRL_4.AEN[n]** , (n∈{16, 17,..., 23}) and **DPLL_CTRL_5.AEN[n]** , (n∈{24, 25,..., 31}).

21.11.4 DPLL_CTRL_3

Description	Action Enable Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_3
Address	$0x800C$

C-Name	GTM.CLS[0].DPLL.CTRL_3

Interface: MCS[i]

Name	DPLL_CTRL_3
Address	$0x800C$
C-Name	

AEN[n]	
Description	Action [n] enable.
Loop	$n = \{n : 8 <= n <= 15\}$
Bit Range	$[n : n]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, n + 8, n + 8) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The corresponding Action [n] is not enabled 1 : The corresponding Action [n] is enabled

WAD[m]	
Description	Write control bit of Action [m].
Loop	$m = \{n : 8 <= n <= 15\}$
Bit Range	$[m + 8 : m + 8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : The corresponding **DPLL_CTRL_3.AEN[n]** bit is not writeable (n=m) 1 : The corresponding **DPLL_CTRL_3.AEN[n]** bit is writeable (n=m)

Note:

DPLL_CTRL_3.AEN[n] can be written only if the corresponding **DPLL_CTRL_3.WAD[m]** is set in the same access. It can be set for debug purposes by CPU also, when DPLL is disabled. The enable bit becomes active only when the DPLL is in operation (**DPLL_CTRL_1.DEN** =1).

Note:

For **DPLL_CTRL_3.WAD[m]** =1 only the corresponding **DPLL_CTRL_3.AEN[n]** bits are writable. The **DPLL_CTRL_3.AEN[n]** bits remain unchanged when the corresponding **DPLL_CTRL_3.WAD[m]** =0.

Note:

The bitfields **DPLL_CTRL_2.AEN[n]** , **DPLL_CTRL_3.AEN[n]** , **DPLL_CTRL_4.AEN[n]** and **DPLL_CTRL_5.AEN[n]** construct the signal *AEN[n]* , where *AEN[n]* is constructed out of the concatenation of the bitfields **DPLL_CTRL_2.AEN[n]** , (n∈{0, 1,..., 7}) and **DPLL_CTRL_3.AEN[n]** , (n∈{8, 9,..., 15}) and **DPLL_CTRL_4.AEN[n]** , (n∈{16, 17,..., 23}) and **DPLL_CTRL_5.AEN[n]** , (n∈{24, 25,..., 31}).

21.11.5 DPLL_CTRL_4

Description	Action Enable Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_4
Address	$0x8010$
C-Name	GTM.CLS[0].DPLL.CTRL_4

Interface: MCS[i]

Name	DPLL_CTRL_4
Address	$0x8010$
C-Name	

AEN[n]	
Description	Action [n] enable.
Loop	$n = \{n : 16 <= n <= 23\}$
Bit Range	$[n-8 : n-8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, n, n) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The corresponding Action [n] is not enabled 1 : The corresponding Action [n] is enabled

WAD[m]	
Description	Write control bit of Action [m].
Loop	$m = \{n : 16 <= n <= 23\}$
Bit Range	$[m : m]$

WAD[m]	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : The corresponding **DPLL_CTRL_4.AEN[n]** bit is not writeable (n=m) 1 : The corresponding **DPLL_CTRL_4.AEN[n]** bit is writeable (n=m)

Note:

DPLL_CTRL_4.AEN[n] can be written only if the corresponding **DPLL_CTRL_4.WAD[m]** is set in the same access. It can be set for debug purposes by CPU also, when DPLL is disabled. The enable bit becomes active only when the DPLL is in operation (**DPLL_CTRL_1.DEN** =1).

Note:

For **DPLL_CTRL_4.WAD[m]** =1 only the corresponding **DPLL_CTRL_4.AEN[n]** bits are writable. The **DPLL_CTRL_4.AEN[n]** bits remain unchanged when the corresponding **DPLL_CTRL_4.WAD[m]** =0.

Note:

The bitfields **DPLL_CTRL_2.AEN[n]** , **DPLL_CTRL_3.AEN[n]** , **DPLL_CTRL_4.AEN[n]** and **DPLL_CTRL_5.AEN[n]** construct the signal *AEN[n]* , where *AEN[n]* is constructed out of the concatenation of the bitfields **DPLL_CTRL_2.AEN[n]** , (n∈{0, 1,..., 7}) and **DPLL_CTRL_3.AEN[n]** , (n∈{8, 9,..., 15}) and **DPLL_CTRL_4.AEN[n]** , (n∈{16, 17,..., 23}) and **DPLL_CTRL_5.AEN[n]** , (n∈{24, 25,..., 31}).

21.11.6 DPLL_CTRL_5

Description	Action Enable Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_5
Address	$0x8014$
C–Name	GTM.CLS[0].DPLL.CTRL_5

Interface: MCS[i]

Name	DPLL_CTRL_5
Address	$0x8014$
C–Name	

AEN[n]	
Description	Action [n] enable.
Loop	$n = \{n : 24 <= n <= 31\}$
Bit Range	$[n-16 : n-16]$

AEN[n]	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, n – 8, n – 8) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : The corresponding Action [n] is not enabled 1 : The corresponding Action [n] is enabled

WAD[m]	
Description	Write control bit of Action [m].
Loop	$m = \{n : 24 <= n <= 31\}$
Bit Range	$[m-8 : m-8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : The corresponding **DPLL_CTRL_5.AEN[n]** bit is not writeable (n=m) 1 : The corresponding **DPLL_CTRL_5.AEN[n]** bit is writeable (n=m)

Note:

DPLL_CTRL_5.AEN[n] can be written only if the corresponding **DPLL_CTRL_5.WAD[m]** Bit is set in the same access. It can be set for debug purposes by CPU also, when DPLL is disabled. The enable bit becomes active only when the DPLL is in operation (**DPLL_CTRL_1.DEN** =1).

Note:

For **DPLL_CTRL_5.WAD[m]** =1 only the corresponding **DPLL_CTRL_5.AEN[n]** bits are writable. The **DPLL_CTRL_5.AEN[n]** bits remain unchanged when the corresponding **DPLL_CTRL_5.WAD[m]** =0.

Note:

The bitfields **DPLL_CTRL_2.AEN[n]** , **DPLL_CTRL_3.AEN[n]** , **DPLL_CTRL_4.AEN[n]** and **DPLL_CTRL_5.AEN[n]** construct the signal *AEN[n]* , where *AEN[n]* is constructed out of the concatenation of the bitfields **DPLL_CTRL_2.AEN[n]** , (n∈{0, 1,..., 7}) and **DPLL_CTRL_3.AEN[n]** , (n∈{8, 9,..., 15}) and **DPLL_CTRL_4.AEN[n]** , (n∈{16, 17,..., 23}) and **DPLL_CTRL_5.AEN[n]** , (n∈{24, 25,..., 31}).

21.11.7 DPLL_ACT_STA

Description	Action Status Register including Shadow Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER

Clock Active Cond	DPLL_CLK_ENABLE == 1

Interface: CPU

Name	DPLL_ACT_STA
Address	$0x8018$
C-Name	GTM.CLS[0].DPLL.ACT_STA

Interface: MCS[i]

Name	DPLL_ACT_STA
Address	$0x8018$
C-Name	

ACT_N[n]	
Description	New output data values concerning to Action n provided.
Loop	$n = \{n : 0 <= n <= \text{NOAC} - 1\}$
Bit Range	$[n : n]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No new output data available after a recent PMT request or actual event value is in the past or invalid. 1 : Either new PMTR data received or calculation is repeated to be more precise by taking into account new *TRIGGER* or *STATE* values
	Note: **DPLL_ACT_STA.ACT_N[n]** is set (for *AEN[n]* =1 and a new valid PMTR), that means when new Action data are to be calculated for the corresponding Action. After each calculation of the new Actions values the **DPLL_ACT_STA.ACT_N[n]** bit updates the corresponding bit in the connected shadow register. The status of the **DPLL_ACT_STA.ACT_N[n]** bits in the shadow register is reflected by the corresponding DPLL output signal *ACT_V* (valid bit). reset together with the corresponding shadow register bit for *AEN[n]* =0; reset without the corresponding shadow register bit when the calculated event is in the past (the shadow register bit is set, when it was not set before in that case) the corresponding shadow register bit is reset, when new PMTR data are written or when the provided Action data are read (blocking read) writeable for debugging purposes together with the corresponding shadow register when **DPLL_CTRL_1.DEN** =0 **Note:** These bits can only be written for test purposes when the DPLL is disabled.

21.11.8 DPLL_OSW

Description	Offset and Switch old/new Address Register
Loop	

Condition	$NDPLL > 0$
Storage Type	REGISTER
Clock Active Cond	$DPLL\_CLK\_ENABLE == 1$

Interface: CPU

Name	DPLL_OSW
Address	$0x801C$
C-Name	GTM.CLS[0].DPLL.OSW

Interface: MCS[i]

Name	DPLL_OSW
Address	$0x801C$
C-Name	

SWON_S	
Description	Switch of new DPLL STATE. Switch bit for LSB address of STATE.
Loop	-
Bit Range	$[0:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0b0 : Use Address offset 0 for accesses to Ram1b address pairs **DPLL_TS_S** / **DPLL_TS_S_OLD** and **DPLL_NMB_S_TAR.NMB_S_TAR** / **DPLL_NMB_S_TAR_OLD.NMB_S_TAR_OLD** 0b1 : Use Address offset 1 for accesses to Ram1b address pairs **DPLL_TS_S** / **DPLL_TS_S_OLD** and **DPLL_NMB_S_TAR.NMB_S_TAR** / **DPLL_NMB_S_TAR_OLD.NMB_S_TAR_OLD**
	This bit is changed for each write access to **DPLL_TS_S** / **DPLL_TS_S_OLD** . Using this unchanged address bit **DPLL_OSW.SWON_S** for any access to **DPLL_TS_S** results always in an access to **DPLL_TS_S_OLD** . For writing to this address the former old (**DPLL_TS_S_OLD**) value is overwritten by the new one while the **DPLL_OSW.SWON_S** bit changes. Thus the former new one is now the old one and the next access is after changing **DPLL_OSW.SWON_S** directed to this place. Therefore, write to **DPLL_TS_S** first and after that immediately to **DPLL_FTV_S** and **DPLL_PSSM.PSSM** , always before a new **DPLL_TS_S** value is to be written. **Note:** After writing **DPLL_TS_S** , **DPLL_FTV_S** and **DPLL_PSSM.PSSM** in this order the address pointer AP with LSB(AP)= **DPLL_OSW.SWON_S** shows for the corresponding address to **DPLL_TS_S_OLD** , **DPLL_FTV_S** and **DPLL_PSSM.PSSM** while LSB(AP)=/ **DPLL_OSW.SWON_S** results in an access to **DPLL_TS_S** , **DPLL_FTV_S** and **DPLL_PSSM_OLD.PSSM_OLD** respectively. The value can be read only. This bit is reset while disabling the DPLL (**DPLL_CTRL_1.DEN** =0).

SWON_T	
Description	Switch of new TRIGGER. Switch bit for LSB address of TRIGGER.
Loop	-

SWON_T	
Bit Range	[1 : 1]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R–Coding	0b0 : Use address offset 0 for accesses to RAM1b address pairs **DPLL_TS_T** / **DPLL_TS_T_OLD** and **DPLL_NMB_T_TAR.NMB_T_TAR** / **DPLL_NMB_T_TAR_OLD.NMB_T_TAR_OLD** 0b1 : Use address offset 1 for accesses to RAM1b address pairs **DPLL_TS_T** / **DPLL_TS_T_OLD** and **DPLL_NMB_T_TAR.NMB_T_TAR** / **DPLL_NMB_T_TAR_OLD.NMB_T_TAR_OLD**
	This bit is changed for each write access to **DPLL_TS_T** / **DPLL_TS_T_OLD** . Using this unchanged address bit **DPLL_OSW.SWON_T** for any access to **DPLL_TS_T** always results in an access to **DPLL_TS_T_OLD** . For writing to this address the former old (**DPLL_TS_T_OLD**) value is overwritten by the new one while the **DPLL_OSW.SWON_T** bit changes. Thus, the former new one is now the old one and the next access is after changing **DPLL_OSW.SWON_T** directed to this place. Therefore, write to **DPLL_TS_T** first and after that immediately to **DPLL_FTV_T** and **DPLL_PSTM.PSTM** , always before a new **DPLL_TS_T** value is to be written. **Note:** After writing **DPLL_TS_T** , **DPLL_FTV_T** and **DPLL_PSTM.PSTM** in this order the address pointer AP with LSB(AP)= **DPLL_OSW.SWON_T** shows for the corresponding address to **DPLL_TS_T_OLD** , **DPLL_FTV_T** and **DPLL_PSTM.PSTM** while LSB(AP)=/ **DPLL_OSW.SWON_T** results in an access to **DPLL_TS_T** , **DPLL_FTV_T** _old and **DPLL_PSTM_OLD.PSTM_OLD** respectively. The value can be read only. This bit is reset while disabling the DPLL (**DPLL_CTRL_1.DEN** =0).

OSS	
Description	Offset size of RAM region 2
Loop	–
Bit Range	[9 : 8]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	2
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0x0 : Offset size 128 of RAM region 2. 0x1 : Offset size 256 of RAM region 2. 0x2 : Offset size 512 of RAM region 2. 0x3 : Offset size 1024 of RAM region 2.

OSS	
	Note: At least 128 and a maximum of 1024 values can be stored in each of the RAM 2 regions a to d accordingly. The value can be set only for **DPLL_CTRL_1.DEN** =0. The change of the **DPLL_OSW.OSS** value results in an automatic change of the offset values in the **DPLL_AOSV_2** register. **DPLL_OSW.OSS** must be set in accordance with the device configuration variable DPLL_RR2_MEM_SIZE (see *86 "GTM Device Configuration Variables, Fixed or Derived Values"*). E.g.: for the largest memory size (12 Kbyte) it can be set to any value, whereas for the smallest memory size (1.5 Kbyte) only 0x0 is sensible. **Note:** This value can only be written when the DPLL is disabled.

21.11.9 DPLL_AOSV_2

Description	Address Offset Register of RAM 2 Regions
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_AOSV_2
Address	$0x8020$
C-Name	GTM.CLS[0].DPLL.AOSV_2

Interface: MCS[i]

Name	DPLL_AOSV_2
Address	$0x8020$
C-Name	

AOSV_2A	
Description	Address offset value of the RAM 2A region.
Loop	-
Bit Range	$[7:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

AOSV_2A	
	The value in this field is to be multiplied by 256 (shift left 8 Bits) and added with the start address of the RAM in order to get the start address of RAM region 2a. When the **DPLL_APT.APT** value is added to this start address, the current RAM cell **DPLL_RDT_T[p].RDT_T** is addressed. The value is set automatically when **DPLL_OSW.OSS** is set: **DPLL_OSW.OSS** =0x0: **DPLL_AOSV_2.AOSV_2A** = 0x00 **DPLL_OSW.OSS** =0x1: **DPLL_AOSV_2.AOSV_2A** = 0x00 **DPLL_OSW.OSS** =0x2: **DPLL_AOSV_2.AOSV_2A** = 0x00 **DPLL_OSW.OSS** =0x3: **DPLL_AOSV_2.AOSV_2A** = 0x00

AOSV_2B	
Description	Address offset value of the RAM 2B region.
Loop	–
Bit Range	$[15:8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0x8$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	The value in this field is to be multiplied by 256 (shift left 8 Bits) and added with the start address of the RAM in order to get the start address of RAM region 2b. When the **DPLL_APT.APT** value is added to this start address, the current RAM cell **DPLL_TSF_T[p].TSF_T** is addressed. The value is set automatically when **DPLL_OSW.OSS** is set: **DPLL_OSW.OSS** =0x0: **DPLL_AOSV_2.AOSV_2B** = 0x02 **DPLL_OSW.OSS** =0x1: **DPLL_AOSV_2.AOSV_2B** = 0x04 **DPLL_OSW.OSS** =0x2: **DPLL_AOSV_2.AOSV_2B** = 0x08 **DPLL_OSW.OSS** =0x3: **DPLL_AOSV_2.AOSV_2B** = 0x10

AOSV_2C	
Description	Address offset value of the RAM 2C region.
Loop	–
Bit Range	$[23:16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0x10$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

AOSV_2C	
	The value in this field is to be multiplied by 256 (shift left 8 Bits) and added with the start address of the RAM in order to get the start address of RAM region 2c. When the **DPLL_APT.APT** value is added to this start address, the current RAM cell **DPLL_ADT_T[p]** is addressed. The value is set automatically when **DPLL_OSW.OSS** is set: **DPLL_OSW.OSS** =0x0: **DPLL_AOSV_2.AOSV_2C** = 0x04 **DPLL_OSW.OSS** =0x1: **DPLL_AOSV_2.AOSV_2C** = 0x08 **DPLL_OSW.OSS** =0x2: **DPLL_AOSV_2.AOSV_2C** = 0x10 **DPLL_OSW.OSS** =0x3: **DPLL_AOSV_2.AOSV_2C** = 0x20

AOSV_2D	
Description	Address offset value of the RAM 2D region.
Loop	–
Bit Range	$[31 : 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0x18$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	The value in this field is to be multiplied by 256 (shift left 8 Bits) and added with the start address of the RAM in order to get the start address of RAM region 2d. When the **DPLL_APT.APT** value is added to this start address, the current RAM cell **DPLL_DT_T[p]** is addressed. The value is set automatically when **DPLL_OSW.OSS** is set: **DPLL_OSW.OSS** =0x0: **DPLL_AOSV_2.AOSV_2D** = 0x06 **DPLL_OSW.OSS** =0x1: **DPLL_AOSV_2.AOSV_2D** = 0x0C **DPLL_OSW.OSS** =0x2: **DPLL_AOSV_2.AOSV_2D** = 0x18 **DPLL_OSW.OSS** =0x3: **DPLL_AOSV_2.AOSV_2D** = 0x30 **Note:** The offset values are needed to support a scalable RAM size of region 2 from 1.5 Kbytes to 12 Kbytes. The values above must be in correlation with the offset size defined in the OSW register. All offset values are set automatically in accordance with the **DPLL_OSW.OSS** value. This value can be set only for **DPLL_CTRL_1.DEN** =0.

21.11.10 DPLL_APT

Description	Actual RAM Pointer Address for TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APT
Address	$0x8024$
C-Name	GTM.CLS[0].DPLL.APT

Interface: MCS[i]

Name	DPLL_APT
Address	$0x8024$
C-Name	

WAPT	
Description	Write bit for address pointer DPLL_APT.APT, read as zero.
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : The **DPLL_APT.APT** is not writeable 1 : The **DPLL_APT.APT** is writeable

APT	
Description	Address pointer DPLL TRIGGER:
Loop	-
Bit Range	$[11:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 1, 1) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This pointer is used for the RAM region 2 subsections 2a and 2d. The pointer **DPLL_APT.APT** is incremented for each active *TRIGGER* event (simultaneously with **DPLL_APT.APT_2B** , **DPLL_APT_2C.APT_2C**) for *DIR1* =0. For *DIR1* =1 the **DPLL_APT.APT** is decremented. The **DPLL_APT.APT** offset value is added in the above shown bit position with the subsection address offset of the corresponding RAM region **Note:** The **DPLL_APT.APT** pointer value is directed to the RAM position, in which the data values are to be written, which corresponds to the last increment. The **DPLL_APT.APT** value is not to be changed, when the direction (shown by *DIR1*) changes, because it points always to a storage place after the considered increment. Changing of *DIR1* takes place always after an active *TRIGGER* event and the resulting increment/decrement. ▽

APT	
	△ **Note:** This value can only be written when the **DPLL_APT.WAPT** bit is set. **Note:** Actual RAM pointer address value offset for DT_T[p] and RDT_T[p] in FULL_SCALE for 2*(**DPLL_CTRL_0.TNU** +1− **DPLL_CTRL_1.SYN_NT**) *TRIGGER* events.

WAPT_2B	
Description	Write bit for address pointer DPLL_APT.APT_2B, read as zero.
Loop	–
Bit Range	[13 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W−Coding	0 : The **DPLL_APT.APT_2B** is not writeable 1 : The **DPLL_APT.APT_2B** is writeable

APT_2B	
Description	Address pointer TRIGGER for RAM region 2b.
Loop	–
Bit Range	[23 : 14]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 13, 13) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

APT_2B	
	Actual RAM pointer address of *TRIGGER* events in FULL_SCALE for 2*(**DPLL_CTRL_0.TNU** +1) *TRIGGER* periods; this pointer is used for the RAM region 2b. The RAM pointer is initially set to zero. For **DPLL_STATUS.SYT** =1: The pointer **DPLL_APT.APT_2B** is incremented by **DPLL_NUTC.SYN_T_OLD** for each active *TRIGGER* event (simultaneously with **DPLL_APT.APT** and **DPLL_APT_2C.APT_2C**) for *DIR1* =0 when an active *TRIGGER* input appears. For *DIR1* =1 (backwards) the **DPLL_APT.APT** is decremented by **DPLL_NUTC.SYN_T_OLD** . For **DPLL_STATUS.SYT** =0: **DPLL_APT.APT_2B** is incremented or decremented by 1. In addition, when the **DPLL_APT_2C.APT_2C** value is written by the CPU – in order to synchronize the DPLL– with the next active *TRIGGER* event the **DPLL_APT_SYNC.APT_2B_EXT** value is added/subtracted (while **DPLL_APT_SYNC.APT_2B_STATUS** is one; see **DPLL_APT_SYNC** register at chapter **DPLL_APT_SYNC**). **Note:** This value can only be written when the **DPLL_APT.WAPT_2B** bit is set. Actual RAM pointer address value for **DPLL_TSF_T[p]** .

21.11.11 DPLL_APS

Description	Actual RAM Pointer Address for STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APS
Address	$0x8028$
C–Name	GTM.CLS[0].DPLL.APS

Interface: MCS[i]

Name	DPLL_APS
Address	$0x8028$
C–Name	

WAPS	
Description	Write bit for address pointer DPLL_APS.APS, read as zero.
Loop	–
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

WAPS	
W-Coding	0 : The **DPLL_APS.APS** is not writeable 1 : The **DPLL_APS.APS** is writeable

APS	
Description	Address pointer STATE: Actual RAM pointer address value for DPLL_DT_S[p] and DPLL_RDT_S[p]
Loop	-
Bit Range	[7 : 2]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 1, 1) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Actual RAM pointer and synchronization position/value of *STATE* events in FULL_SCALE for up to 64 *STATE* events but limited to 2*(**DPLL_CTRL_0.SNU** +1- **DPLL_CTRL_1.SYN_NS**) in normal and emergency mode for **DPLL_CTRL_1.SYSF** =0 or to 2*(**DPLL_CTRL_0.SNU** +1)- **DPLL_CTRL_1.SYN_NS** for **DPLL_CTRL_1.SYSF** =1 respectively; this pointer is used for the RAM region 1c1 and 1c4. **DPLL_APS.APS** is incremented (decremented) by one for each active *STATE* event and *DIR2* =0 *DIR2* =1). The **DPLL_APS.APS** offset value is added in the above shown bit position with the subsection offset of the RAM region. **Note:** The **DPLL_APS.APS** pointer value is directed to the RAM position, in which the data values are to be written, which correspond to the last increment. The **DPLL_APS.APS** value is not to be changed, when the direction (shown by *DIR2*) changes, because it points always to a storage place after the considered increment. Changing of *DIR2* takes place always after an active *STATE* event and the resulting increment/decrement. **Note:** This value can only be written when the **DPLL_APS.WAPS** bit is set.

WAPS_1C2	
Description	Write bit for address pointer DPLL_APS.APS_1C2, read as zero.
Loop	-
Bit Range	[13 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : The **DPLL_APS.APS_1C2** is not writeable 1 : The **DPLL_APS.APS_1C2** is writeable

APS_1C2	
Description	Actual RAM pointer address value for DPLL_TSF_S[p].
Loop	–
Bit Range	$[19 : 14]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 13, 13) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Initial value: zero (0x00). Actual RAM pointer and synchronization position/value of *STATE* events in FULL_SCALE for up to 64 *STATE* events but limited to 2*(**DPLL_CTRL_0.SNU** +1) in normal and emergency mode; this pointer is used for the RAM region 1c2. For **DPLL_STATUS.SYS** =1: **DPLL_APS.APS_1C2** is incremented (decremented) by **DPLL_NUSC.SYN_S_OLD** for each active *STATE* event and *DIR2* =0 (*DIR2* =1). For **DPLL_STATUS.SYS** =0: **DPLL_APS.APS_1C2** is incremented or decremented by 1 respectively. The **DPLL_APS.APS_1C2** offset value is added in the above shown bit position with the subsection offset of the RAM region. In addition, when the **DPLL_APS_1C3.APS_1C3** value is written by the CPU – in order to synchronize the D-PLL– with the next active *STATE* event the **DPLL_APS_SYNC.APS_1C2_EXT** value is added/subtracted (while **DPLL_APS_SYNC.APS_1C2_STATUS** is one; see **DPLL_APT_SYNC** register at chapter **DPLL_APS_SYNC**). **Note:** This value can only be written when the **DPLL_APS.WAPS_1C2** bit is set

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is not set. If **DPLL_CTRL_11.STATE_EXT** is set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is set a read access to this register will return *AEI_STATUS* = 0b00 and the read value is equal to **DPLL_APS_EXT.APS_1C2** and **DPLL_APS_EXT.APS** .

21.11.12 DPLL_APT_2C

Description	Actual RAM Pointer Address for Region 2c
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APT_2C
Address	$0x802C$
C–Name	GTM.CLS[0].DPLL.APT_2C

Interface: MCS[i]

Name	DPLL_APT_2C
Address	$0x802C$

C-Name	

APT_2C	
Description	Address pointer TRIGGER for RAM region 2c: Actual RAM pointer address value for DPLL_ADT_T[p].
Loop	-
Bit Range	$[11 : 2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Actual RAM pointer address value of *TRIGGER* adapt events in FULL_SCALE for 2*(**DPLL_CTRL_0.TNU** +1-- **DPLL_CTRL_1.SYN_NT**) *TRIGGER* periods depending on the size of the used RAM 2; this pointer is used for the RAM region 2 for the subsection 2c only. The RAM pointer is initially set to zero. The **DPLL_APT_2C.APT_2C** value is set by the CPU when the synchronization condition was detected. Within the RAM region 2c initially the conditions for synchronization gaps and adapted values are stored by the CPU.

Note:

The **DPLL_APT_2C.APT_2C** pointer values are directed to the RAM position of the profile element in RAM region 2c, which correspond to the current increment. For *DIR1* =0 (*DIR1* =1) the pointer **DPLL_APT_2C.APT_2C** is incremented (decremented) by 1 simultaneously with **DPLL_APT.APT** . For **DPLL_CTRL_1.SMC** =0 the change of *DIR1* takes place always after an active *TRIGGER* event (by evaluation of the inactive slope) and the resulting increment/decrement. In case **DPLL_CTRL_1.SMC** =1 the direction change is known before the input event is processed.
The correction of the **DPLL_APT_2C.APT_2C** pointer differs: for **DPLL_CTRL_1.SMC** =0 correct 4 times and for **DPLL_CTRL_1.SMC** =1 correct only 2 times.
The **DPLL_APT_2C.APT_2C** offset value is added in the above shown bit position with the subsection address offset of the corresponding RAM region.

21.11.13 DPLL_APS_1C3

Description	Actual RAM Pointer Address for RAM region 1c3
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APS_1C3
Address	$0x8030$
C-Name	GTM.CLS[0].DPLL.APS_1C3

Interface: MCS[i]

Name	DPLL_APS_1C3
Address	$0x8030$

C-Name	

APS_1C3	
Description	Address pointer STATE for RAM region 1c3: Actual RAM pointer address value for DPLL_ADT_S[p]
Loop	-
Bit Range	$[7:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Initial value: zero (0x00). Actual RAM pointer and synchronization position/value of *STATE* events in FULL_SCALE for up to 64 *STATE* events but limited to 2*(**DPLL_CTRL_0.SNU** +1- **DPLL_CTRL_1.SYN_NS**) in normal and emergency mode for **DPLL_CTRL_1.SYSF** =0 or to 2*(**DPLL_CTRL_0.SNU** +1)- **DPLL_CTRL_1.SYN_NS** for **DPLL_CTRL_1.SYSF** =1 respectively; this pointer is used for the RAM region 1c3. The RAM pointer is set by the CPU accordingly, when the synchronization condition was detected.

Note:

The **DPLL_APS_1C3.APS_1C3** pointer value is directed to the RAM position of the profile element in RAM region 1c2, which corresponds to the current increment. When changing the direction *DIR1* or *DIR2* respectively, this is always known before an active *STATE* event is processed. This is because of the pattern recognition in SPE (for PMSM) or because of the direction change recognition by *TRIGGER* . This direction change results in an automatic increment (forwards) or decrement (backwards) when the input event occurs in addition to a 2 times correction.
The **DPLL_APS_1C3.APS_1C3** offset value is added in the above shown bit position with the subsection address offset of the corresponding RAM region.

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is not set. If **DPLL_CTRL_11.STATE_EXT** is set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is set any read access to this register will return *AEI_STATUS* = 0b00 and the read value is equal to **DPLL_APS_1C3_EXT.APS_1C3** .

21.11.14 DPLL_NUTC

Description	Number of Recent TRIGGER Events used for Calculations
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NUTC
Address	$0x8034$
C-Name	GTM.CLS[0].DPLL.NUTC

Interface: MCS[i]

Name	DPLL_NUTC

Address	$0x8034$
C-Name	

NUTE	
Description	Number of recent TRIGGER events used for SUB_INC1 and Action calculations modulo 2*(TNU_max+1).
Loop	-
Bit Range	$[9:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 29, 29) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The **DPLL_NUTC.NUTE** value is less than FULL_SCALE 1 : The **DPLL_NUTC.NUTE** value is equal to FULL_SCALE
	DPLL_NUTC.NUTE : number of last nominal increments to be considered for the calculations. No gap is considered in that case for this value, but in the **DPLL_NUTC.VTN** value (see below): This value is set by the CPU, but reset automatically to "1" by a change of direction or loss of LOCK. Every other value can be set by the CPU, maybe FULL_SCALE, HALF_SCALE or parts of them. For FULL_SCALE set **DPLL_NUTC.NUTE** = 2*(**DPLL_CTRL_0.TNU** +1) and for HALF_SCALE **DPLL_NUTC.NUTE** = **DPLL_CTRL_0.TNU** +1. The relation values QDT_T[x] are calculated using **DPLL_NUTC.NUTE** values in the past with its maximum value of 2*(**DPLL_CTRL_0.TNU** +1). The value zero (in combination with the value **DPLL_NUTC.FST** =1) means 2^{11} values in the past. **Note:** To prevent that inconsistencies between internal pointer in which **DPLL_NUTC.NUTE** is used and the case decision of different prediction method's for prediction of the next event and PMT(position minus time) occur, the **DPLL_NUTC.NUTE** value is stored internally at that point of time when the internal pointers are calculated for the next event cycle This value is set by the CPU, but reset automatically to "0" by a change of direction or loss of LOCK. **Note:** This value can only be written when the **DPLL_NUTC.WNUT** bit is set.

FST	
Description	FULL_SCALE of TRIGGER.
Loop	-
Bit Range	$[10:10]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 29, 29) == 0

FST	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0b0 : The **DPLL_NUTC.NUTE** value is less than FULL_SCALE 0b1 : The **DPLL_NUTC.NUTE** value is equal to FULL_SCALE
	Note: This value can only be written when the **DPLL_NUTC.WNUT** bit is set. This value is to be set, when DPLL_NUTC.NUTE is set to FULL_SCALE.

SYN_T	
Description	Number of real and virtual events to be considered for the current increment.
Loop	–
Bit Range	[15 : 13]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 30, 30) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value reflects the **DPLL_ADT_T[p].NT** value of the last valid increment; to be updated after all calculations in step 17 of Table in chapter *51 "State description of the State Machine Table"* . **Note:** This value can only be written when the **DPLL_NUTC.WSYN** bit in this register is set.

SYN_T_OLD	
Description	Number of real and virtual events to be considered for the last increment.
Loop	–
Bit Range	[18 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 30, 30) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Note: This value can only be written when the **DPLL_NUTC.WSYN** bit in this register is set.

VTN	
Description	Virtual DPLL TRIGGER number:
Loop	–
Bit Range	[24 : 19]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 31, 31) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value reflects the number of virtual increments in the current **DPLL_NUTC.NUTE** region; for **DPLL_NUTC.NUTE** =1 this value is zero, when the CPU sets **DPLL_NUTC.NUTE** to a value > 1 , it must also set **DPLL_NUTC.VTN** to the corresponding value; for **DPLL_NUTC.NUTE** is set to FULL_SCALE including **DPLL_NUTC.NUTE** =zero (2^{11} modulo 2^{11}) the **DPLL_NUTC.VTN** is to be set to 2* **DPLL_CTRL_1.SYN_NT** . The **DPLL_NUTC.VTN** value is subtracted from the **DPLL_NUTC.NUTE** value in order to get the corresponding **DPLL_APT.APT** value for the past; the **DPLL_NUTC.VTN** value is not used for the **DPLL_APT.APT_2B** pointer. **DPLL_NUTC.VTN** is to be updated by the CPU when a new gap is to be considered for **DPLL_NUTC.NUTE** or a gap is leaving the **DPLL_NUTC.NUTE** region; for this purpose the **DPLL_ADT_T[p].TINT** values in the profile can be used to generate an interrupt for the CPU at the corresponding positions; no further update of **DPLL_NUTC.VTN** is necessary when **DPLL_NUTC.NUTE** is set to FULL_SCALE **Note:** This value can only be written when the **DPLL_NUTC.WVTN** bit is set. Number of virtual increments in the current **DPLL_NUTC.NUTE** region

WNUT	
Description	Write control bit for DPLL_NUTC.NUTE and DPLL_NUTC.FST. read as zero.
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : The **DPLL_NUTC.NUTE** value is not writeable 1 : The **DPLL_NUTC.NUTE** value is writeable

WSYN	
Description	Write control bit for DPLL_NUTC.SYN_T and DPLL_NUTC.SYN_T_OLD. read as zero.
Loop	–

WSYN	
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : The **DPLL_NUTC.SYN_T** value is not writeable 1 : The **DPLL_NUTC.SYN_T** value is writeable

WVTN	
Description	Write control bit for DPLL_NUTC.VTN. read as zero.
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : The **DPLL_NUTC.VTN** value is not writeable 1 : The **DPLL_NUTC.VTN** value is writeable

Note:
DPLL Number of recent *TRIGGER* events used for calculations (mod 2*(**DPLL_CTRL_0.TNU** +1– **DPLL_CTRL_1.SYN_NT**)).

21.11.15 DPLL_NUSC

Description	Number of Recent STATE Events used for Calculations
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NUSC
Address	$0x8038$

C–Name	GTM.CLS[0].DPLL.NUSC

Interface: MCS[i]

Name	DPLL_NUSC
Address	$0x8038$
C–Name	

NUSE	
Description	Number of recent STATE events used for SUB_INC1 and SUB_INC2 calculations modulo 2*(DPLL_CTRL_0.SNU+1).
Loop	–
Bit Range	$[5:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 29, 29) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	No gap is considered in that case for this value, but in the **DPLL_NUSC.VSN** value (see below): This register is set by the CPU but reset automatically to "1" by a change of direction or loss of LOCK. Each other value can be set by the CPU, maybe Full_SCALE, HALF_SCALE or parts of them. The relation values QDT_S[x] are calculated using **DPLL_NUSC.NUSE** values in the past with its maximum value of 2* **DPLL_CTRL_0.SNU** +1. **Note:** In order to prevent inconsistencies between internal pointer, in which **DPLL_NUSC.NUSE** is used and while deciding on different prediction methods to predict the next event and PMT(position minus time), the **DPLL_NUSC.NUSE** value is stored internally at the time, when the internal pointers are calculated for the next event cycle. **Note:** This value can only be written when the **DPLL_NUSC.WNUS** bit is set.

FSS	
Description	Full scale of STATE.
Loop	–
Bit Range	$[6:6]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 29, 29) == 0

FSS	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The **DPLL_NUSC.NUSE** value is less than FULL_SCALE 1 : The **DPLL_NUSC.NUSE** value is equal to FULL_SCALE
	This value is set by the CPU, but reset automatically to "0" by a change of direction or loss of LOCK. **Note:** This value can only be written when the **DPLL_NUSC.WNUS** bit is set. **Note:** This value is to be set, when **DPLL_NUSC.NUSE** is set to FULL_SCALE.

SYN_S	
Description	Number of real and virtual events to be considered for the current increment.
Loop	-
Bit Range	[12 : 7]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 30, 30) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value reflects the **DPLL_ADT_S[p].NS** value of the last valid increment; to be updated after all calculations in step 37 of Table *51 "State description of the State Machine Table"* . **Note:** This value can only be written when the **DPLL_NUSC.WSYN** bit in this register is set.

SYN_S_OLD	
Description	Number of real and virtual events to be considered for the last increment.
Loop	-
Bit Range	[18 : 13]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 30, 30) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

SYN_S_OLD	
	Note: This value can only be written when the **DPLL_NUSC.WSYN** bit in this register is set.

VSN	
Description	Virtual STATE number.
Loop	–
Bit Range	[24 : 19]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 31, 31) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value reflects the number of virtual increments in the current **DPLL_NUSC.NUSE** region; for **DPLL_NUSC.NUSE** =1 this value is zero, when the CPU sets **DPLL_NUSC.NUSE** to a value > 1 or zero(2^7 modulo 2^7) , it must also set **DPLL_NUSC.VSN** to the corresponding value; the **DPLL_NUSC.VSN** value is subtracted from the **DPLL_NUSC.NUSE** value in order to get the corresponding **DPLL_APS.APS** value for the past; the **DPLL_NUSC.VSN** value is not used for the **DPLL_APS.APS_1C2** pointer. **DPLL_NUSC.VSN** is to be updated by the CPU when a new gap is to be considered for **DPLL_NUSC.NUSE** or a gap is leaving the **DPLL_NUSC.NUSE** region; for this purpose the **DPLL_IRQ_NOTIFY.SASI** interrupt can be used; no further update of **DPLL_NUSC.VSN** is necessary when **DPLL_NUSC.NUSE** is set to FULL_SCALE **Note:** This value can only be written when the **DPLL_NUSC.WVSN** bit in this register is set. **Note:** Number of virtual *STATE* increments in the current DPLL_NUSC.NUSE region.

WNUS	
Description	Write control bit for DPLL_NUSC.NUSE, read as zero.
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

WNUS	
W-Coding	0 : The **DPLL_NUSC.NUSE** value is not writeable 1 : The **DPLL_NUSC.NUSE** value is writeable

WSYN	
Description	Write control bit for DPLL_NUSC.SYN_S and DPLL_NUSC.SYN_S_OLD, read as zero.
Loop	-
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : the **DPLL_NUSC.SYN_S** value is not writeable 1 : the **DPLL_NUSC.SYN_S** value is writeable

WVSN	
Description	Write control bit for DPLL_NUSC.VSN, read as zero.
Loop	-
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : The **DPLL_NUSC.VSN** value is not writeable 1 : The **DPLL_NUSC.VSN** value is writeable

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is not set. If **DPLL_CTRL_11.STATE_EXT** is set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is set any read access to this register will return *AEI_STATUS* = 0b00 and the data values are equal to: **DPLL_NUSC.NUSE** to **DPLL_NUSC_EXT2.NUSE** , **DPLL_NUSC.FSS** to **DPLL_NUSC_EXT2.FSS** , **DPLL_NUSC.SYN_S** to **DPLL_NUSC_EXT1.SYN_S** , **DPLL_NUSC.SYN_S_OLD** to **DPLL_NUSC_EXT1.SYN_S_OLD** and **DPLL_NUSC.VSN** to **DPLL_NUSC_EXT2.VSN** .

21.11.16 DPLL_NTI_CNT

Description	Number of Active TRIGGER Events to Interrupt
Loop	

Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NTI_CNT
Address	$0x803C$
C-Name	GTM.CLS[0].DPLL.NTI_CNT

Interface: MCS[i]

Name	DPLL_NTI_CNT
Address	$0x803C$
C-Name	

NTI_CNT	
Description	Number of triggers to interrupt.
Loop	-
Bit Range	$[9:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value shows the remaining *TRIGGER* events until an active *TRIGGER* slope results in a *DPLL_CDTI* interrupt; the value is to be count down for each active *TRIGGER* event. Number of active *TRIGGER* events to the next *DPLL_CDTI* interrupt.

21.11.17 DPLL_IRQ_NOTIFY

Description	Interrupt Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_IRQ_NOTIFY

Address	$0x8040$
C-Name	GTM.CLS[0].DPLL.IRQ_NOTIFY

Interface: MCS[i]

Name	DPLL_IRQ_NOTIFY
Address	$0x8040$
C-Name	

PDI	
Description	DPLL disable interrupt.
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This event is combined with the **DPLL_IRQ_NOTIFY.PEI** interrupt to the common **DPLL_IRQ_NOTIFY.PDI** + **DPLL_IRQ_NOTIFY.PEI** interrupt line number 1. **Note:** **DPLL_IRQ_NOTIFY.PDI** announces the switch off of the **DPLL_CTRL_1.DEN** bit.

PEI	
Description	DPLL enable interrupt.
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

PEI	
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: **DPLL_IRQ_NOTIFY.PEI** announces the switch on of the DPLL_CTRL_1.DEN bit. **Note:** This event is combined with the **DPLL_IRQ_NOTIFY.PDI** interrupt to the common **DPLL_IRQ_NOTIFY.PDI** + **DPLL_IRQ_NOTIFY.PEI** interrupt line number 1.

TINI	
Description	TRIGGER minimum hold time violation interrupt (dt <= DPLL_THMI.THMI > 0).
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TAXI	
Description	TRIGGER maximum hold time violation interrupt (dt > DPLL_THMA.THMA > 0).
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised

TAXI	
W-Coding	0 : No action 1 : Clear interrupt

SISI	
Description	STATE inactive slope interrupt.
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TISI	
Description	TRIGGER inactive slope interrupt.
Loop	–
Bit Range	$[5:5]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: The **DPLL_IRQ_NOTIFY.TISI** bit is only set for an inactive slope when the preceding active slope was accepted. In the case of suppression of the last active slope by the plausibility check the next inactive slope is to be ignored. No set of **DPLL_IRQ_NOTIFY.TISI** is performed in this case.

MSI	
Description	Missing STATE interrupt.

MSI	
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

MTI	
Description	Missing TRIGGER interrupt.
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

SASI	
Description	STATE active slope interrupt.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–

SASI	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TASI	
Description	TRIGGER active slope interrupt.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

PWI	
Description	Plausibility window (PVT) violation interrupt of TRIGGER.
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised

PWI	
W-Coding	0 : No action 1 : Clear interrupt

W2I	
Description	RAM write access to RAM region 2 interrupt.
Loop	-
Bit Range	[11 : 11]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

W1I	
Description	Write access to RAM region 1b or 1c interrupt.
Loop	-
Bit Range	[12 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

GL1I	
Description	Get lock interrupt for SUB_INC1.
Loop	-
Bit Range	[13 : 13]
Access Type	RW

GL1I	
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

LL1I	
Description	Loss of lock interrupt for SUB_INC1.
Loop	–
Bit Range	[14 : 14]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

EI	
Description	Error interrupt (see status register bit 31).
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–

EI	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

GL2I	
Description	Get lock interrupt for SUB_INC2.
Loop	-
Bit Range	[16 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

LL2I	
Description	Loss of lock interrupt for SUB_INC2.
Loop	-
Bit Range	[17 : 17]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TE0I	
Description	TRIGGER event interrupt 0.
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TE1I	
Description	TRIGGER event interrupt 1.
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TE2I	
Description	TRIGGER event interrupt 2.
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–

TE2I	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TE3I	
Description	TRIGGER event interrupt 3.
Loop	–
Bit Range	[21 : 21]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TE4I	
Description	TRIGGER event interrupt 4.
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised

TE4I	
W-Coding	0 : No action 1 : Clear interrupt

CDTI	
Description	TRIGGER duration is calculated only when DPLL_NTI_CNT.NTI_CNT is zero.
Loop	-
Bit Range	[23 : 23]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

CDSI	
Description	Calculation of STATE duration done
Loop	-
Bit Range	[24 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

TORI	
Description	TRIGGER out of range interrupt
Loop	-
Bit Range	[25 : 25]
Access Type	RW

TORI	
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

SORI	
Description	STATE out of range
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

DCGI	
Description	Direction change interrupt
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–

DCGI	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: The interrupt occurs at line number 0.

Note:
All bits in the **DPLL_IRQ_NOTIFY** register are set permanently until writing a one bit value is performed to the corresponding bit.

21.11.18 DPLL_IRQ_EN

Description	Interrupt Enable Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_IRQ_EN
Address	$0x8044$
C-Name	GTM.CLS[0].DPLL.IRQ_EN

Interface: MCS[i]

Name	DPLL_IRQ_EN
Address	$0x8044$
C-Name	

PDI_IRQ_EN	
Description	DPLL disable interrupt enable, when switch off of the DPLL_CTRL_1.DEN bit.
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

PDI_IRQ_EN	
RW-Coding	0 : The DPLL disable interrupt is not enabled 1 : The DPLL disable interrupt is enabled

PEI_IRQ_EN	
Description	DPLL enable interrupt enable, when switch on of the DPLL_CTRL_1.DEN bit.
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The DPLL enable interrupt is not enabled 1 : The DPLL enable interrupt is enabled

TINI_IRQ_EN	
Description	TRIGGER minimum hold time violation interrupt enable bit.
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Minimum hold time violation of *TRIGGER* interrupt is not enabled 1 : The minimum hold time violation of *TRIGGER* interrupt is enabled

TAXI_IRQ_EN	
Description	TRIGGER maximum hold time violation interrupt enable bit.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

TAXI_IRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Maximum hold time violation of *TRIGGER* interrupt is not enabled 1 : The maximum hold time violation of *TRIGGER* interrupt is enabled

SISI_IRQ_EN	
Description	STATE inactive slope interrupt enable bit.
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The interrupt at the inactive slope of *STATE* is not enabled 1 : The interrupt at the inactive slope of *STATE* is enabled

TISI_IRQ_EN	
Description	TRIGGER inactive slope interrupt enable bit.
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The interrupt at the inactive slope of *TRIGGER* is not enabled 1 : The interrupt at the inactive slope of *TRIGGER* is enabled

MSI_IRQ_EN	
Description	Missing STATE interrupt enable.
Loop	–

MSI_IRQ_EN	
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The missing *STATE* interrupt is not enabled 1 : The missing *STATE* interrupt is enabled

MTI_IRQ_EN	
Description	Missing TRIGGER interrupt enable.
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The missing *TRIGGER* interrupt is not enabled 1 : The missing *TRIGGER* interrupt is enabled

SASI_IRQ_EN	
Description	STATE active slope interrupt enable.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

SASI_IRQ_EN	
RW-Coding	0 : The active slope *STATE* interrupt is not enabled. 1 : The active slope *STATE* interrupt is enabled

TASI_IRQ_EN	
Description	TRIGGER active slope interrupt enable.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The active slope *TRIGGER* interrupt is not enabled 1 : The active slope *TRIGGER* interrupt is enabled

PWI_IRQ_EN	
Description	Plausibility window (PVT) violation interrupt of TRIGGER enable.
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The plausibility violation interrupt is not enabled 1 : The plausibility violation interrupt is enabled

W2I_IRQ_EN	
Description	RAM write access to RAM region 2 interrupt enable.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

W2I_IRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The RAM write access interrupt is not enabled 1 : The RAM write access interrupt is enabled

W1I_IRQ_EN	
Description	Write access to RAM region 1b or 1c interrupt.
Loop	–
Bit Range	[12 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The RAM write access interrupt is not enabled 1 : The RAM write access interrupt is enabled.

GL1I_IRQ_EN	
Description	Get lock interrupt enable, when lock arises.
Loop	–
Bit Range	[13 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The lock getting interrupt is not enabled 1 : The lock getting interrupt is enabled

LL1I_IRQ_EN	
Description	Loss of lock interrupt enable.
Loop	–

LL1I_IRQ_EN	
Bit Range	[14 : 14]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The lock loss interrupt is not enabled 1 : The lock loss interrupt is enabled

EI_IRQ_EN	
Description	Error interrupt enable (see status register).
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The error interrupt is not enabled 1 : The error interrupt is enabled

GL2I_IRQ_EN	
Description	Get lock interrupt enable for SUB_INC2.
Loop	–
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

GL2I_IRQ_EN	
RW-Coding	0 : The lock getting interrupt is not requested 1 : The lock getting interrupt is requested

LL2I_IRQ_EN	
Description	Loss of lock interrupt enable for SUB_INC2.
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The lock loss interrupt is not requested 1 : The lock loss interrupt is requested

TE0I_IRQ_EN	
Description	TRIGGER event interrupt 0 enable.
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 0 enabled 1 : Interrupt on *TRIGGER* event 0 enabled

TE1I_IRQ_EN	
Description	TRIGGER event interrupt 1 enable.
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

TE1I_IRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 1 enabled 1 : Interrupt on *TRIGGER* event 1 enabled

TE2I_IRQ_EN	
Description	TRIGGER event interrupt 2 enable.
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 2 enabled 1 : Interrupt on *TRIGGER* event 2 enabled

TE3I_IRQ_EN	
Description	TRIGGER event interrupt 3 enable.
Loop	–
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 3 enabled 1 : Interrupt on *TRIGGER* event 3 enabled

TE4I_IRQ_EN	
Description	TRIGGER event interrupt 4 enable.
Loop	–

TE4I_IRQ_EN	
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 4 enabled 1 : Interrupt on *TRIGGER* event 4 enabled

CDTI_IRQ_EN	
Description	Enable interrupt when calculation of TRIGGER duration done
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on calculated *TRIGGER* duration enabled 1 : Interrupt on calculated *TRIGGER* duration enabled

CDSI_IRQ_EN	
Description	Enable interrupt when calculation of TRIGGER duration done
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

CDSI_IRQ_EN	
RW-Coding	0 : No Interrupt on calculated *STATE* duration enabled 1 : Interrupt on calculated *STATE* duration enabled

TORI_IRQ_EN	
Description	TRIGGER out of range interrupt
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt when *TRIGGER* is out of range enabled 1 : Interrupt when *TRIGGER* is out of range enabled

SORI_IRQ_EN	
Description	STATE out of range
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt when *STATE* is out of range enabled 1 : Interrupt when *STATE* is out of range enabled

DCGI_IRQ_EN	
Description	Direction change interrupt
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

DCGI_IRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt when a direction change of *TRIGGER* is detected 1 : Interrupt when a direction change of *TRIGGER* is detected

21.11.19 DPLL_IRQ_FORCINT

Description	Force Interrupt Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_IRQ_FORCINT
Address	$0x8048$
C-Name	GTM.CLS[0].DPLL.IRQ_FORCINT

Interface: MCS[i]

Name	DPLL_IRQ_FORCINT
Address	$0x8048$
C-Name	

TRG_PDI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.PDI by software.
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.PDI**

TRG_PEI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.PEI by software.
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.PEI**

TRG_TINI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TINI by software.
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TINI**

TRG_TAXI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TAXI by software.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0

TRG_TAXI	
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TAXI**

TRG_SISI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.SISI by software.
Loop	-
Bit Range	$[4:4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.SISI**

TRG_TISI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TISI by software.
Loop	-
Bit Range	$[5:5]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TISI**

TRG_MSI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.MSI by software.
Loop	-

TRG_MSI	
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.MSI**

TRG_MTI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.MTI by software.
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.MTI**

TRG_SASI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.SASI by software.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)

TRG_SASI	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.SASI**

TRG_TASI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TASI by software.
Loop	-
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TASI**

TRG_PWI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.PWI by software.
Loop	-
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.PWI**

TRG_W2I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.W2I by software.
Loop	-

TRG_W2I	
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.W2I**

TRG_W1I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.W1I by software.
Loop	–
Bit Range	[12 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.W1I**

TRG_GL1I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.GL1I by software.
Loop	–
Bit Range	[13 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)

TRG_GL1I	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.GL1I**

TRG_LL1I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.LL1I by software.
Loop	-
Bit Range	[14 : 14]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.LL1I**

TRG_EI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.EI by software.
Loop	-
Bit Range	[15 : 15]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.EI**

TRG_GL2I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.GL2I by software.
Loop	-

TRG_GL2I	
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The corresponding interrupt is not forced
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.GL2I**

TRG_LL2I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.LL2I by software.
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.LL2I**

TRG_TE0I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TE0I by software.
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)

TRG_TE0I	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TE0I**

TRG_TE1I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TE1I by software.
Loop	-
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TE1I**

TRG_TE2I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TE2I by software.
Loop	-
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TE2I**

TRG_TE3I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TE3I by software.
Loop	-

TRG_TE3I	
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TE2I**

TRG_TE4I	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TE4I by software.
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TE4I**

TRG_CDTI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.CDTI by software.
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)

TRG_CDTI	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.CDTI**

TRG_CDSI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.CDSI by software.
Loop	-
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.CDSI**

TRG_TORI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.TORI by software.
Loop	-
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.TORI**

TRG_SORI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.SORI by software.
Loop	-

TRG_SORI	
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.SORI**

TRG_DCGI	
Description	Trigger the bit DPLL_IRQ_NOTIFY.DCGI by software.
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[0]_AEI_ARB_WDATA, 27, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **DPLL_IRQ_NOTIFY.DCGI**

Note:
These bits are cleared automatically after write.

Note:
These bits are write protected by bit **GTM_CTRL.RF_PROT**

21.11.20 DPLL_IRQ_MODE

Description	Interrupt Request Mode
Loop	
Condition	NDPLL > 0
Storage Type	REGISTER
Clock Active Cond	DPLL_CLK_ENABLE == 1

Interface: CPU

Name	DPLL_IRQ_MODE
Address	$0x804C$
C-Name	GTM.CLS[0].DPLL.IRQ_MODE

Interface: MCS[i]

Name	DPLL_IRQ_MODE
Address	$0x804C$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	-
Bit Range	$[1:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in chapter *3.12 "GTM-IP Interrupt Concept"* .

21.11.21 DPLL_EIRQ_EN

Description	Error Interrupt Enable Register
Loop	
Condition	$NDPLL > 0$
Storage Type	REGISTER
Clock Active Cond	$DPLL\_CLK\_ENABLE == 1$

Interface: CPU

Name	DPLL_EIRQ_EN
Address	$0x8050$
C-Name	GTM.CLS[0].DPLL.EIRQ_EN

Interface: MCS[i]

Name	DPLL_EIRQ_EN
Address	$0x8050$
C-Name	

PDI_EIRQ_EN	
Description	DPLL disable interrupt enable, when switch off of the DPLL_CTRL_1.DEN bit.
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The DPLL disable interrupt is not enabled 1 : The DPLL disable interrupt is enabled

PEI_EIRQ_EN	
Description	Enabling of DPLL interrupt enable, while switching on the DPLL_CTRL_1.DEN bit.
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The DPLL enable interrupt is not enabled 1 : The DPLL enable interrupt is enabled

TINI_EIRQ_EN	
Description	TRIGGER minimum hold time violation interrupt enable bit.
Loop	-
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

TINI_EIRQ_EN	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Minimum hold time violation of *TRIGGER* interrupt is not enabled 1 : The minimum hold time violation of *TRIGGER* interrupt is enabled

TAXI_EIRQ_EN	
Description	TRIGGER maximum hold time violation interrupt enable bit.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Maximum hold time violation of *TRIGGER* interrupt is not enabled 1 : The maximum hold time violation of *TRIGGER* interrupt is enabled

SISI_EIRQ_EN	
Description	STATE inactive slope interrupt enable bit.
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The interrupt at the inactive slope of *STATE* is not enabled 1 : The interrupt at the inactive slope of *STATE* is enabled

TISI_EIRQ_EN	
Description	TRIGGER inactive slope interrupt enable bit.
Loop	–

TISI_EIRQ_EN	
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The interrupt at the inactive slope of *TRIGGER* is not enabled 1 : The interrupt at the inactive slope of *TRIGGER* is enabled

MSI_EIRQ_EN	
Description	Missing STATE interrupt enable.
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The missing *STATE* interrupt is not enabled 1 : The missing *STATE* interrupt is enabled

MTI_EIRQ_EN	
Description	Missing TRIGGER interrupt enable.
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

MTI_EIRQ_EN	
RW-Coding	0 : The missing *TRIGGER* interrupt is not enabled 1 : The missing *TRIGGER* interrupt is enabled

SASI_EIRQ_EN	
Description	STATE active slope interrupt enable.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The active slope *STATE* interrupt is not enabled. 1 : The active slope *STATE* interrupt is enabled

TASI_EIRQ_EN	
Description	TRIGGER active slope interrupt enable.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The active slope *TRIGGER* interrupt is not enabled 1 : The active slope *TRIGGER* interrupt is enabled

PWI_EIRQ_EN	
Description	Plausibility window (PVT) violation interrupt of TRIGGER enable.
Loop	–
Bit Range	[10 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

PWI_EIRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The plausibility violation interrupt is not enabled 1 : The plausibility violation interrupt is enabled

W2I_EIRQ_EN	
Description	RAM write access to RAM region 2 interrupt enable.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The RAM write access interrupt is not enabled 1 : The RAM write access interrupt is enabled

W1I_EIRQ_EN	
Description	Write access to RAM region 1b or 1c interrupt.
Loop	–
Bit Range	[12 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The RAM write access interrupt is not enabled 1 : The RAM write access interrupt is enabled.

GL1I_EIRQ_EN	
Description	Get lock interrupt enable, when lock is available.
Loop	–

GL1I_EIRQ_EN	
Bit Range	[13 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The lock getting interrupt is not enabled 1 : The lock getting interrupt is enabled

LL1I_EIRQ_EN	
Description	Loss of lock interrupt enable.
Loop	–
Bit Range	[14 : 14]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The lock loss interrupt is not enabled 1 : The lock loss interrupt is enabled

EI_EIRQ_EN	
Description	Error interrupt enable (see status register).
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

EI_EIRQ_EN	
RW-Coding	0 : The error interrupt is not enabled 1 : The error interrupt is enabled

GL2I_EIRQ_EN	
Description	Get lock interrupt enable for SUB_INC2.
Loop	-
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The lock getting interrupt is not requested 1 : The lock getting interrupt is requested

LL2I_EIRQ_EN	
Description	Loss of lock interrupt enable for SUB_INC2.
Loop	-
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The lock loss interrupt is not requested 1 : The lock loss interrupt is requested

TE0I_EIRQ_EN	
Description	TRIGGER event interrupt 0 enable.
Loop	-
Bit Range	[18 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

TE0I_EIRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 0 enabled 1 : Interrupt on *TRIGGER* event 0 enabled

TE1I_EIRQ_EN	
Description	TRIGGER event interrupt 1 enable.
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 1 enabled 1 : Interrupt on *TRIGGER* event 1 enabled

TE2I_EIRQ_EN	
Description	TRIGGER event interrupt 2 enable.
Loop	–
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 2 enabled 1 : Interrupt on *TRIGGER* event 2 enabled

TE3I_EIRQ_EN	
Description	TRIGGER event interrupt 3 enable.
Loop	–

TE3I_EIRQ_EN	
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 3 enabled 1 : Interrupt on *TRIGGER* event 3 enabled

TE4I_EIRQ_EN	
Description	TRIGGER event interrupt 4 enable.
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on *TRIGGER* event 4 enabled 1 : Interrupt on *TRIGGER* event 4 enabled

CDTI_EIRQ_EN	
Description	Enable interrupt when calculation of TRIGGER duration done
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

CDTI_EIRQ_EN	
RW-Coding	0 : No Interrupt on calculated *TRIGGER* duration enabled 1 : Interrupt on calculated *TRIGGER* duration enabled

CDSI_EIRQ_EN	
Description	Enable interrupt when calculation of TRIGGER duration done
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt on calculated *STATE* duration enabled 1 : Interrupt on calculated *STATE* duration enabled

TORI_EIRQ_EN	
Description	TRIGGER out of range interrupt
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt when *TRIGGER* is out of range enabled 1 : Interrupt when *TRIGGER* is out of range enabled

SORI_EIRQ_EN	
Description	STATE out of range
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

SORI_EIRQ_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt when *STATE* is out of range enabled 1 : Interrupt when *STATE* is out of range enabled

DCGI_EIRQ_EN	
Description	Direction change interrupt
Loop	–
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No Interrupt when a direction change of *TRIGGER* is detected 1 : Interrupt when a direction change of *TRIGGER* is detected

21.11.22 DPLL_INC_CNT1

Description	Counter Value of Sent SUB_INC1 Pulses
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_INC_CNT1
Address	$0x80B0$
C-Name	GTM.CLS[0].DPLL.INC_CNT1

Interface: MCS[i]

Name	DPLL_INC_CNT1
Address	$0x80B0$
C-Name	

INC_CNT1	
Description	Actual number of pulses yet to be sent out at the current increment until the next active input signal in automatic end mode;
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Automatic addition of the number of demanded pulses **DPLL_CTRL_0.MLT** / **DPLL_MLS1.MLS1** when getting an active *TRIGGER* / *STATE* input in normal or emergency mode respectively when **DPLL_CTRL_1.SGE1** =1; writeable only for test purposes when **DPLL_CTRL_1.DEN** =0 In case there is a change in the direction, the wrong number of pulses is corrected twice: Add the difference between **DPLL_NMB_T.NMB_T** and **DPLL_INC_CNT1.INC_CNT1** twice to **DPLL_INC_CNT1.INC_CNT1** before sending out the correction pulses. **Note:** This value can only be written when the DPLL is disabled.

21.11.23 DPLL_INC_CNT2

Description	Counter Value of sent SUB_INC2 values (for DPLL_CTRL_1.SMC=1 and DPLL_CTRL_0.RMO=1)
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_INC_CNT2
Address	$0x80B4$
C-Name	GTM.CLS[0].DPLL.INC_CNT2

Interface: MCS[i]

Name	DPLL_INC_CNT2
Address	$0x80B4$
C-Name	

INC_CNT2	
Description	Actual number of pulses yet to be sent out at the current increment until the next active input signal in automatic end mode;
Loop	–

INC_CNT2	
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Automatic addition of the number of demanded pulses **DPLL_MLS2.MLS2** when getting an active *TRIGGER / STATE* input in normal or emergency mode respectively when **DPLL_CTRL_1.SGE2** =1; writeable only for test purposes when **DPLL_CTRL_1.DEN** =0 In case there is a change in the direction, the wrong number of pulses is corrected twice: Add the difference between **DPLL_NMB_S.NMB_S** and **DPLL_INC_CNT2.INC_CNT2** twice to **DPLL_INC_CNT2.INC_CNT2** before sending out the correction pulses. **Note:** This value can only be written when the DPLL is disabled.

21.11.24 DPLL_APT_SYNC

Description	TRIGGER Time Stamp Field Offset at Synchronization Time
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APT_SYNC
Address	$0x80B8$
C-Name	GTM.CLS[0].DPLL.APT_SYNC

Interface: MCS[i]

Name	DPLL_APT_SYNC
Address	$0x80B8$
C-Name	

APT_2B_EXT	
Description	Address pointer 2b extension; this offset value determines, by which value the DPLL_APT.APT_2B is changed at the synchronization time; set by CPU before the synchronization is performed.
Loop	–
Bit Range	$[5:0]$
Access Type	RW

APT_2B_EXT	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This offset value is the number of virtual increments to be inserted in the TSF for an imminent intended synchronization; the CPU sets its value depending on the gaps until the synchronization time considering the **DPLL_NUTC.NUTE** value to be set and including the future increment (when **DPLL_NUTC.SYN_T_OLD** is still 1). When the synchronization takes place, this value is to be added to the **DPLL_APT.APT_2B** address pointer (for forward direction, *DIR1* =0) and the **DPLL_APT_SYNC.APT_2B_STATUS** bit is cleared after it. For backward direction subtract **DPLL_APT_SYNC.APT_2B_EXT** accordingly. This correction is done after updating the RAM **DPLL_TSF_S[p]** with the last **DPLL_TS_T** value. **Note:** When the synchronization is intended and the **DPLL_NUTC.NUTE** value is to be set to FULL_SCALE after it, the **DPLL_APT_SYNC.APT_2B_EXT** value must be set to 2* **DPLL_CTRL_1.SYN_NT** in order to fill all gaps in the extended **DPLL_TSF_T[p].TSF_T** with the corresponding values by the CPU. When all values for FULL_SCALE are still not available, the **DPLL_APT_SYNC.APT_2B_EXT** value considers only a share according to the corresponding **DPLL_NUTC.NUTE** value to be set after the synchronization.

APT_2B_STATUS	
Description	Address pointer 2b status; set by the CPU before the synchronization is performed. The value is cleared when the DPLL_APT_SYNC.APT_2B_OLD value is written.
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : **DPLL_APT_SYNC.APT_2B_EXT** is not to be considered. 1 : **DPLL_APT_SYNC.APT_2B_EXT** has to be considered for time stamp field extension.

APT_2B_OLD	
Description	Address pointer TRIGGER for RAM region 2b at synchronization time; this value is set by the current DPLL_APT.APT_2B value when the synchronization takes place for the first active TRIGGER event after writing DPLL_APT_2C.APT_2C but before adding the offset value DPLL_APT_SYNC.APT_2B_EXT (that means: when DPLL_APT_SYNC.APT_2B_STATUS=1).
Loop	–
Bit Range	[23 : 14]
Access Type	RW
Volatile	True

APT_2B_OLD	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Address pointer **DPLL_APT.APT_2B** value at the time of synchronization, before the offset value is added, that means the pointer with this value points to the last value before the additional inserted gap

21.11.25 DPLL_APS_SYNC

Description	STATE Time Stamp Field Offset at Synchronization Time
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APS_SYNC
Address	$0x80BC$
C-Name	GTM.CLS[0].DPLL.APS_SYNC

Interface: MCS[i]

Name	DPLL_APS_SYNC
Address	$0x80BC$
C-Name	

APS_1C2_EXT	
Description	Address pointer 1c2 extension; this offset value determines, by which value the DPLL_APS.APS_1C2 is changed at the time of synchronization; set by CPU before the synchronization is performed.
Loop	–
Bit Range	$[5:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

APS_1C2_EXT	
	This offset value is the number of virtual increments to be inserted in the **DPLL_TSF_S[p]** for an imminent intended synchronization; the CPU sets its value depending on the gaps until the synchronization time considering the **DPLL_NUSC.NUSE** value to be set and including the future increment (when **DPLL_NUSC.SYN_S_OLD** is still 1). When the synchronization takes place, this value is to be added to the **DPLL_APS.APS_1C2** address pointer (for forward direction, *DIR2* =0) and the **DPLL_APS_SYNC.APS_1C2_STATUS** bit is cleared after it. For backward direction subtract **DPLL_APS_SYNC.APS_1C2_EXT** accordingly. **Note:** When the synchronization is intended and the **DPLL_NUSC.NUSE** value is to be set to FULL_SCALE after it, the **DPLL_APS_SYNC.APS_1C2_EXT** value must be set to **DPLL_CTRL_1.SYN_NS** (for **DPLL_CTRL_1.SYSF** =1) or 2* **DPLL_CTRL_1.SYN_NS** (for **DPLL_CTRL_1.SYSF** =0) in order to be able to fill all gaps in the extended **DPLL_TSF_S[p].TSF_S** with the corresponding values by the CPU. When all values for FULL_SCALE are still not available, the **DPLL_APS_SYNC.APS_1C2_EXT** value considers only a share according to the **DPLL_NUSC.NUSE** value to be set after the synchronization.

APS_1C2_STATUS	
Description	Address pointer 1c2 status; set by CPU before the synchronization is performed. The value is cleared automatically when the DPLL_APS_SYNC.APS_1C2_OLD value is written.
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : **DPLL_APS_SYNC.APS_1C2_EXT** is not to be considered. 1 : **DPLL_APS_SYNC.APS_1C2_EXT** has to be considered for time stamp field extension.

APS_1C2_OLD	
Description	Address pointer STATE for RAM region 1c2 at synchronization time; this value is set by the current DPLL_APS_APS_1C2 value when the synchronization takes place for the first active STATE event after writing DPLL_APS_1C3.APS_1C3 but before adding the offset value DPLL_APS_SYNC.APS_1C2_EXT (that means: when DPLL_APS_SYNC.APS_1C2_STATUS=1).
Loop	–
Bit Range	[19 : 14]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

APS_1C2_OLD	
	Address pointer **DPLL_APS.APS_1C2** value at the moment of synchronization, before the offset value is added, that means the pointer with this value points to the last value before the additional inserted gap

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is not set. If **DPLL_CTRL_11.STATE_EXT** is set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is set any read access to this register will return *AEI_STATUS* = 0b00 and the data values are equal to: **DPLL_APS_SYNC.APS_1C2_EXT** to **DPLL_APS_SYNC_EXT.APS_1C2_EXT** , **DPLL_APS_SYNC.APS_1C2_STATUS** to **DPLL_APS_SYNC_EXT.APS_1C2_STATUS** , **DPLL_APS_SYNC.APS_1C2_OLD** to **DPLL_APS_SYNC_EXT.APS_1C2_OLD** .

21.11.26 DPLL_TBU_TS0_T

Description	Time Stamp Value for the last active TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TBU_TS0_T
Address	$0x80C0$
C-Name	GTM.CLS[0].DPLL.TBU_TS0_T

Interface: MCS[i]

Name	DPLL_TBU_TS0_T
Address	$0x80C0$
C-Name	

DPLL_TBU_TS0_T	
Description	Value of CCM[0]_TBU_TS0 at the last TRIGGER event;
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

DPLL_TBU_TS0_T	
	for each *T_VALID* the value of *CCM[0]_TBU_TS0* is stored in this register; the register is writeable only for test purposes when **DPLL_CTRL_1.DEN** =0. **Note:** This value can only be written when the DPLL is disabled.

21.11.27 DPLL_TBU_TS0_S

Description	Time Stamp Value for the last active STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TBU_TS0_S
Address	$0x80C4$
C-Name	GTM.CLS[0].DPLL.TBU_TS0_S

Interface: MCS[i]

Name	DPLL_TBU_TS0_S
Address	$0x80C4$
C-Name	

DPLL_TBU_TS0_S	
Description	Value of CCM[0]_TBU_TS0 at the last STATE event;
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	for each *S_VALID* the value of *CCM[0]_TBU_TS0* is stored in this register; the register is writeable only for test purposes when **DPLL_CTRL_1.DEN** =0. **Note:** This value can only be written when the DPLL is disabled.

21.11.28 DPLL_ADD_IN_LD1

Description	ADD_IN Value in Direct Load Mode for TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_ADD_IN_LD1
Address	$0x80C8$
C-Name	GTM.CLS[0].DPLL.ADD_IN_LD1

Interface: MCS[i]

Name	DPLL_ADD_IN_LD1
Address	$0x80C8$
C-Name	

ADD_IN_LD1	
Description	Input value for SUB_INC1 generation, given by CPU. This value can be used in normal and emergency mode (DPLL_CTRL_1.SMC=0) as well as for DPLL_CTRL_1.SMC=1.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	For **DPLL_CTRL_1.DLM1** = 1: The value is loaded by the CPU but used by the DPLL only for **DPLL_CTRL_1.DLM1** =1. When switching **DPLL_CTRL_1.DLM1** to 1, the value in the register is used for the *SUB_INC1* generation beginning from the next active *TRIGGER* or *STATE* event respectively independent of whether the new values are written by the CPU or not. When a new value is written the output frequency changes according to the given value beginning immediately from the time of writing. Do not wait for performing step 10 in the state machine for *ADD_IN* calculations. If the **DPLL_ADD_IN_LD1.ADD_IN_LD1** value is zero all pulses are sent with the highest possible frequency. For **DPLL_CTRL_1.DLM1** = 0: The value loaded by the CPU is stored directly in the internal add_in register which is used to control the sub increment pulse generator directly (**DPLL_CTRL_1.DLM1** = 0). ▽

ADD_IN_LD1	
	△ When a new **DPLL_ADD_IN_LD1.ADD_IN_LD1** value is written the output frequency is immediately changed from the time of writing. The ADD_IN values of the DPLL calculated internally are also written to the internal ADD_IN register. At present, when the internal calculation of the ADD_IN values writes the results in the internal ADD_IN register of the pulse generator, the internally calculated ADD_IN values always have higher priority compared to the values written via the **DPLL_ADD_IN_LD1.ADD_IN_LD1** register. If the **DPLL_ADD_IN_LD1.ADD_IN_LD1** value is zero all pulses are sent with the highest possible frequency.

21.11.29 DPLL_ADD_IN_LD2

Description	ADD_IN Value in Direct Load Mode for STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_ADD_IN_LD2
Address	$0x80CC$
C-Name	GTM.CLS[0].DPLL.ADD_IN_LD2

Interface: MCS[i]

Name	DPLL_ADD_IN_LD2
Address	$0x80CC$
C-Name	

ADD_IN_LD2	
Description	ADD_IN_LD2: Input value for SUB_INC2 generation, given by CPU. This value can be used for DPLL_CTRL_1.SMC=1 while DPLL_CTRL_0.RMO=1.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

ADD_IN_LD2	
	For **DPLL_CTRL_1_SHADOW_STATE.DLM2** = 1: The value is loaded by the CPU but used by the DPLL only for **DPLL_CTRL_1_SHADOW_STATE.DLM2** =1. When switching **DPLL_CTRL_1_SHADOW_STATE.DLM2** to 1, the value in the register is used for the *SUB_INC2* generation beginning with the next *STATE* event respectively independent of whether the new values are written by the CPU or not. When a new value is written the output frequency changes according to the given value beginning immediately from the time of writing. Do not wait for performing step 30 in the state machine for ADD_IN calculations. If the **DPLL_ADD_IN_LD2.ADD_IN_LD2** value is zero all pulses are sent with the highest possible frequency. For **DPLL_CTRL_1_SHADOW_STATE.DLM2** = 0: The value loaded by the CPU is stored directly in the internal add_in register which is used to control the sub increment pulse generator directly (**DPLL_CTRL_1_SHADOW_STATE.DLM2** = 0). When a new **DPLL_ADD_IN_LD2.ADD_IN_LD2** value is written the output frequency is immediately changed from the time of writing. The ADD_IN values of the DPLL calculated internally are also written to the internal ADD_IN register. At present, when the internal calculation of the ADD_IN values writes the results in the internal ADD_IN register of the pulse generator, the internally calculated ADD_IN values always have higher priority compared to the values written via the **DPLL_ADD_IN_LD2.ADD_IN_LD2** register. If the **DPLL_ADD_IN_LD2.ADD_IN_LD2** value is zero all pulses are sent with the highest possible frequency.

21.11.30 DPLL_STATUS

Description	Status Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_STATUS
Address	$0x80FC$
C-Name	GTM.CLS[0].DPLL.STATUS

Interface: MCS[i]

Name	DPLL_STATUS
Address	$0x80FC$
C-Name	

FPCE	
Description	Fast pulse correction error
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-

FPCE	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No error at fast pulse correction detected 1 : Negative value of **DPLL_MPVAL1.MPVAL1** / **DPLL_MPVAL2.MPVAL2** used for fast pulse correction mode
W-Coding	0 : No action 1 : Clear status bit to zero

CSO	
Description	Calculated STATE duration overflow.
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No overflow at equations *[eq_41]* or *[eq_47]* 1 : Overflow at equations *[eq_41]* or *[eq_47]*
W-Coding	0 : No action 1 : Clear status bit to zero
	Bit is set when equations *[eq_41]* or *[eq_47]* lead to an overflow.

CTO	
Description	Calculated TRIGGER duration overflow.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–

CTO	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No overflow at equation *[eq_17]* or *[eq_24]* 1 : Overflow at equation *[eq_17]* or *[eq_24]*
W-Coding	0 : No action 1 : Clear status bit to zero
	Bit is set when equations *[eq_17]* or *[eq_24]* lead to an overflow. **Note:** When one of the above bits is set the corresponding register contains the maximum value 0xFFFFFF.

CRO	
Description	Calculated Reciprocal value overflow.
Loop	-
Bit Range	$[4:4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No overflow at any reciprocal calculation 1 : Overflow for at least one reciprocal calculation
W-Coding	0 : No action 1 : Clear status bit to zero
	Bit is set when the calculation of **DPLL_RDT_T_ACT.RDT_T_ACT** or **DPLL_RDT_S_ACT.RDT_S_ACT** leads to an overflow. **Note:** An overflow in calculation of reciprocal values can occur, when the condition of note at **DPLL_CTRL_0.RMO** is violated. Such an overflow can occur according to the calculations in equations *[eq_11]* or *[eq_35]* . The overflow is detected when after the calculation and shifting left 32 bits at least one of the bits 31 to 24 is not zero. In that case the corresponding register is set to 0xFFFFFF.

RCS	
Description	Resolution conflict STATE.
Loop	-
Bit Range	$[5:5]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-

RCS	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No resolution conflict detected 1 : The **DPLL_CTRL_1.TS0_HRS** value is set to 1 while *LOW_RES* =0
	Bit is reset, when condition is changed accordingly

RCT	
Description	Resolution conflict TRIGGER.
Loop	–
Bit Range	[6 : 6]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No resolution conflict detected 1 : The **DPLL_CTRL_1.TS0_HRT** value is set to 1 while *LOW_RES* =0
	Bit is reset, when condition is changed accordingly

PSE	
Description	Prediction space configuration error
Loop	–
Bit Range	[7 : 7]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_REST: sync

PSE	
R-Coding	0 : No prediction space error detected 1 : Configured offset value of RAM2 is too small in order to store all **DPLL_CTRL_0.TNU** +1 values twice in FULL_SCALE

SOR	
Description	STATE out of range
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : All *STATE* signal events appear within **DPLL_SLR.SLR** interval or a direction change was detected 1 : At least one *STATE* signal event is out of **DPLL_SLR.SLR** ;
W-Coding	0 : No action 1 : Clear status bit to zero

MS	
Description	Missing STATE detected according to DPLL_SOV.
Loop	–
Bit Range	[9 : 9]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No missing *STATE* detected or a new active *STATE* slope occurred 1 : At least one missing *STATE* detected after the last active slope
W-Coding	0 : No action 1 : Clear status bit to zero

TOR	
Description	TRIGGER out of range
Loop	–

TOR	
Bit Range	[10 : 10]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : All *TRIGGER* signal events occur within **DPLL_TLR.TLR** interval or a direction change was detected 1 : At least one *TRIGGER* signal event is out of **DPLL_TLR.TLR** ;
W-Coding	0 : No action 1 : Clear status bit to zero

MT	
Description	Missing TRIGGER detected according to DPLL_TOV
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No missing *TRIGGER* detected or a new active *TRIGGER* slope occurred 1 : At least one missing *TRIGGER* detected after the last active slope
W-Coding	0 : No action 1 : Clear status bit to zero

RAM2_ERR	
Description	DPLL internal access to not configured RAM2 memory space
Loop	–
Bit Range	[12 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–

RAM2_ERR	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : No access to not configured RAM2 memory space 1 : Access to not configured RAM2 memory space
W-Coding	0 : No action 1 : Clear status bit to zero

LOW_RES	
Description	Low resolution of CCM[0]_TBU_TS0 is used for DPLL input; this value reflects the input signal LOW_RES
Loop	–
Bit Range	[15 : 15]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : The lower 24 Bits of *CCM[0]_TBU_TS0* are used as input for the DPLL 1 : The higher 24 Bits of *CCM[0]_TBU_TS0* are used as input for the DPLL

CSVS	
Description	Current signal value STATE
Loop	–
Bit Range	[16 : 16]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : The last *STATE_S* value was 0 1 : The last *STATE_S* value was 1

CSVT	
Description	Current signal value TRIGGER
Loop	–
Bit Range	[17 : 17]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : The last *TRIGGER_S* value was 0 1 : The last *TRIGGER_S* value was 1

CAIP2	
Description	Calculation of upper half Actions in progress
Loop	–
Bit Range	[18 : 18]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : Currently no action calculation, new data requests possible 1 : Action calculation in progress, no new data requests possible

CAIP1	
Description	Calculation of lower half Actions in progress
Loop	–
Bit Range	[19 : 19]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

CAIP1	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : Currently no Action calculation, new data requests possible 1 : Action calculation in progress, no new data requests possible

ISN	
Description	Incrementing number of STATE events is not plausible; Bit is set when the number of states is different from profile
Loop	–
Bit Range	[20 : 20]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : The number of *STATE* events between synchronization gaps is plausible, a direction change is detected or the DPLL_APS_1C3_1C3 pointer is written 1 : After setting **DPLL_STATUS.LOCK1** in emergency mode (**DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0.RMO** =1) or **DPLL_STATUS.LOCK2** for **DPLL_CTRL_1.SMC** = **DPLL_CTRL_0.RMO** =1 missing or additional *STATE* signals detected; bit is cleared when a direction change is detected or the DPLL_APS_1C3_1C3 is written

ITN	
Description	Incrementing number of TRIGGER events is not plausible; Bit is set when the number of TRIGGERS is different from profile
Loop	–
Bit Range	[21 : 21]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync

ITN	
R-Coding	0 : The number of *TRIGGER* events between synchronization gaps is plausible, a direction change is detected or the address pointer **DPLL_APT_2C.APT_2C** is written 1 : After setting **DPLL_STATUS.LOCK1** in normal mode (for **DPLL_CTRL_1.SMC** =0 or **DPLL_CTRL_1.SMC** =1) or in emergency mode (only for **DPLL_CTRL_1.SMC** =0) for missing or additional *TRIGGER* signals detected; bit is cleared when a direction change is detected or the **DPLL_APT_2C.APT_2C** is written

BWD2	
Description	Backward drive of SUB_INC2
Loop	–
Bit Range	[22 : 22]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : forward direction 1 : backward direction

BWD1	
Description	Backwards drive of SUB_INC1
Loop	–
Bit Range	[23 : 23]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : forward direction 1 : backward direction

LOCK2	
Description	DPLL Lock status concerning SUB_INC2
Loop	–
Bit Range	[25 : 25]
Access Type	R

LOCK2	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : The DPLL is not locked concerning *STATE* for **DPLL_CTRL_1.SMC** =1 1 : The DPLL is locked concerning *STATE* for **DPLL_CTRL_1.SMC** =1
	Note: Locking of *SUB_INC2* appears for **DPLL_CTRL_0.RMO** = **DPLL_CTRL_1.SMC** =1: Bit is set, when **DPLL_STATUS.SYS** is set and the number of events between two missing *STATE* ports is as expected by the DPLL_NUSC_SYN_S values. **Note:** **DPLL_STATUS.LOCK2** is set for **DPLL_CTRL_1.SMC** = **DPLL_CTRL_0.RMO** =1: for an active *STATE* event when **DPLL_STATUS.SYS** is set and DPLL_SYN_NS=0 or when **DPLL_STATUS.SYS** is set and the profile stored in the ADT_Si field matches once between two gaps. **DPLL_STATUS.LOCK2** is reset: for **DPLL_CTRL_1.SMC** = **DPLL_CTRL_0.RMO** =1 – when a missing *STATE* event occurs while DPLL_NUSC_SYN_S=1. This means an unexpected missing *STATE* . – when the corresponding input signal *STATE* is out of locking range **DPLL_SLR.SLR**

SYS	
Description	Synchronization condition of STATE fixed.
Loop	–
Bit Range	[26 : 26]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync DPLL_RESET: sync
R-Coding	0 : Synchronization condition of *STATE* not set 1 : Synchronization condition of *STATE* set
	This bit is set when the CPU writes to the DPLL_APS_1C3_1C3 address pointer.

SYT	
Description	Synchronization condition of TRIGGER fixed.
Loop	–

SYT	
Bit Range	[27 : 27]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : Synchronization condition of *TRIGGER* not set 1 : Synchronization condition of *TRIGGER* set
	This bit is set when the CPU writes to the **DPLL_APT_2C.APT_2C** address pointer.

FSD	
Description	First STATE detected.
Loop	–
Bit Range	[28 : 28]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : Still no active *STATE* event was detected after enabling DPLL 1 : At least one active *STATE* event was detected after enabling DPLL
	Note: No change of **DPLL_STATUS.FSD** for switching from normal to emergency mode or vice versa.

FTD	
Description	First TRIGGER detected.
Loop	–
Bit Range	[29 : 29]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

FTD	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No active *TRIGGER* event was detected after enabling DPLL 1 : At least one active *TRIGGER* event was detected after enabling DPLL
	Note: No change of **DPLL_STATUS.FTD** for switching from normal to emergency mode or vice versa.

LOCK1	
Description	DPLL Lock status concerning SUB_INC1.
Loop	–
Bit Range	[30 : 30]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The DPLL is not locked for *TRIGGER* (while **DPLL_CTRL_1.SMC** = **DPLL_CTRL_0.RMO** =0 or **DPLL_CTRL_1.SMC** =1) or for *STATE* (while **DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0.RMO** =1) 1 : The DPLL is locked for *TRIGGER* (while **DPLL_CTRL_1.SMC** = **DPLL_CTRL_0.RMO** =0 or **DPLL_CTRL_1.SMC** =1) or for *STATE* (while **DPLL_CTRL_1.SMC** =0 and **DPLL_CTRL_0.RMO** =1)
	Note: ▽

LOCK1	
	△ **DPLL_STATUS.LOCK1** is set : – in normal mode (for **DPLL_CTRL_0.RMO** = **DPLL_CTRL_1.SMC** =0, **DPLL_CTRL_1.LCD** =0): Bit is set for an active *TRIGGER* event when **DPLL_STATUS.SYT** is set and the number of events between two gaps is as expected by the profile (**DPLL_ADT_T[p].NT** values) or when **DPLL_CTRL_1.SYN_NT** =0 and **DPLL_STATUS.SYT** =1. – in normal mode (for **DPLL_CTRL_0.RMO** = **DPLL_CTRL_1.SMC** =0, **DPLL_CTRL_1.LCD** =1): Bit is set for an active *TRIGGER* event when **DPLL_STATUS.SYT** is set and the number of events between two increments without missing *TRIGGER* (no gap) is as expected by the profile (**DPLL_ADT_T[p].NT** values). – in emergency mode (for **DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =0): Bit is set for an active *STATE* event, when **DPLL_STATUS.SYS** is set and the received events are corresponding to the profile (**DPLL_ADT_S[p].NS** values) for at least two (four in case of direction change) expected missing *STATE* events or when DPLL_SYN_NS=0. – for **DPLL_CTRL_1.SMC** =1: Bit is set for an active *TRIGGER* even when **DPLL_STATUS.SYT** is set and **DPLL_CTRL_1.SYN_NT** =0 or when **DPLL_STATUS.SYT** is set and the profile stored in the **DPLL_ADT_T[p]** field matches once between two gaps. **DPLL_STATUS.LOCK1** is reset for **DPLL_CTRL_0.RMO** = **DPLL_CTRL_1.SMC** =0: – when a corresponding missing *TRIGGER* event occurs while **DPLL_NUTC.SYN_T** =1. This means an unexpected missing *TRIGGER* . – when the corresponding input signal *TRIGGER* is out of locking range **DPLL_TLR.TLR** , – when a corresponding direction change is detected for **DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =0: – when a corresponding missing *STATE* event occurs while DPLL_NUSC_SYN_S=1. This means an unexpected missing *STATE* . – when the corresponding input signal *STATE* is out of locking range **DPLL_TLR.TLR** for **DPLL_CTRL_1.SMC** =1: – when a corresponding missing *TRIGGER* event occurs while **DPLL_NUTC.SYN_T** =1. This means an unexpected missing *TRIGGER* . – when the corresponding input signal *TRIGGER* is out of locking range **DPLL_TLR.TLR** , – when a corresponding direction change is detected

ERR	
Description	Error during configuration or operation resulting in unexpected values.
Loop	–
Bit Range	[31 : 31]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : when all bits in position [8:0] and [10:10] and [12:12] are zero 1 : when at least one bit in position [8:0] and [10:10] and [12:12] is one
	Note: The **DPLL_STATUS.SOR** bit is set, when the time from the last valid *STATE* event to the next active *STATE* slope exceeds the value of the last nominal *STATE* duration multiplied with the value of the **DPLL_SLR.SLR** register (see chapter **DPLL_SLR**) and is reset, when at the current or last active input event a direction change was detected. The **DPLL_STATUS.SYS** bit is not influenced by setting the **DPLL_STATUS.SOR** bit. **Note:** The **DPLL_STATUS.TOR** bit is set, when the time from the last valid *TRIGGER* event to the next active *TRIGGER* slope exceeds the value of the last nominal *TRIGGER* duration multiplied with the value of the **DPLL_TLR.TLR** register (see chapter **DPLL_TLR**) and is reset, when at the current or last active input event a direction change was detected. The **DPLL_STATUS.SYT** bit is not influenced by setting the **DPLL_STATUS.TOR** bit

Note:
The **DPLL_STATUS** register is reset, when the DPLL is disabled (switching **DPLL_CTRL_1.DEN** from 1 to 0).

21.11.31 DPLL_ID_PMTR_[n]

Description	ID Information for Input Signal PMTR[n] (Position minus Time Request)
Loop	$n = \{n : 0 <= n <= \text{NOAC} - 1\}$
Condition	$\text{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\text{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_ID_PMTR_[n]
Address	$0x4 * n + 0x8100$
C-Name	GTM.CLS[0].DPLL.ID_PMTR[n]

Interface: MCS[i]

Name	DPLL_ID_PMTR_[n]
Address	$0x4 * n + 0x8100$
C-Name	

ID_PMTR	
Description	ID information to the input signal PMTR[n] from the ARU.
Loop	-
Bit Range	$[8 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x1FE$
Protect Enable Cond	DPLL_ACTION_ENABLE[n] == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Note: This value can only be written when the Action [n] is disabled by the corresponding signal *AEN[n]* =0. Where *AEN[n]* is constructed out of the concatenation of the bitfields **DPLL_CTRL_2.AEN[n]** , (n={0, 1,..., 7}) and **DPLL_CTRL_3.AEN[n]** , (n={8, 9,...,15}) and **DPLL_CTRL_4.AEN[n]** , (n={16, 17,..., 23}) and **DPLL_CTRL_5.AEN[n]** , (n={24, 25,..., 31}). or when the DPLL is disabled (**DPLL_CTRL_1.DEN** =0).

21.11.32 DPLL_CTRL_0_SHADOW_TRIGGER

Description	Shadow Register of DPLL_CTRL_0 controlled by an active TRIGGER Slope
Loop	
Condition	$\text{NDPLL} > 0$

Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_0_SHADOW_TRIGGER
Address	$0x81E0$
C-Name	GTM.CLS[0].DPLL.CTRL_0_SHADOW_TRIGGER

Interface: MCS[i]

Name	DPLL_CTRL_0_SHADOW_TRIGGER
Address	$0x81E0$
C-Name	

MLT	
Description	Multiplier for TRIGGER; MLT+1 is number of SUB_INC1 pulses between two TRIGGER events in normal mode (1...1024);
Loop	-
Bit Range	$[9:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x257$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

IFP	
Description	Input filter position; value contains position or time related information.
Loop	-
Bit Range	$[10:10]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

IFP	
R-Coding	0 : **DPLL_FTV_T.TRIGGER_FT** and **DPLL_FTV_S.STATE_FT** mean time related values, that means the number of time stamp clocks 1 : **DPLL_FTV_T.TRIGGER_FT** and **DPLL_FTV_S.STATE_FT** mean position related values, that means the number of *SUB_INC1* (or *SUB_INC2* in the case **DPLL_CTRL_1.SMC** =1) pulses respectively

AMT	
Description	Use of adaptation information of Adapt mode TRIGGER;
Loop	–
Bit Range	[26 : 26]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No adaptation information for *TRIGGER* is used 1 : Immediate adapting mode; the values for physical deviation **DPLL_ADT_T[p].PD** are considered to calculate the *SUB_INC1* pulses in normal mode and for **DPLL_CTRL_1.SMC** =1

IDT	
Description	Input delay TRIGGER; use of input delay information transmitted in FT part of the TRIGGER signal.
Loop	–
Bit Range	[28 : 28]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : Delay information is not used 1 : Up to 24 bits of the FT part contain the delay value of the input signal, concerning the corresponding edge

RMO	
Description	Reference mode; selection of the relevant the input signal for generation of SUB_INC1.
Loop	–
Bit Range	[31 : 31]
Access Type	R
Volatile	True
Multithread	False

RMO	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : Normal mode; the signal *TRIGGER* is used to generate the *SUB_INC1* signals 1 : Emergency mode for **DPLL_CTRL_1.SMC** =0; signal *STATE* is used to generate the *SUB_INC1* signals ; Double synchronous mode for **DPLL_CTRL_1.SMC** =1: signal *TRIGGER* is used to generate the *SUB_INC1* signals and *STATE* is used to generate the *SUB_INC2* signals
	Note: The coding is applicable if **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** = **DPLL_CTRL_0_SHADOW_STATE.RMO** . If not (they are unequal) the **DPLL_CTRL_0.RMO** defines which input is evaluated: *STATE* is selected if **DPLL_CTRL_0.RMO** = 1 else *TRIGGER* is selected.

Note:
The values characterized by **DPLL_CTRL_0_SHADOW_TRIGGER.MLT** , **DPLL_CTRL_0.IFP** , **DPLL_CTRL_0.AMT** , **DPLL_CTRL_0.IDT** , and **DPLL_CTRL_0.RMO** are stored to **DPLL_CTRL_0_SHADOW_TRIGGER** for an active *TRIGGER* slope. All other values remain 0. When **DPLL_CTRL_1.DEN** =0 the relevant bit values of the original register **DPLL_CTRL_0** are transferred without any input event at the next system clock. This results in the above reset value.

21.11.33 DPLL_CTRL_0_SHADOW_STATE

Description	Shadow Register of DPLL_CTRL_0 controlled by an active STATE Slope
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_0_SHADOW_STATE
Address	$0x81E4$
C-Name	GTM.CLS[0].DPLL.CTRL_0_SHADOW_STATE

Interface: MCS[i]

Name	DPLL_CTRL_0_SHADOW_STATE
Address	$0x81E4$
C-Name	

IFP	
Description	Input filter position; value contains position or time related information.
Loop	–
Bit Range	$[10 : 10]$
Access Type	R
Volatile	True
Multithread	False

IFP	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : **DPLL_FTV_T.TRIGGER_FT** and **DPLL_FTV_S.STATE_FT** mean time related values, that means the number of time stamp clocks 1 : **DPLL_FTV_T.TRIGGER_FT** and **DPLL_FTV_S.STATE_FT** mean position related values, that means the number of *SUB_INC1* (or *SUB_INC2* in the case **DPLL_CTRL_1.SMC** =1) pulses respectively

AMS	
Description	Adapt mode STATE; Use of adaptation information of STATE.
Loop	–
Bit Range	[25 : 25]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No adaptation information is used for *STATE* 1 : Immediate adapting mode; the values for physical deviation **DPLL_ADT_S[p].PD_S** are considered to calculate *SUB_INC1* pulses in emergency mode (**DPLL_CTRL_1.SMC** =0) or *SUB_INC2* pulses for **DPLL_CTRL_1.SMC** =1

IDS	
Description	Input delay STATE; Use of input delay information transmitted in FT part of the STATE signal.
Loop	–
Bit Range	[27 : 27]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : Delay information is not used 1 : Up to 24 bits of the FT part contain the delay value of the input signal, concerning the corresponding edge

RMO	
Description	Reference mode; selection of the relevant the input signal for generation of SUB_INC1.
Loop	–
Bit Range	[31 : 31]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : Normal mode; the signal *TRIGGER* is used to generate the *SUB_INC1* signals 1 : Emergency mode for **DPLL_CTRL_1.SMC** =0; signal *STATE* is used to generate the *SUB_INC1* signals ; Double synchronous mode for **DPLL_CTRL_1.SMC** =1: signal *TRIGGER* is used to generate the *SUB_INC1* signals and *STATE* is used to generate the *SUB_INC2* signals
	Note: The coding is applicable if **DPLL_CTRL_0_SHADOW_TRIGGER.RMO** = **DPLL_CTRL_0_SHADOW_STATE.RMO** . If not (they are unequal) the **DPLL_CTRL_0.RMO** defines which input is evaluated: *STATE* is selected if **DPLL_CTRL_0.RMO** = 1 else *TRIGGER* is selected.

Note:

The values characterized by **DPLL_CTRL_0.IFP** , **DPLL_CTRL_0.AMS** , **DPLL_CTRL_0.IDS** , and **DPLL_CTRL_0.RMO** are stored to **DPLL_CTRL_0_SHADOW_STATE** for an active *STATE* slope. All other values remain 0. When **DPLL_CTRL_1.DEN** =0 the relevant bit values of the original register **DPLL_CTRL_0** are transferred without any input event at the next system clock.

21.11.34 DPLL_CTRL_1_SHADOW_TRIGGER

Description	Shadow Register of DPLL_CTRL_1 controlled by an active TRIGGER Slope
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_1_SHADOW_TRIGGER
Address	$0x81E8$
C-Name	GTM.CLS[0].DPLL.CTRL_1_SHADOW_TRIGGER

Interface: MCS[i]

Name	DPLL_CTRL_1_SHADOW_TRIGGER
Address	$0x81E8$
C-Name	

DMO	
Description	DPLL mode select.

DMO	
Loop	–
Bit Range	[0 : 0]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : Automatic end mode; if the number of pulses for an increment is reached, no further pulse is generated until the next active *TRIGGER* / *STATE* is received; in the case of getting a new active *TRIGGER* / *STATE* before the defined number of pulses is reached, the pulse frequency is changed according to the conditions described below (**DPLL_CTRL_1_SHADOW_TRIGGER.COA**). 1 : Continuous mode; in this mode a difference between the predefined number of pulses and the actual number of generated pulses can influence the pulse frequency by writing a corresponding pulse number into **DPLL_CNT_NUM_1.CNT_NUM_1** or **DPLL_CNT_NUM_2.CNT_NUM_2** respectively in RAM region 1b.

COA	
Description	Correction strategy in automatic end mode (DPLL_CTRL_1.DMO=0).
Loop	–
Bit Range	[3 : 3]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The pulse frequency of the *CCM[0]_CLK_RES* [0:0] will be used to make up for missing pulses from last increment; the output of the calculated new pulses will start after resetting the storage elements in the pulse generation unit. The frequency of *CCM[0]_CLK_RES* [0:0] should not exceed half the frequency of the system clock (see chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*). 1 : Missing pulses of the last increment are distributed evenly to the next increment, calculations are done when the next active input event appears. The number of missing sub-pulses will be determined by the pulse counter difference between the last two active *TRIGGER* / *STATE* events respectively; the storage elements in the pulse generation unit are not reset before sending new pulses.

PIT	
Description	Plausibility value PVT to next active TRIGGER is time related
Loop	–
Bit Range	[4 : 4]
Access Type	R
Volatile	True
Multithread	False

PIT	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The plausibility value is position related (**DPLL_PVT.PVT** contains the number of *SUB_INC1* pulses) 1 : The plausibility value is time related (the **DPLL_PVT.PVT** value is to be multiplied with the duration of the last increment **DPLL_DT_T_ACT.DT_T_ACT** and divided by 1024)

SGE1	
Description	SUB_INC1 generator enable.
Loop	–
Bit Range	[5 : 5]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The *SUB_INC1* generator is not enabled 1 : The *SUB_INC1* generator is enabled

DLM1	
Description	Direct Load Mode for SUB_INC1 generation
Loop	–
Bit Range	[6 : 6]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The DPLL uses the calculated **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** value for the *SUB_INC1* generation 1 : The **DPLL_ADD_IN_LD1.ADD_IN_LD1** value is used for the *SUB_INC1* generation and is provided by the CPU; the value remains valid until the CPU writes a new one; the calculated ADD_IN values are stored as **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** in the RAM at different locations for normal and emergency mode

PCM1	
Description	Pulse Correction Mode for SUB_INC1 generation.
Loop	–
Bit Range	$[7:7]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The DPLL does not use the correction value stored in **DPLL_MPVAL1.MPVAL1** 1 : The DPLL uses the correction value stored in **DPLL_MPVAL1.MPVAL1** in normal and emergency mode

Note:
The values characterized by **DPLL_CTRL_1.DMO** , **DPLL_CTRL_1.COA** , **DPLL_CTRL_1.PIT** , **DPLL_CTRL_1.SGE1** , **DPLL_CTRL_1.DLM1** , and **DPLL_CTRL_1.PCM1** are stored for an active *TRIGGER* slope to **DPLL_CTRL_1_SHADOW_TRIGGER** . All other values remain 0. When **DPLL_CTRL_1.DEN** =0 the relevant bit values of the original register **DPLL_CTRL_1** are transferred without any input event at the next system clock.

21.11.35 DPLL_CTRL_1_SHADOW_STATE

Description	DPLL Shadow Register of DPLL_CTRL_1 controlled by an active STATE Slope
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_1_SHADOW_STATE
Address	$0x81EC$
C-Name	GTM.CLS[0].DPLL.CTRL_1_SHADOW_STATE

Interface: MCS[i]

Name	DPLL_CTRL_1_SHADOW_STATE
Address	$0x81EC$
C-Name	

DMO	
Description	DPLL mode select.
Loop	–
Bit Range	$[0:0]$
Access Type	R
Volatile	True

DMO	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : Automatic end mode; if the number of pulses for an increment is reached, no further pulse is generated until the next active *TRIGGER* / *STATE* is received; in the case of getting a new active *TRIGGER* / *STATE* before the defined number of pulses is reached, the pulse frequency is changed according to the conditions described below (**DPLL_CTRL_1_SHADOW_STATE.COA**). 1 : Continuous mode; in this mode a difference between the predefined number of pulses and the actual number of generated pulses can influence the pulse frequency by writing a corresponding pulse number into **DPLL_CNT_NUM_1.CNT_NUM_1** or **DPLL_CNT_NUM_2.CNT_NUM_2** respectively in RAM region 1b.

COA	
Description	Correction strategy in automatic end mode (DPLL_CTRL_1.DMO=0).
Loop	–
Bit Range	[3 : 3]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The pulse frequency of the *CCM[0]_CLK_RES* [0:0] will be used to make up for missing pulses from last increment; the output of the calculated new pulses will start after resetting the storage elements in the pulse generation unit. The frequency of *CCM[0]_CLK_RES* [0:0] should not exceed half the frequency of the system clock (see chapter *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*). 1 : Missing pulses of the last increment are distributed evenly to the next increment, calculations are done when the next active input event appears. The number of missing sub-pulses will be determined by the pulse counter difference between the last two active *TRIGGER* / *STATE* events respectively; the storage elements in the pulse generation unit are not reset before sending new pulses.

SGE1	
Description	SUB_INC1 generator enable.
Loop	–
Bit Range	[5 : 5]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

SGE1	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The *SUB_INC1* generator is not enabled 1 : The *SUB_INC1* generator is enabled

DLM1	
Description	Direct Load Mode for SUB_INC1 generation
Loop	–
Bit Range	[6 : 6]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The DPLL uses the calculated **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** value for the *SUB_INC1* generation 1 : The **DPLL_ADD_IN_LD1.ADD_IN_LD1** value is used for the *SUB_INC1* generation and is provided by the CPU; the value remains valid until the CPU writes a new one; the calculated ADD_IN values are stored as **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** in the RAM at different locations for normal and emergency mode

PCM1	
Description	Pulse Correction Mode for SUB_INC1 generation.
Loop	–
Bit Range	[7 : 7]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The DPLL does not use the correction value stored in **DPLL_MPVAL1.MPVAL1** 1 : The DPLL uses the correction value stored in **DPLL_MPVAL1.MPVAL1** in normal and emergency mode

SGE2	
Description	SUB_INC2 generator enable.
Loop	–
Bit Range	[8 : 8]
Access Type	R

SGE2	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The *SUB_INC2* generator is not enabled. 1 : The *SUB_INC2* generator is enabled

DLM2	
Description	Direct Load Mode for SUB_INC2 generation
Loop	–
Bit Range	[9 : 9]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The DPLL uses the calculated **DPLL_ADD_IN_CAL2.ADD_IN_CAL2** value for the *SUB_INC2* generation. 1 : The **DPLL_ADD_IN_LD2.ADD_IN_LD2** value is used for the *SUB_INC2* generation and is provided by the CPU; the value remains valid until the CPU writes a new one; the calculated ADD_IN values are stored as **DPLL_ADD_IN_CAL2.ADD_IN_CAL2** in the RAM at different locations for normal and emergency mode

PCM2	
Description	Pulse Correction Mode for SUB_INC2 generation.
Loop	–
Bit Range	[10 : 10]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

PCM2	
R-Coding	0 : The DPLL does not use the correction value stored in **DPLL_MPVAL2.MPVAL2** . 1 : The DPLL uses the correction value stored in **DPLL_MPVAL2.MPVAL2**

Note:

Only the values characterized by **DPLL_CTRL_1.DMO** , **DPLL_CTRL_1.COA** , **DPLL_CTRL_1.SGE1** , **DPLL_CTRL_1.DLM1** , **DPLL_CTRL_1.-PCM1** , **DPLL_CTRL_1.SGE2** , **DPLL_CTRL_1.DLM2** , and **DPLL_CTRL_1.PCM2** are stored for an active *STATE* slope to **DPLL_CTRL_1_S-HADOW_STATE** . All other values remain 0. When **DPLL_CTRL_1.DEN** =0 the relevant bit values of the original register **DPLL_CTRL_1** are transferred without any input event at the next system clock.

21.11.36 DPLL_RAM_INI

Description	Register to control the RAM Initialization
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RAM_INI
Address	$0x81FC$
C-Name	GTM.CLS[0].DPLL.RAM_INI

Interface: MCS[i]

Name	DPLL_RAM_INI
Address	$0x81FC$
C-Name	

INIT_1A	
Description	RAM region 1a initialization in progress
Loop	-
Bit Range	$[0:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No initialization of considered RAM region in progress 1 : Initialization of considered RAM region in progress

INIT_1BC	
Description	RAM region 1b and 1c initialization in progress
Loop	-

INIT_1BC	
Bit Range	[1 : 1]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No initialization of considered RAM region in progress 1 : Initialization of considered RAM region in progress

INIT_2	
Description	RAM region 2 initialization in progress
Loop	–
Bit Range	[2 : 2]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : No initialization of considered RAM region in progress 1 : Initialization of considered RAM region in progress

INIT_RAM	
Description	RAM regions 1a, 1b and 2 are to be initialized.
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

INIT_RAM	
W-Coding	0 : No Action 1 : Start initialization of all RAM regions
R-Coding	0 : No RAM reset is active 1 : DPLL currently resets RAM content
	Note: Setting the **DPLL_RAM_INI.INIT_RAM** bit results only in a RAM reset when the DPLL is not enabled (**DPLL_CTRL_1.DEN** =0). **Note:** Depending on the vendor configuration the connected RAM regions are initialized to zero in the case of a module HW reset or for setting the **GTM_RST.RST** bit. **Note:** In the case of no RAM initialization it must be ensured that all relevant parameters are configured correctly. Otherwise there is no guarantee to get a predictable behavior.

21.11.37 DPLL_TSAC[n]

Description	Calculated Time Value to start Action [n]
Loop	$n = \{n : 0 <= n <= \mathrm{NOAC} - 1\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TSAC[n]
Address	$0x4 * n + 0x8E00$
C-Name	GTM.CLS[0].DPLL.TSAC[n]

Interface: MCS[i]

Name	DPLL_TSAC[n]
Address	$0x4 * n + 0x8E00$
C-Name	

TSAC	
Description	Calculated time stamp for Action [n]
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x7FFFFF$
Protect Enable Cond	DPLL_CTRL_1.DEN == 1

TSAC	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Note: This value can only be written when the DPLL is disabled.

21.11.38 DPLL_PSAC[n]

Description	Calculated Position Value to start Action [n]
Loop	$n = \{n : 0 <= n <= \text{NOAC} - 1\}$
Condition	$\text{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\text{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PSAC[n]
Address	$0x4 * n + 0x8E80$
C-Name	GTM.CLS[0].DPLL.PSAC[n]

Interface: MCS[i]

Name	DPLL_PSAC[n]
Address	$0x4 * n + 0x8E80$
C-Name	

PSAC	
Description	Calculated position value for the start of Action [n].
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0x7FFFFF$
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	In normal or emergency mode according to equations *[eq_162]* , *[eq_165]* , *[eq_168]* , *[eq_171]* or *[eq_177]* , *[eq_180]* , *[eq_183]* , *[eq_186]* respectively. **Note:** This value can only be written when the DPLL is disabled.

21.11.39 DPLL_ACB_[n]

Description	Control Bits for NOAC Actions
Loop	$n = \{n : 0 <= n <= \text{ceil}(\text{NOAC}/4) - 1\}$
Condition	$\text{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\text{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_ACB_[n]
Address	$0x4 * n + 0x8F00$
C-Name	GTM.CLS[0].DPLL.ACB[n]

Interface: MCS[i]

Name	DPLL_ACB_[n]
Address	$0x4 * n + 0x8F00$
C-Name	

ACB_[m]	
Description	Action Control Bits of Action [n*4+m]
Loop	$m = \{n : 0 <= n <= 3\}$
Bit Range	$[m * 8 + 4 : m * 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	DPLL_CTRL_1.DEN == 1
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Note: When **DPLL_CTRL_11.ACBU** = '0': **DPLL_ACB_[n].ACB_[m]** [4:0] are taken as received by ARU interface and are transmitted unchanged as result of Action (PMT) calculation. When **DPLL_CTRL_11.ACBU** = '1': **DPLL_ACB_[n].ACB_[m]** [4:2] are taken as received by ARU interface and are transmitted unchanged as result of Action (PMT) calculation. **DPLL_ACB_[n].ACB_[m]** [1:1]= '1' is used as input signal to control if "Action in past" shall be checked based on position information. **DPLL_ACB_[n].ACB_[m]** [1:1] is written to '1' if Action channel has reached "Action in past" condition after Action has been calculated, written to '0' if Action has not reached "past" so far. **DPLL_ACB_[n].ACB_[m]** [0:0] is used as input signal to control if "Action in past" shall be checked based on time information. **DPLL_ACB_[n].ACB_[m]** [0:0] is written to '1' if Action channel has reached "Action in past" condition after Action has been calculated, written to '0' if Action has not reached "past" so far. **Note:** This value can only be written via AEI-interface when the DPLL is disabled.

BOSCH

21.11.40 DPLL_CTRL_11

Description	Control Register 11
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_11
Address	$0x8F20$
C-Name	GTM.CLS[0].DPLL.CTRL_11

Interface: MCS[i]

Name	DPLL_CTRL_11
Address	$0x8F20$
C-Name	

SIP1	
Description	Simplified increment prediction in normal mode and for the first engine in the case DPLL_CTRL_1.SMC=1.
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 16, 16) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Increment prediction calculation; the current increment duration **DPLL_CDT_TX.CDT_TX** is calculated using the relation between increment duration in the past like explained by the corresponding equations. 1 : Increment prediction continuation; in this mode for the increment prediction value calculation of **DPLL_CDT_TX.CDT_TX** the value of QDT_T is replaced by 1 for all calculations when **DPLL_NUTC.NUTE** - **DPLL_NUTC.VTN** =1; in the other case the value of **DPLL_CTRL_11.SIP1** is ignored and the calculation is performed like for **DPLL_CTRL_11.SIP1** =0.
	Note: For the first increment after setting **DPLL_CTRL_11.SIP1** from 0 to 1 the value of **DPLL_DT_T_ACT.DT_T_ACT** is replaced by the value of the **DPLL_DT_T_START.DT_T_START** bitfield. This results in a **DPLL_CDT_TX.CDT_TX** value which is equal to **DPLL_DT_T_START.DT_T_START** . Please notice that this **DPLL_DT_T_START.DT_T_START** value must be always > 256. **Note:** The value of **DPLL_CTRL_11.SIP1** influences only the increment prediction **DPLL_CDT_TX.CDT_TX** and when **DPLL_NUTC.NUTE** - **DPLL_NUTC.VTN** =1. The calculation of QDT_T itself is not influenced by the **DPLL_CTRL_11.SIP1** bit. The value of **DPLL_CTRL_11.SIP1** can be only written when **DPLL_CTRL_11.WSIP1** =1. ▽

SIP1	
	△ **Note:** When **DPLL_CTRL_11.SIP1** = 1 is set the first pulses of the subincrement generator are not generated with highest frequency for the first increment (**DPLL_STATUS.FTD** = 0, **DPLL_CTRL_1.SGE1** = 1).

ERZ1	
Description	Error is assumed as zero in normal mode and for the first engine for DPLL_CTRL_1.SMC=1.
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 17, 17) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The **DPLL_MEDT_T.MEDT_T** value is considered as provided in the corresponding equations. 1 : Instead of using **DPLL_MEDT_T.MEDT_T** the value "0" is used in the corresponding equations.
	Note: The calculation of **DPLL_EDT_T.EDT_T** and **DPLL_MEDT_T.MEDT_T** is performed independent of the **DPLL_CTRL_11.ERZ1** value in all modes without any influence on the **DPLL_MEDT_T.MEDT_T** value itself. The **DPLL_CTRL_11.ERZ1** value influences the use of **DPLL_MEDT_T.MEDT_T** in normal mode and for **DPLL_CTRL_1.SMC** =1. The value of **DPLL_CTRL_11.ERZ1** can be only written when **DPLL_CTRL_11.WERZ1** =1.

PCMF1	
Description	Pulse correction mode fast for DPLL_INC_CNT1.INC_CNT1
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 18, 18) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No fast update of pulses, provided by **DPLL_MPVAL1.MPVAL1** . 1 : When **DPLL_CTRL_1.PCM1** is set while **DPLL_CTRL_11.PCMF1** =1, the pulses provided by **DPLL_MPVAL1.MPVAL1** are sent using the rapid pulse generator RPCUx without waiting for a new input event.

PCMF1	
	Note: The fast pulse generation is performed immediately within the current increment. **DPLL_MPVAL1.MPVAL1** must be positive integers for the fast pulse correction mode – in the case of negative values the correction is suppressed and the **DPLL_STATUS.FPCE** (fast pulse correction error) bit is set, causing the **DPLL_IRQ_NOTIFY.EI** (error interrupt) when enabled. The setting of **DPLL_CTRL_11.PCMF1** prevents the transfer of control bits **DPLL_CTRL_1.PCM1** to the corresponding shadow registers with an active input event and prevents therefore the distribution of the **DPLL_MPVAL1.MPVAL1** values over the current or next increment. The **DPLL_MPVAL1.MPVAL1** pulses are sent with the fast clock *CCM[0]_CLK_RES* [0:0] by the rapid pulse generator RPCUx triggered in the state 6/26 or 18/38 of the state machines respectively. The **DPLL_INC_CNT1.INC_CNT1** is incremented by **DPLL_MPVAL1.MPVAL1** respectively. When the **DPLL_MPVAL1.MPVAL1** value is stored to RPCUx and added to **DPLL_INC_CNT1.INC_CNT1** the **DPLL_CTRL_1.PCM1** bit is reset immediately. The value of **DPLL_CTRL_11.PCMF1** can be only written when **DPLL_CTRL_11.WPCMF1** =1. Be careful when using the fast pulse correction during a change in direction. The result is typically unpredictable, because the correction pulses are sent before, during or after the recognition of the change in direction. No automatic correction of the fast correction pulses is provided. The necessary corrections must be performed on the user's responsibility.

FSYL1	
Description	Force Synchronization Loss of DPLL_STATUS.LOCK1.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 19, 19) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : No force of synchronization loss. 1 : Reset **DPLL_STATUS.LOCK1** , and reset **DPLL_STATUS.SYT** (in normal mode and for **DPLL_CTRL_1.SMC** =1) or reset **DPLL_STATUS.SYS** (in emergency mode)
	Note: The synchronization loss resets **DPLL_STATUS.SYT** / **DPLL_STATUS.SYS** and prevents the use of profiles respectively. The value of **DPLL_CTRL_11.FSYL1** can be only written when **DPLL_CTRL_11.WFSYL1** =1.

INCF1	
Description	DPLL_INC_CNT1.INC_CNT1 fast correction
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 20, 20) == 0

INCF1	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The calculation of a new **DPLL_INC_CNT1.INC_CNT1** is performed after an active slope was detected and the plausibility check was performed. 1 : The calculation of a new **DPLL_INC_CNT1.INC_CNT1** is prepared before an active slope is detected; the plausibility check is supported by an additional HW checker in order to get the decision earlier and after this decision the pulse generator for *SUB_INC1* starts immediately sending out pulses The calculation of *ADD_IN* for the *SUB_INC1* and *SUB_INC2* generation is performed without adding the 0.5 value to **DPLL_NMB_T.NMB_T** / **DPLL_NMB_S.NMB_S** in equations (*[eq_193]* , *[eq_194]* , *[eq_196]* , *[eq_197]* , *[eq_201]* , *[eq_202]* , *[eq_203]* , *[eq_204]*). The signal *RESET_SIG[1]* of the pulse generator (see *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*) is activated for each new active input slope in order to reset the register values.
	Note: The **DPLL_CTRL_11.INCF1** value can be only written when **DPLL_CTRL_11.WINCF1** =1. The **DPLL_CTRL_11.INCF1** bit should only be written when **DPLL_CTRL_1.DEN** = '0' (DPLL disabled) to prevent generation of wrong number of sub increments.

PCMF1_INCCNT_B	
Description	No increment of DPLL_INC_CNT1.INC_CNT1 when DPLL_CTRL_11.PCMF1 active (automatic end mode).
Loop	-
Bit Range	[5 : 5]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 21, 21) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Add **DPLL_MPVAL1.MPVAL1** value is as well to the **DPLL_INC_CNT1.INC_CNT1** when fast pulse correction is done by **DPLL_CTRL_1.PCM1** or **DPLL_CTRL_11.PCMF1** . 1 : Do not add **DPLL_MPVAL1.MPVAL1** value to the **DPLL_INC_CNT1.INC_CNT1** register when fast pulse correction is done by **DPLL_CTRL_1.PCM1** or **DPLL_CTRL_11.PCMF1** . This means that just fast pulses are done by decrementing current content of **DPLL_INC_CNT1.INC_CNT1** register as long as **DPLL_INC_CNT1.INC_CNT1** is not zero (automatic end mode). The number of pulses (**DPLL_MPVAL1.MPVAL1**) shall be sufficiently smaller than **DPLL_INC_CNT1.INC_CNT1** when **DPLL_MPVAL1.MPVAL1** is written.
	Note: The **DPLL_CTRL_11.PCMF1_INCCNT_B** value can be only written when **DPLL_CTRL_11.WPCMF1_INCCNT_B** =1.

ADT	
Description	Correction of DPLL_DT_T_ACT.DT_T_ACT, DPLL_CDT_TX_NOM.CDT_TX_NOM_corr by PD_T
Loop	-
Bit Range	[6 : 6]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-

ADT	
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 22, 22) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No correction of **DPLL_DT_T_ACT.DT_T_ACT** , **DPLL_CDT_TX_NOM.CDT_TX_NOM** _corr by physical deviation (PD_T) defined in profile of *TRIGGER* processing unit. 1 : Correction of **DPLL_DT_T_ACT.DT_T_ACT** , **DPLL_CDT_TX_NOM.CDT_TX_NOM** _corr by physical deviation (PD_T) defined in profile of *TRIGGER* processing unit.

ADS	
Description	Correction of DPLL_DT_S_ACT.DT_S_ACT, DPLL_CDT_SX_NOM.CDT_SX_NOM_corr by DPLL_ADT_S[p].PD_S
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 23, 23) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No correction of **DPLL_DT_S_ACT.DT_S_ACT** , **DPLL_CDT_SX_NOM.CDT_SX_NOM** _corr by physical deviation (**DPLL_ADT_S[p].PD_S**) defined in profile of *STATE* processing unit. 1 : Correction of **DPLL_DT_S_ACT.DT_S_ACT** , **DPLL_CDT_SX_NOM.CDT_SX_NOM** _corr by physical deviation (**DPLL_ADT_S[p].PD_S**) defined in profile of *STATE* processing unit.

SIP2	
Description	Simplified increment prediction in emergency mode and for the second engine in the case DPLL_CTRL_0.RMO=1.
Loop	–
Bit Range	[8 : 8]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 24, 24) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

SIP2	
RW-Coding	0 : Increment prediction calculation; the current increment duration **DPLL_CDT_SX.CDT_SX** is calculated using the relation between increments duration in the past like explained by the corresponding equations. 1 : Increment prediction continuation; in this mode for the increment prediction value calculation **DPLL_CDT_SX.CDT_SX** the value of QDT_S is replaced by 1 for all calculations when DPLL_NUSC_NUSE-DPLL_NUSC_VSN=1; in the other case the value of **DPLL_CTRL_11.SIP2** is ignored and the calculation is performed like for **DPLL_CTRL_11.SIP2** =0. For the first increment after setting **DPLL_CTRL_11.SIP2** from 0 to 1 the value of **DPLL_DT_S_ACT.DT_S_ACT** is replaced by the value of the **DPLL_DT_S_START.DT_S_START** register. This results in a **DPLL_CDT_SX.CDT_SX** value which is equal to **DPLL_DT_S_START.DT_S_START** . Please notice that this **DPLL_DT_S_START.DT_S_START** value must be always > 256.
	Note: The value of **DPLL_CTRL_11.SIP2** influences only the increment prediction and error accumulation when DPLL_NUSC_NUSE-DPLL_NUSC_VSN=1. The calculation of QDT_S itself is not influenced by the **DPLL_CTRL_11.SIP2** bit. The value of **DPLL_CTRL_11.SIP2** can be only written when **DPLL_CTRL_11.WSIP2** =1.

ERZ2	
Description	Error is assumed as zero in emergency mode and for the second engine for DPLL_CTRL_1.SMC=1.
Loop	-
Bit Range	[9 : 9]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 25, 25) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The **DPLL_MEDT_S.MEDT_S** value is considered as provided in the corresponding equations. 1 : Instead of using **DPLL_MEDT_S.MEDT_S** the value "0" is used in the corresponding equations.
	Note: The calculation of **DPLL_EDT_S.EDT_S** and **DPLL_MEDT_S.MEDT_S** is performed independently from the **DPLL_CTRL_11.ERZ2** value in all modes without any influence on the **DPLL_MEDT_S.MEDT_S** value itself. The **DPLL_CTRL_11.ERZ2** value influences the use of **DPLL_MEDT_S.MEDT_S** in emergency mode and for **DPLL_CTRL_1.SMC** =1 with **DPLL_CTRL_0.RMO** =1. The value of **DPLL_CTRL_11.ERZ2** can be only written when **DPLL_CTRL_11.WERZ2** =1.

PCMF2	
Description	Pulse correction mode fast for DPLL_INC_CNT2.INC_CNT2
Loop	-
Bit Range	[10 : 10]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 26, 26) == 0

PCMF2	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : No fast update of pulses, provided by **DPLL_MPVAL2.MPVAL2** . 1 : When **DPLL_CTRL_1.PCM2** is set while **DPLL_CTRL_11.PCMF2** =1, the pulses provided by **DPLL_MPVAL2.MPVAL2** are sent using the rapid pulse generator RPCUx without waiting for a new input event.
	Note: The fast pulse generation is performed immediately within the current increment. **DPLL_MPVAL2.MPVAL2** must be positive integers for the fast pulse correction mode – in the case of negative values the correction is suppressed and the **DPLL_STATUS.FPCE** (fast pulse correction error) bit is set, causing the **DPLL_IRQ_NOTIFY.EI** (error interrupt) when enabled. The setting of **DPLL_CTRL_11.PCMF2** prevents the transfer of control bits **DPLL_CTRL_1.PCM2** to the corresponding shadow registers with an active input event and prevents therefore the distribution of the **DPLL_MPVAL1.MPVAL1** values over the current or next increment. The **DPLL_MPVAL2.MPVAL2** pulses are sent with the fast clock *CCM[0]_CLK_RES* [0:0] by the rapid pulse generator RPCUx triggered in the state 6/26 or 18/38 of the state machines respectively. The **DPLL_INC_CNT2.INC_CNT2** is incremented by **DPLL_MPVAL2.MPVAL2** respectively. When the **DPLL_MPVAL2.MPVAL2** value is stored to RPCUx and added to **DPLL_INC_CNT2.INC_CNT2** the **DPLL_CTRL_1.PCM2** bit is reset immediately.The value of **DPLL_CTRL_11.PCMF2** can be only written when **DPLL_CTRL_11.WPCMF2** =1. Be careful when using the fast pulse correction during a change in direction. Because of sending the correction pulses before, during or after the direction change recognition the result is typically unpredictable. No automatic correction of the fast correction pulses is provided. The necessary corrections must be performed on the user's responsibility.

FSYL2	
Description	Force Synchronization Loss of DPLL_STATUS.LOCK2.
Loop	–
Bit Range	[11 : 11]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 27, 27) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : no force of synchronization loss. 1 : Reset **DPLL_STATUS.LOCK2** , and reset **DPLL_STATUS.SYS** (in emergency mode and for **DPLL_CTRL_1.SMC** =1).
	Note: The synchronization loss resets **DPLL_STATUS.SYS** and prevents the use of profiles respectively. The value of **DPLL_CTRL_11.FSYL2** can be only written when **DPLL_CTRL_11.WFSYL2** =1.

INCF2	
Description	DPLL_INC_CNT2.INC_CNT2 fast
Loop	–
Bit Range	[12 : 12]
Access Type	RW
Volatile	True

INCF2	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 28, 28) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : the calculation of a new **DPLL_INC_CNT2.INC_CNT2** is performed after an active slope was detected and the plausibility check was performed. 1 : the calculation of a new **DPLL_INC_CNT2.INC_CNT2** is prepared before an active slope is detected; the plausibility check is supported by an additional HW checker in order to get the decision earlier and after this decision the pulse generator for *SUB_INC2* starts immediately sending out pulses. The calculation of *ADD_IN* for the *SUB_INC1* and *SUB_INC2* generation is performed without adding the 0.5 value to **DPLL_NMB_S.NMB_S** in equations (*[eq_196]* , *[eq_197]* , *[eq_203]* , *[eq_204]*). The signal *RESET_SIG[2]* of the pulse generator (see *154 "Adder for generation of SUB_INC1 and SUB_INC2 by the carry c_out. "*) is activated for each new active input slope in order to reset the register values.
	Note: The **DPLL_CTRL_11.INCF2** value can be only written when **DPLL_CTRL_11.WINCF2** =1.

PCMF2_INCCNT_B	
Description	No increment of DPLL_INC_CNT2.INC_CNT2 when DPLL_CTRL_11.PCMF2 active (automatic end mode).
Loop	–
Bit Range	[13 : 13]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 29, 29) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Add **DPLL_MPVAL2.MPVAL2** value is as well added to the **DPLL_INC_CNT2.INC_CNT2** when fast pulse correction is done by **DPLL_CTRL_1.PCM2** or **DPLL_CTRL_11.PCMF2** . 1 : Do not add **DPLL_MPVAL2.MPVAL2** value to the **DPLL_INC_CNT2.INC_CNT2** register when fast pulse correction is done by **DPLL_CTRL_1.PCM2** or **DPLL_CTRL_11.PCMF2** . This means that just fast pulses are done by decrementing current content of **DPLL_INC_CNT2.INC_CNT2** register as long as **DPLL_INC_CNT2.INC_CNT2** is not zero (automatic end mode). The number of pulses (**DPLL_MPVAL2.MPVAL2**) shall be sufficiently smaller than **DPLL_INC_CNT2.INC_CNT2** when **DPLL_MPVAL2.MPVAL2** is written.
	Note: The **DPLL_CTRL_11.PCMF2_INCCNT_B** value can be only written when **DPLL_CTRL_11.WPCMF2_INCCNT_B** =1.

STATE_EXT	
Description	Use of STATE engine extension
Loop	–
Bit Range	[14 : 14]
Access Type	RW

STATE_EXT	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 30, 30) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : *STATE* extension is not considered. 1 : *STATE* extension is enabled up to 128 *STATE* events
	Note: The **DPLL_CTRL_11.STATE_EXT** value can be only written when **DPLL_CTRL_11.WSTATE_EXT** = 1 and the D-PLL is disabled. See chapter *22.2 "MCS to DPLL Interface Description"* for a further explanation. If this bit shall be modified during operation a software reset of the DPLL module is strongly recommended. A RAM initialization should also be considered depending on the given application case.

ACBU	
Description	Use DPLL_ACB_[n].ACB_[m]. The DPLL_ACB_[n].ACB_[m] values of PMTR are used to decide if an Action is in the past
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 31, 31) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The decision if an Action with index n*4+m is in the past is made considering the calculated time value regardless of the value of **DPLL_ACB_[n].ACB_[m]** . 1 : For an Action with index n*4+m, **DPLL_ACB_[n].ACB_[m]** values are considered as follows: if **DPLL_ACB_[n].ACB_[m]** [1:1] = 1, the calculated position value is used to decide whether the Action is in the past. If **DPLL_ACB_[n].ACB_[m]** [0:0] = 1, the calculated time value is used to decide whether the Action is in the past. **DPLL_ACB_[n].ACB_[m]** [1:1] and **DPLL_ACB_[n].ACB_[m]** [0:0] can be set also simultaneously to 1.
	Note: Return **DPLL_ACB_[n].ACB_[m]** values together with Actions as zero, when the Actions are in future; set the ARU Control Bit (ACB) with index 1 to 1, when calculated position value is in the past and the **DPLL_ACB_[n].ACB_[m]** [1:1] of the respective PMTR with index n*4+m was 1. Set the ARU Control Bit (ACB) with index 0 to 1, when calculated time value is in the past and the **DPLL_ACB_[n].ACB_[m]** [0:0] of the respective PMTR with index n*4+m was 1. The value of **DPLL_CTRL_11.ACBU** can be only written when **DPLL_CTRL_11.WACBU** = 1.

WSIP1	
Description	Write enable for simplified increment prediction 1.
Loop	–
Bit Range	[16 : 16]

WSIP1	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : Writing to **DPLL_CTRL_11.SIP1** is not enabled. 1 : Writing to **DPLL_CTRL_11.SIP1** is enabled.

WERZ1	
Description	Write enable for error zero 1.
Loop	–
Bit Range	[17 : 17]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : Writing to **DPLL_CTRL_11.ERZ1** is not enabled. 1 : Writing to **DPLL_CTRL_11.ERZ1** is enabled.

WPCMF1	
Description	Write enable for pulse correction mode fast 1
Loop	–
Bit Range	[18 : 18]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

WPCMF1	
W-Coding	0 : Writing to **DPLL_CTRL_11.PCMF1** is not enabled. 1 : Writing to **DPLL_CTRL_11.PCMF1** is enabled.

WFSYL1	
Description	Write enable for Force Synchronization Loss 1.
Loop	-
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.FSYL1** is not enabled. 1 : Writing to **DPLL_CTRL_11.FSYL1** is enabled.

WINCF1	
Description	Write enable for DPLL_INC_CNT1.INC_CNT1 fast
Loop	-
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.INCF1** is not enabled. 1 : Writing to **DPLL_CTRL_11.INCF1** is enabled.

WPCMF1_INCCNT_B	
Description	Write enable of DPLL_CTRL_11.PCMF1_INCCNT_B
Loop	-
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-

WPCMF1_INCCNT_B	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.PCMF1_INCCNT_B** is not enabled. 1 : Writing to **DPLL_CTRL_11.PCMF1_INCCNT_B** is enabled.

WADT	
Description	Write enable of DPLL_CTRL_11.ADT
Loop	–
Bit Range	[22 : 22]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.ADT** is not enabled. 1 : Writing to **DPLL_CTRL_11.ADT** is enabled.

WADS	
Description	Write enable of DPLL_CTRL_11.ADS
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.ADS** is not enabled. 1 : Writing to **DPLL_CTRL_11.ADS** is enabled.

WSIP2	
Description	Write enable for simplified increment prediction 2.
Loop	–

WSIP2	
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : Writing to **DPLL_CTRL_11.SIP2** is not enabled. 1 : Writing to **DPLL_CTRL_11.SIP2** is enabled.

WERZ2	
Description	Write enable for error zero 2.
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : Writing to **DPLL_CTRL_11.ERZ2** is not enabled 1 : Writing to **DPLL_CTRL_11.ERZ2** is enabled.

WPCMF2	
Description	Write enable for pulse correction mode fast 2
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

WPCMF2	
W-Coding	0 : Writing to **DPLL_CTRL_11.PCMF2** is not enabled 1 : Writing to **DPLL_CTRL_11.PCMF2** is enabled

WFSYL2	
Description	Write enable for Force Synchronization Loss 2.
Loop	-
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.FSYL2** is not enabled. 1 : Writing to **DPLL_CTRL_11.FSYL2** is enabled

WINCF2	
Description	Write enable for DPLL_INC_CNT2.INC_CNT2 fast
Loop	-
Bit Range	[28 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.INCF2** is not enabled 1 : Writing to **DPLL_CTRL_11.INCF2** is enabled.

WPCMF2_INCCNT_B	
Description	Write enable of DPLL_CTRL_11.PCMF2_INCCNT_B
Loop	-
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-

WPCMF2_INCCNT_B	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.PCMF2_INCCNT_B** is not enabled. 1 : Writing to **DPLL_CTRL_11.PCMF2_INCCNT_B** is enabled.

WSTATE_EXT	
Description	Write enable of DPLL_CTRL_11.STATE_EXT
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.STATE_EXT** is not enabled. 1 : Writing to **DPLL_CTRL_11.STATE_EXT** is enabled.

WACBU	
Description	Write enable for DPLL_CTRL_11.ACBU use; the DPLL_CTRL_11.ACBU values of DPLL_ID_PMTR_[n].ID_PMTR are used to decide if an Action is in the past
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_11.ACBU** is not enabled. 1 : Writing to **DPLL_CTRL_11.ACBU** is enabled.

21.11.41 DPLL_THVAL2

Description	Measured TRIGGER Hold Time Value 2

Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_THVAL2
Address	$0x8F24$
C-Name	GTM.CLS[0].DPLL.THVAL2

Interface: MCS[i]

Name	DPLL_THVAL2
Address	$0x8F24$
C-Name	

THVAL	
Description	Measured last pulse time from active to inactive slope of TRIGGER after correction of input slope filter delays
Loop	-
Bit Range	$[23 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Note: This value is available immediately after the inactive slope of *TRIGGER* . The measured value considers all input slope filter delays. From the received input the corresponding filter delays are subtracted before the time stamp difference of active and inactive slope is calculated.

21.11.42 DPLL_TIDEL

Description	TRIGGER input delay
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TIDEL

Address	$0x8F28$
C-Name	GTM.CLS[0].DPLL.TIDEL

Interface: MCS[i]

Name	DPLL_TIDEL
Address	$0x8F28$
C-Name	

TIDEL	
Description	TRIGGER input delay
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Transmit this value with each active *TRIGGER* slope into a shadow register. Subtract this shadow register value from each *TRIGGER* time stamp (active and inactive slope). This feature is always active and cannot be disabled by a control bit.

21.11.43 DPLL_SIDEL

Description	STATE input delay
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_SIDEL
Address	$0x8F2C$
C-Name	GTM.CLS[0].DPLL.SIDEL

Interface: MCS[i]

Name	DPLL_SIDEL
Address	$0x8F2C$
C-Name	

SIDEL	
Description	STATE input delay
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Transmit this value with each active *STATE* slope into a shadow register. Subtract this shadow register value from each *STATE* time stamp (active and inactive slope). This feature is always active and cannot be disabled by a control bit.

21.11.44 DPLL_CTN_MIN

Description	Minimum value of predicted nominal increment of TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTN_MIN
Address	$0x8F6C$
C-Name	GTM.CLS[0].DPLL.CTN_MIN

Interface: MCS[i]

Name	DPLL_CTN_MIN
Address	$0x8F6C$
C-Name	

CTN_MIN	
Description	DPLL_CDT_TX_NOM.CDT_TX_NOM min value
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

CTN_MIN	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Use this register value as **DPLL_CDT_TX_NOM.CDT_TX_NOM** value when the calculated value for the nominal increment prediction of *TRIGGER* is less than the register value

21.11.45 DPLL_CTN_MAX

Description	Maximum value of predicted nominal increment of TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTN_MAX
Address	$0x8F70$
C–Name	GTM.CLS[0].DPLL.CTN_MAX

Interface: MCS[i]

Name	DPLL_CTN_MAX
Address	$0x8F70$
C–Name	

CTN_MAX	
Description	DPLL_CDT_TX_NOM.CDT_TX_NOM max value
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0xFFFFFF$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

CTN_MAX	
	Use this register value as **DPLL_CDT_TX_NOM.CDT_TX_NOM** value when the calculated value for the nominal increment prediction of *TRIGGER* is greater than the register value

21.11.46 DPLL_CSN_MIN

Description	Minimum value of predicted nominal increment of STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CSN_MIN
Address	$0x8F74$
C-Name	GTM.CLS[0].DPLL.CSN_MIN

Interface: MCS[i]

Name	DPLL_CSN_MIN
Address	$0x8F74$
C-Name	

CSN_MIN	
Description	DPLL_CDT_SX_NOM.CDT_SX_NOM min value
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Use this register value as **DPLL_CDT_SX_NOM.CDT_SX_NOM** value when the calculated value for the nominal increment prediction of *STATE* is less than the register value

21.11.47 DPLL_CSN_MAX

Description	Maximum value of predicted nominal increment of STATE
Loop	

Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CSN_MAX
Address	$0x8F78$
C-Name	GTM.CLS[0].DPLL.CSN_MAX

Interface: MCS[i]

Name	DPLL_CSN_MAX
Address	$0x8F78$
C-Name	

CSN_MAX	
Description	DPLL_CDT_SX_NOM.CDT_SX_NOM max value
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$0xFFFFFF$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Use this register value as **DPLL_CDT_SX_NOM.CDT_SX_NOM** value when the calculated value for the nominal increment prediction of *STATE* is greater than the register value

21.11.48 DPLL_STA

Description	Status of the state machine states
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_STA
Address	$0x8F40$

C-Name	GTM.CLS[0].DPLL.STA

Interface: MCS[i]

Name	DPLL_STA
Address	$0x8F40$
C-Name	

STA_T	
Description	Status of TRIGGER state machine; state binary coded
Loop	-
Bit Range	$[7:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0x00 : Reset state 0x01 : Wait, **DPLL_CTRL_1.DEN** =0 0x02 : Calculation of 1/ **DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1, **DPLL_MLS1.MLS1** , **DPLL_MLS2.MLS2** . 0x03 : Calculation of direction change issues (pointers and profile update) 0x04 : **DPLL_APT_2C.APT_2C** is being incremented in normal mode 0x05 : **DPLL_APT_2C.APT_2C** was incremented 3 times in normal mode, 1 in emergency or **DPLL_CTRL_1.SMC** =1 0x06 : **DPLL_APT_2C.APT_2C** was incremented 4 times in normal mode, 2 in emergency or **DPLL_CTRL_1.SMC** =1 0x07 : Update of pointers is finished, perform change of direction operations 0x08 : **DPLL_PVT.PVT** _check 0x09 : Update of RAM: write **DPLL_RDT_T[p].RDT_T** ; **DPLL_DT_T[p].DT_T** ; **DPLL_TSF_T[p].TSF_T** 0x0A : Loading of profile (**DPLL_NUTC.SYN_T** , update **DPLL_NUTC.SYN_T_OLD**) from **DPLL_ADT_T[p]** 0x0B : **DPLL_IRQ_NOTIFY.TASI** , store **DPLL_FTV_S** into RAM1b 0x0C : Write **DPLL_PSTC.PSTC** ; modify **DPLL_APT.APT** , **DPLL_APT.APT_2B** ; **DPLL_APT_2C.APT_2C** (if synchronized); Start fast pulse updates if necessary; Update **DPLL_INC_CNT1.INC_CNT1** ; 0x10 : Write **DPLL_TS_T** to RAM1B, calculate **DPLL_RDT_T_ACT.RDT_T_ACT** 0x18 : Update **DPLL_NTI_CNT.NTI_CNT** , **DPLL_IRQ_NOTIFY.CDTI** if **DPLL_NTI_CNT.NTI_CNT** =0; 0x19 : Calculated **DPLL_EDT_T.EDT_T** , **DPLL_MEDT_T.MEDT_T** , **DPLL_RDT_T_ACT.RDT_T_ACT** 0x20 : Calculate **DPLL_CDT_TX_NOM.CDT_TX_NOM** , **DPLL_CDT_TX.CDT_TX** 0x28 : Calculate **DPLL_PSTM.PSTM** , **DPLL_RDT_T[p].RDT_T** , **DPLL_NMB_T_TAR.NMB_T_TAR** , start fast correction of missing pulses (if necessary) for **DPLL_CTRL_0.RMO** =0 or **DPLL_CTRL_1.SMC** =1, Set **DPLL_STATUS.FPCE** -> **DPLL_IRQ_NOTIFY.EI** (if necessary), Write **DPLL_MP_T.MP_T** 0x30 : Calculate **DPLL_NMB_T.NMB_T** for **DPLL_CTRL_0.RMO** =0 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1.DMO** =0, **DPLL_CTRL_1.COA** =0 0x38 : Calculate **DPLL_NMB_T.NMB_T** for **DPLL_CTRL_0.RMO** =0 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1.DMO** =0, **DPLL_CTRL_1.COA** =1 0x40 : Calculate **DPLL_NMB_T.NMB_T** for **DPLL_CTRL_0.RMO** =0 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1.DMO** =1 0x48 : State not used ▽

STA_T	
	△ 0x50 : Calculate **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** 0x51 : Write of **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** finished, all subincrement calculations done for last active input event 0x58 : Calculate **DPLL_TS_T** _check (**DPLL_IRQ_NOTIFY.MTI**), r_add_caln (prepare time stamp calculation(**DPLL_TS_T**)) for **DPLL_CTRL_0.IDT** = **DPLL_CTRL_0.IFP** =1 0x60 : Set **DPLL_STATUS.CAIP1** , **DPLL_STATUS.CAIP2** , Action masking bits , Action calculation loop control. 0x68 : Calculate **DPLL_NA[n]** , 0x70 : Calculate PDT_T[n] 0x71 : Calculate **DPLL_DTA[n].DTA** 0x78 : Calculate **DPLL_TSAC[n].TSAC** 0x79 : Calculate **DPLL_PSAC[n].PSAC** 0x7A : Action(n) in past condition occured: assignment of output data. 0x7B : Action loop control 0x80 : Wait for new Action calculation 0x90 : Calculate **DPLL_NMB_T_TAR.NMB_T_TAR** ; Set **DPLL_STATUS.FPCE** -> **DPLL_IRQ_NOTIFY.EI** (if necessary); Calculation of fast pulse correction while waiting for new input event or new PMT data. When finished change to "Action Loop control" (0x7B.)
	This bit field reflects the status of the *TRIGGER* state machine **Note:** The decimal step number 1 to 20 of the state machine is binary coded from 0x01 to 0x14 respectively using the upper 5 bits [7:3]. The lower 3 bit [2:0] show substates of the corresponding state machine. When the DPLL is disabled this field is 0x000.

CNT_T	
Description	Count TRIGGER; this reflects the count of active TRIGGER slopes (mod8).
Loop	-
Bit Range	$[11:9]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value shows the number of active *TRIGGER* slopes (mod8) **Note:** This value can be used to distinguish, whether the above status of the state machine is consistent to other status values read before or after it.

STA_S	
Description	Status of STATE state machine; state binary coded
Loop	-
Bit Range	$[19:12]$
Access Type	R

STA_S	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0x00 : Reset state 0x01 : Wait, **DPLL_CTRL_1.DEN** =0 0x02 : Calculation of 1/ **DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1, **DPLL_MLS1.MLS1** , **DPLL_MLS2.MLS2** . 0x03 : Calculation of direction change issues (pointers and profile update) 0x05 : DPLL_APS_1C3_1C3 was incremented once 0x06 : DPLL_APS_1C3_1C3 was incremented twice 0x07 : Update of pointers is finished, perform change of direction operations 0x08 : **DPLL_PVT.PVT** _check 0x09 : Update of RAM: write **DPLL_RDT_S[p].RDT_S** ; **DPLL_DT_S[p].DT_S** ; **DPLL_TSF_S[p].TSF_S** 0x0A : Loading of profile (DPLL_NUSC_SYN_S, update DPLL_NUSC_SYN_S_OLD) from **DPLL_ADT_S[p]** 0x0B : **DPLL_IRQ_NOTIFY.SASI** , store **DPLL_FTV_S** into RAM1b 0x0C : Write **DPLL_PSSC.PSSC** ; modify DPLL_APS_APS, DPLL_APS_1C2; DPLL_APS_1C3_1C3 (if synchronized); Start fast pulse updates if necessary; Update **DPLL_INC_CNT1.INC_CNT1** /2; 0x10 : Write **DPLL_TS_S** to RAM1b, calculate **DPLL_DT_S_ACT.DT_S_ACT** 0x18 : Update **DPLL_IRQ_NOTIFY.CDSI** 0x19 : Calculate **DPLL_EDT_S.EDT_S** , **DPLL_MEDT_S.MEDT_S** , **DPLL_RDT_S_ACT.RDT_S_ACT** 0x20 : Calculate **DPLL_CDT_SX_NOM.CDT_SX_NOM** , **DPLL_CDT_SX.CDT_SX** 0x28 : Calculate **DPLL_PSSM.PSSM** , **DPLL_RCDT_SX.RCDT_SX** , **DPLL_NMB_S_TAR.NMB_S_TAR** , start fast correction of missing pulses (if necessary), Set **DPLL_STATUS.FPCE** -> **DPLL_IRQ_NOTIFY.EI** (if necessary) , Write **DPLL_MP_S.MP_S** . 0x30 : Calculate **DPLL_NMB_S.NMB_S** for **DPLL_CTRL_0.RMO** =1 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1.DMO** =0, **DPLL_CTRL_1.COA** =0. 0x38 : Calculate **DPLL_NMB_S.NMB_S** for **DPLL_CTRL_0.RMO** =1 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1.DMO** =0, **DPLL_CTRL_1.COA** =1. 0x40 : Calculate **DPLL_NMB_T.NMB_T** for **DPLL_CTRL_0.RMO** =1 or **DPLL_CTRL_1.SMC** =1, **DPLL_CTRL_1.DMO** =1. 0x50 : Calculate **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** 0x51 : Write of **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** finished, all subincrement calculations done for last active input event 0x58 : Calculate **DPLL_TS_S** _check (**DPLL_IRQ_NOTIFY.MSI**), r_add_caln (prepare time stamp calculation(**DPLL_TS_S**)) for **DPLL_CTRL_0.IDT** equal **DPLL_CTRL_0.IFP** equal 1. 0x60 : Set **DPLL_STATUS.CAIP1** , **DPLL_STATUS.CAIP2** , Action masking bits , Action calculation loop 0x68 : Calculate **DPLL_NA[n]** , 0x70 : Calculate PDT_S[n] 0x71 : Calculate **DPLL_DTA[n].DTA** 0x78 : Calculate **DPLL_TSAC[n].TSAC** 0x79 : Calculate **DPLL_PSAC[n].PSAC** 0x7A : Action(n) in past condition occured: assignment of output data. 0x7B : Action loop control 0x80 : Wait for new Action calculation 0x90 : Calculate **DPLL_NMB_S_TAR.NMB_S_TAR** , Set **DPLL_STATUS.FPCE** -> **DPLL_IRQ_NOTIFY.EI** (if necessary) , Calculation of fast pulse correction while waiting for new input event or new PMT data. When finished change to "Action loop control" (0x7B).

STA_S	
	This bit field reflects the status of the *STATE* state machine **Note:** The decimal step number 21 to 40 of the state machine is binary coded from 0x01 to 0x14 respectively using the upper 5 bits [19:15] after subtraction of 20 to the decimal value. The lower 3 bits [14:12] show substates of the corresponding state machine. When the DPLL is disabled this field is 0x000.

CNT_S	
Description	Count STATE; this reflects the count of active STATE slopes (mod8).
Loop	–
Bit Range	$[23 : 21]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value shows the number of active *STATE* slopes (mod8) **Note:** This value allows distinguishing if the above state machine status is consistent to other status values read before or after it.

21.11.49 DPLL_INCF1_OFFSET

Description	Start value of ADD_IN_ADDER1
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_INCF1_OFFSET
Address	$0x8F44$
C-Name	GTM.CLS[0].DPLL.INCF1_OFFSET

Interface: MCS[i]

Name	DPLL_INCF1_OFFSET
Address	$0x8F44$
C-Name	

INCF1_OFFSET	
Description	Start value of the ADD_IN_ADDER1
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	In the case of **DPLL_CTRL_11.INCF1** =1 the ADD_IN_ADDER1 starts always after an active new input event (*TRIGGER* in normal mode or *STATE* in emergency mode respectively) with this offset value. In the case of choosing **DPLL_INCF1_OFFSET.INCF1_OFFSET** = 0xFFFFFF the generation of the first *SUB_INC1* pulse is performed with the next *TS_CLK* . In the case of **DPLL_INCF1_OFFSET.INCF1_OFFSET** = 0x000000 the first pulse is delayed by a full *SUB_INC1* period and in the case of **DPLL_INCF1_OFFSET.INCF1_OFFSET** = 0x7FFFFF the first pulse is delayed by a half *SUB_INC1* period. Any other value is possible.

21.11.50 DPLL_INCF2_OFFSET

Description	Start value of the ADD_IN_ADDER2
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_INCF2_OFFSET
Address	$0x8F48$
C-Name	GTM.CLS[0].DPLL.INCF2_OFFSET

Interface: MCS[i]

Name	DPLL_INCF2_OFFSET
Address	$0x8F48$
C-Name	

INCF2_OFFSET	
Description	Start value of the ADD_IN_ADDER2
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False

INCF2_OFFSET	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	In the case of set **DPLL_CTRL_11.INCF2** the ADD_IN_ADDER2 starts always after an active new input event (*STATE*) with this offset value. In the case of choosing **DPLL_INCF2_OFFSET.INCF2_OFFSET** = 0xFFFFFF the generation of the first *SUB_INC2* pulse is performed with the next *TS_CLK* . In the case of **DPLL_INCF2_OFFSET.INCF2_OFFSET** = 0x000000 the first pulse is delayed by a full *SUB_INC2* period and in the case of **DPLL_INCF2_OFFSET.INCF2_OFFSET** = 0x7FFFFF the first pulse is delayed by a half *SUB_INC2* period. Any other value is possible.

21.11.51 DPLL_DT_T_START

Description	Start value of DT_T_ACT
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_DT_T_START
Address	$0x8F4C$
C–Name	GTM.CLS[0].DPLL.DT_T_START

Interface: MCS[i]

Name	DPLL_DT_T_START
Address	$0x8F4C$
C–Name	

DT_T_START	
Description	Start value of DPLL_DT_T_ACT for the first increment after DPLL_CTRL_11.SIP1 is set to 1
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	257
Protect Enable Cond	–

DT_T_START	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	For the first increment after setting **DPLL_CTRL_11.SIP1** from 0 to 1 the value of **DPLL_DT_T_START.DT_T_START** is taken instead of the calculated **DPLL_DT_T_ACT** for the current increment duration. This value should be always > 256 in order to avoid an overflow during the calculation of **DPLL_RDT_T_ACT** .

21.11.52 DPLL_DT_S_START

Description	Start value of DT_S_ACT
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_DT_S_START
Address	$0x8F50$
C-Name	GTM.CLS[0].DPLL.DT_S_START

Interface: MCS[i]

Name	DPLL_DT_S_START
Address	$0x8F50$
C-Name	

DT_S_START	
Description	Start value of DPLL_DT_S_ACT for the first increment after DPLL_CTRL_11.SIP2 is set to 1
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	257
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	For the first increment after setting **DPLL_CTRL_11.SIP2** from 0 to 1 the value of **DPLL_DT_S_START.DT_S_START** is taken instead of the calculated **DPLL_DT_S_ACT.DT_S_ACT** for the current increment duration. This value should be always > 256 in order to avoid an overflow during the calculation of **DPLL_RDT_S_ACT.RDT_S_ACT** .

21.11.53 DPLL_STA_MASK

Description	Notify values for DPLL_STA
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_STA_MASK
Address	$0x8F54$
C-Name	GTM.CLS[0].DPLL.STA_MASK

Interface: MCS[i]

Name	DPLL_STA_MASK
Address	$0x8F54$
C-Name	

STA_NOTIFY_T	
Description	Notify value for DPLL_STA.STA_T.
Loop	-
Bit Range	$[7:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	The **DPLL_STA_MASK.STA_NOTIFY_T** is representing a trigger mask of **DPLL_STA.STA_T** . When **DPLL_STA.STA_T** reaches the value of **DPLL_STA_MASK.STA_NOTIFY_T** the flag **DPLL_STA_FLAG.STA_FLAG_T** is set to '1' when **DPLL_STA.STA_T** is leaving the state **DPLL_STA_MASK.STA_NOTIFY_T** . The signal is visible to MCS0 sub module as part of the special function register.

STA_NOTIFY_S	
Description	Notify value for DPLL_STA.STA_S.
Loop	-
Bit Range	$[15:8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

STA_NOTIFY_S	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	The **DPLL_STA_MASK.STA_NOTIFY_S** is representing a trigger mask of **DPLL_STA.STA_S** . When **DPLL_STA.STA_S** reaches the value of **DPLL_STA_MASK.STA_NOTIFY_S** the flag **DPLL_STA_FLAG.STA_FLAG_S** is set to '1' when **DPLL_STA.STA_S** is leaving the state **DPLL_STA_MASK.STA_NOTIFY_S** .

21.11.54 DPLL_STA_FLAG

Description	DPLL STA Flags
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_STA_FLAG
Address	$0x8F58$
C-Name	GTM.CLS[0].DPLL.STA_FLAG

Interface: MCS[i]

Name	DPLL_STA_FLAG
Address	$0x8F58$
C-Name	

STA_FLAG_T	
Description	Flag according to DPLL_MASK.STA_NOTIFY_T
Loop	–
Bit Range	$[0:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

STA_FLAG_T	
R-Coding	0 : The binary coded DPLL state (**DPLL_STA.STA_T**) for *TRIGGER* did not leave the state defined in **DPLL_STA_MASK.STA_NOTIFY_T** 1 : The binary coded DPLL state (**DPLL_STA.STA_T**) for *TRIGGER* left the state defined in **DPLL_STA_MASK.STA_NOTIFY_T**
W-Coding	0 : No action 1 : Clear flag
	The **DPLL_STA_FLAG.STA_FLAG_T** is set to '1' indicating that the signal **DPLL_STA.STA_T** has left the state defined by the trigger mask of **DPLL_STA_MASK.STA_NOTIFY_T** . The Flag is reset when this bit of the register is written to '1'. The signal is visible to MCS0 sub module as part of the special function register.

STA_FLAG_S	
Description	Flag according to DPLL_STA_MASK.STA_NOTIFY_S
Loop	-
Bit Range	[8 : 8]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The binary coded DPLL state (**DPLL_STA.STA_S**) for *STATE* did not leave the state defined in **DPLL_STA_MASK.STA_NOTIFY_S** 1 : The binary coded DPLL state (**DPLL_STA.STA_S**) for *STATE* left the state defined in **DPLL_STA_MASK.STA_NOTIFY_S**
W-Coding	0 : No action 1 : Clear flag
	The **DPLL_STA_FLAG.STA_FLAG_S** is set to '1' indicating that the signal **DPLL_STA.STA_S** has left the state defined by the trigger mask of **DPLL_STA_MASK.STA_NOTIFY_S** . The Flag is reset when this bit of the register is written to '1'. The signal is visible to MCS0 sub module as part of the special function register.

INC_CNT1_FLAG	
Description	Flag according to DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY
Loop	-
Bit Range	[9 : 9]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-

INC_CNT1_FLAG	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The counter value for *SUB_INC1* (**DPLL_INC_CNT1.INC_CNT1**) did not leave the state in which it is equal to the value stored in **DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY** 1 : The counter value for *SUB_INC1* (**DPLL_INC_CNT1.INC_CNT1**) left the state in which it is equal to the value stored in **DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY**
W-Coding	0 : No action 1 : Clear flag
	The **DPLL_STA_FLAG.INC_CNT1_FLAG** is set to '1' indicating that the signal **DPLL_INC_CNT1.INC_CNT1** has left the state defined by the trigger mask of **DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY** . The Flag is reset when this bit of the register is written to '1'. The signal is visible to MCS0 sub module as part of the special function register.

INC_CNT2_FLAG	
Description	Flag according to DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY
Loop	-
Bit Range	$[10 : 10]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
R-Coding	0 : The counter value for *SUB_INC2* (**DPLL_INC_CNT2.INC_CNT2**) did not leave the state in which it is equal to the value stored in **DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY** 1 : The counter value for *SUB_INC2* (**DPLL_INC_CNT2.INC_CNT2**) left the state in which it is equal to the value stored in **DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY**
W-Coding	0 : No action 1 : Clear flag
	The **DPLL_STA_FLAG.INC_CNT2_FLAG** is set to '1' indicating that the signal **DPLL_INC_CNT2.INC_CNT2** has left the state defined by the trigger mask of **DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY** . The Flag is reset when this bit of the register is written to '1'. The signal is visible to MCS0 sub module as part of the special function register.

21.11.55 DPLL_INC_CNT1_MASK

Description	Notify value of DPLL_INC_CNT1
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_INC_CNT1_MASK
Address	$0x8F5C$
C-Name	GTM.CLS[0].DPLL.INC_CNT1_MASK

Interface: MCS[i]

Name	DPLL_INC_CNT1_MASK
Address	$0x8F5C$
C-Name	

INC_CNT1_NOTIFY	
Description	Notify value for DPLL_INC_CNT1.INC_CNT1.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	The **DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY** is representing a trigger mask of **DPLL_INC_CNT1.INC_CNT1** . When **DPLL_INC_CNT1.INC_CNT1** reaches the value of **DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY** the flag **DPLL_STA_FLAG.INC_CNT1_FLAG** is set to '1' when **DPLL_INC_CNT1.INC_CNT1** is leaving the state **DPLL_INC_CNT1_MASK.INC_CNT1_NOTIFY** .

21.11.56 DPLL_INC_CNT2_MASK

Description	Notify value of DPLL_INC_CNT2
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_INC_CNT2_MASK
Address	$0x8F60$
C-Name	GTM.CLS[0].DPLL.INC_CNT2_MASK

Interface: MCS[i]

Name	DPLL_INC_CNT2_MASK
Address	$0x8F60$

C-Name	

INC_CNT2_NOTIFY	
Description	Notify value for DPLL_INC_CNT2.INC_CNT2 of register DPLL_INC_CNT2.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	The **DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY** is representing a trigger mask of **DPLL_INC_CNT2.INC_CNT2** . When **DPLL_INC_CNT2.INC_CNT2** reaches the value of **DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY** the flag **DPLL_STA_FLAG.INC_CNT2_FLAG** is set to '1' when **DPLL_INC_CNT2.INC_CNT2** is leaving the state **DPLL_INC_CNT2_MASK.INC_CNT2_NOTIFY** .

21.11.57 DPLL_NUSC_EXT1

Description	Number of Recent STATE Events used for Calculations
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NUSC_EXT1
Address	$0x8F64$
C-Name	GTM.CLS[0].DPLL.NUSC_EXT1

Interface: MCS[i]

Name	DPLL_NUSC_EXT1
Address	$0x8F64$
C-Name	

SYN_S	
Description	Number of real and virtual events to be considered for the current increment.
Loop	-
Bit Range	$[6:0]$
Access Type	RW
Volatile	True

SYN_S	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 30, 30) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value reflects the **DPLL_ADT_S[p].NS** value of the last valid increment, stored in **DPLL_ADT_S[p]** ; to be updated after all calculations in step 37 of table in chapter *51 "State description of the State Machine Table"* . **Note:** This value can only be written when the **DPLL_NUSC_EXT1.WSYN** bit in this register is set.

SYN_S_OLD	
Description	Number of real and virtual events to be considered for the last increment.
Loop	–
Bit Range	[22 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 30, 30) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value reflects the **DPLL_ADT_S[p].NS** value of the last but one valid increment, stored in **DPLL_ADT_S[p]** ; is updated automatically when writing **DPLL_NUSC_EXT1.SYN_S** . **Note:** This value can only be written when the **DPLL_NUSC_EXT1.WSYN** bit in this register is set.

WSYN	
Description	Write control bit for DPLL_NUSC_EXT1.SYN_S and DPLL_NUSC_EXT1.SYN_S_OLD; read as zero.
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–

WSYN	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : The **DPLL_NUSC_EXT1.SYN_S** value is not writeable 1 : The **DPLL_NUSC_EXT1.SYN_S** value is writeable

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is set. If **DPLL_CTRL_11.STATE_EXT** is not set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is not set any read access to this register will return *AEI_STATUS* = 0b00. Following the recommendation of section *21.6.12 "Software reset and DPLL deactivation"* a read access, when **DPLL_CTRL_11.STATE_EXT** is not set, delivers data for **DPLL_NUSC_EXT1.SYN_S** equal to (0 & **DPLL_NUSC.SYN_S**) and for **DPLL_NUSC_EXT1.SYN_S_OLD** equal to (0 & **DPLL_NUSC.SYN_S_OLD**).

21.11.58 DPLL_NUSC_EXT2

Description	Number of Recent STATE Events used for Calculations
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NUSC_EXT2
Address	$0x8F68$
C-Name	GTM.CLS[0].DPLL.NUSC_EXT2

Interface: MCS[i]

Name	DPLL_NUSC_EXT2
Address	$0x8F68$
C-Name	

NUSE	
Description	Number of recent STATE events used for SUB_INC1 and SUB_INC1 calculations modulo 2*(DPLL_CTRL_EXT.SNU+1).
Loop	-
Bit Range	$[6 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	1
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 29, 29) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

NUSE	
	No gap is considered in that case for this value, but in the **DPLL_NUSC_EXT2.VSN** value (see below): This register is set by the CPU but reset automatically to "1" by a change of direction or loss of LOCK. Each other value can be set by the CPU, maybe Full_SCALE, HALF_SCALE or parts of them. The relation values QDT_S[x] are calculated using **DPLL_NUSC_EXT2.NUSE** values in the past with its maximum value of 2* **DPLL_CTRL_EXT.SNU** +1. **Note:** This value can only be written when the **DPLL_NUSC_EXT2.WNUS** bit is set.

FSS	
Description	FULL_SCALE of STATE; this value is to be set, when DPLL_NUSC_EXT2.NUSE is set to FULL_SCALE
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 29, 29) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The **DPLL_NUSC_EXT2.NUSE** value is less than FULL_SCALE 1 : The **DPLL_NUSC_EXT2.NUSE** value is equal to FULL_SCALE
	This value is set by the CPU, but reset automatically to "0" by a change of direction or loss of LOCK. **Note:** This value can only be written when the **DPLL_NUSC_EXT2.WNUS** bit is set.

VSN	
Description	Virtual STATE number; number of virtual STATE increments in the current DPLL_NUSC_EXT2.NUSE region.
Loop	–
Bit Range	[22 : 16]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 31, 31) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync

VSN	
	This value reflects the number of virtual increments in the current **DPLL_NUSC_EXT2.NUSE** region; for **DPLL_NUSC_EXT2.NUSE** =1 this value is zero, when the CPU sets **DPLL_NUSC_EXT2.NUSE** to a value > 1 or zero(2^7 modulo 2^7), it must also set **DPLL_NUSC_EXT2.VSN** to the corresponding value; the **DPLL_NUSC_EXT2.VSN** value is subtracted from the **DPLL_NUSC_EXT2.NUSE** value in order to get the corresponding **DPLL_APS_EXT.APS** value for the past; the **DPLL_NUSC_EXT2.VSN** value is not used for the **DPLL_APS_EXT.APS_1C2** pointer. **DPLL_NUSC_EXT2.VSN** is to be updated by the CPU when a new gap is to be considered for **DPLL_NUSC_EXT2.NUSE** or a gap is leaving the **DPLL_NUSC_EXT2.NUSE** region; for this purpose the **DPLL_IRQ_NOTIFY.SASI** interrupt can be used; no further update of **DPLL_NUSC_EXT2.VSN** is necessary when **DPLL_NUSC_EXT2.NUSE** is set to FULL_SCALE **Note:** This value can only be written when the **DPLL_NUSC_EXT2.WVSN** bit is set.

WNUS	
Description	Write control bit for DPLL_NUSC_EXT2.NUSE; read as zero.
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : The **DPLL_NUSC_EXT2.NUSE** value is not writeable 1 : The **DPLL_NUSC_EXT2.NUSE** value is writeable

WVSN	
Description	Write control bit for DPLL_NUSC_EXT2.VSN; read as zero.
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : The **DPLL_NUSC_EXT2.VSN** value is not writeable 1 : The **DPLL_NUSC_EXT2.VSN** value is writeable

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is set. If **DPLL_CTRL_11.STATE_EXT** is not set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is not set any read access to this register will return *AEI_STATUS* = 0b00. Following the recommendation of section *21.6.12 "Software reset and DPLL deactivation"* a read access, when **DPLL_CTRL_11.STATE_EXT** is not set, delivers data for **DPLL_NUSC_EXT2.NUSE** equal to (0 & **DPLL_NUSC.NUSE**) for **DPLL_NUSC_EXT2.FSS** equal to **DPLL_NUSC.FSS** and for **DPLL_NUSC_EXT2.VSN** equal to (0 & **DPLL_NUSC.VSN**).

21.11.59 DPLL_APS_EXT

Description	Actual RAM Pointer Address for STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APS_EXT
Address	$0x8F38$
C-Name	GTM.CLS[0].DPLL.APS_EXT

Interface: MCS[i]

Name	DPLL_APS_EXT
Address	$0x8F38$
C-Name	

WAPS	
Description	Write bit for address pointer DPLL_APS_EXT.APS, read as zero.
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : The **DPLL_APS_EXT.APS** is not writeable 1 : The **DPLL_APS_EXT.APS** is writeable

APS	
Description	Address pointer STATE; Actual RAM pointer address value for DPLL_DT_S[p] and DPLL_RDT_S[p]
Loop	-
Bit Range	$[8:2]$
Access Type	RW
Volatile	True
Multithread	False

APS	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 1, 1) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Actual RAM pointer and synchronization position/value of *STATE* events in FULL_SCALE for up to 128 *STATE* events but limited to 2*(**DPLL_CTRL_EXT.SNU** +1− **DPLL_CTRL_EXT.SYN_NS**) in normal and emergency mode for **DPLL_CTRL_1.SYSF** =0 or to 2*(**DPLL_CTRL_EXT.SNU** +1)− **DPLL_CTRL_EXT.SYN_NS** for **DPLL_CTRL_1.SYSF** =1 respectively; See chapter *22.2 "MCS to DPLL Interface Description"* . **DPLL_APS_EXT.APS** is incremented (decremented) by one for each active *STATE* event and *DIR2* =0 *DIR2* =1). The **DPLL_APS_EXT.APS** offset value is added in the above shown bit position with the subsection offset of the RAM region. **Note:** The **DPLL_APS_EXT.APS** pointer value is directed to the RAM position, in which the data values are to be written, which correspond to the last increment. The **DPLL_APS_EXT.APS** value is not to be changed, when the direction (shown by *DIR2*) changes, because it points always to a storage place after the considered increment. Changing of *DIR2* takes place always after an active *STATE* event and the resulting increment/decrement. **Note:** This value can only be written when the **DPLL_APS_EXT.WAPS** bit is set.

WAPS_1C2	
Description	Write bit for address pointer DPLL_APS_EXT.APS_1C2, read as zero.
Loop	–
Bit Range	[13 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W−Coding	0 : The **DPLL_APS_EXT.APS_1C2** is not writeable 1 : The **DPLL_APS_EXT.APS_1C2** is writeable

APS_1C2	
Description	Address pointer STATE for RAM region 1c2; Actual RAM pointer address value for DPLL_TSF_S[p].
Loop	–
Bit Range	[20 : 14]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

APS_1C2	
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 13, 13) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Initial value: zero (0x00). Actual RAM pointer and synchronization position/value of *STATE* events in FULL_SCALE for up to 128 *STATE* events but limited to 2*(**DPLL_CTRL_EXT.SNU** +1) in normal and emergency mode; this pointer is used for the RAM region 1c2. For **DPLL_STATUS.SYS** =1: **DPLL_APS_EXT.APS_1C2** is incremented (decremented) by **DPLL_NUSC_EXT1.SYN_S_OLD** for each active *STATE* event and *DIR2* =0 (*DIR2* =1). For **DPLL_STATUS.SYS** =0: **DPLL_APS_EXT.APS_1C2** is incremented or decremented by 1 respectively. The **DPLL_APS_EXT.APS_1C2** offset value is added in the bit position shown above with the subsection offset of the RAM region. In addition, when the **DPLL_APS_1C3_EXT.APS_1C3** value is written by the CPU – in order to synchronize the DPLL– with the next active *STATE* event the **DPLL_APS_SYNC_EXT.APS_1C2_EXT** value is added/subtracted (while **DPLL_APS_SYNC_EXT.APS_1C2_STATUS** is one; see **DPLL_APT_SYNC** register at chapter **DPLL_APS_SYNC**). **Note:** This value can only be written when the **DPLL_APS_EXT.WAPS_1C2** bit is set

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is set. If **DPLL_CTRL_11.STATE_EXT** is not set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is not set any read access to this register will return *AEI_STATUS* = 0b00. Following the recommendation of section *21.6.12 "Software reset and DPLL deactivation"* a read access, when **DPLL_CTRL_11.STATE_EXT** is not set, delivers data for **DPLL_APS_EXT.APS** equal to (0 & **DPLL_APS.APS**) and for **DPLL_APS_EXT.APS_1C2** equal to (0 & **DPLL_APS.APS_1C2**).

21.11.60 DPLL_APS_1C3_EXT

Description	Actual RAM Pointer Address for RAM region 1c3 (DPLL_CTRL_11.STATE_EXT=1)
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APS_1C3_EXT
Address	$0x8F3C$
C–Name	GTM.CLS[0].DPLL.APS_1C3_EXT

Interface: MCS[i]

Name	DPLL_APS_1C3_EXT
Address	$0x8F3C$
C–Name	

APS_1C3	
Description	Address pointer STATE for RAM region 1c3; Actual RAM pointer address value for DPLL_ADT_S[p]
Loop	–
Bit Range	$[8:2]$

APS_1C3	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_11.STATE_EXT == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Initial value: zero (0x00). Actual RAM pointer and synchronization position/value of *STATE* events in FULL_SCALE for up to 128 *STATE* events but limited to 2*(**DPLL_CTRL_EXT.SNU** +1− **DPLL_CTRL_EXT.SYN_NS**) in normal and emergency mode for **DPLL_CTRL_1.SYSF** =0 or to 2*(**DPLL_CTRL_EXT.SNU** +1)− **DPLL_CTRL_EXT.SYN_NS** for **DPLL_CTRL_1.SYSF** =1 respectively; this pointer is used for the RAM region 1c3. See chapter *22.2 "MCS to DPLL Interface Description"* . The RAM pointer is set by the CPU accordingly, when the synchronization condition was detected.

Note:

The **DPLL_APS_1C3_EXT.APS_1C3** pointer value is directed to the RAM position of the profile element in RAM region 1c3, which corresponds to the current increment. When changing the direction *DIR1* or *DIR2* respectively, this is always known before an active *STATE* event is processed. This is because of the pattern recognition in SPE (for PMSM) or because of the direction change recognition by *TRIGGER* . This direction change results in an automatic increment (forward) or decrement (backward) when the input event occurs in addition to 2 times correction.
The **DPLL_APS_1C3_EXT.APS_1C3** offset value is added in the bit position shown above with the subsection address offset of the corresponding RAM region.

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is set. If **DPLL_CTRL_11.STATE_EXT** is not set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is not set any read access to this register will return *AEI_STATUS* = 0b00. Following the recommendation of section *21.6.12 "Software reset and DPLL deactivation"* a read access, when **DPLL_CTRL_11.STATE_EXT** is not set, delivers data for **DPLL_APS_1C3_EXT.APS_1C3** equal to (0 & **DPLL_APS_1C3.APS_1C3**).

21.11.61 DPLL_APS_SYNC_EXT

Description	STATE Time Stamp Field Offset at Synchronization Time
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_APS_SYNC_EXT
Address	$0x8F30$
C–Name	GTM.CLS[0].DPLL.APS_SYNC_EXT

Interface: MCS[i]

Name	DPLL_APS_SYNC_EXT
Address	$0x8F30$
C–Name	

APS_1C2_EXT	
Description	Address pointer 1c2 extension; this offset value determines, by which value the DPLL_APS_EXT.APS_1C2 is changed at the time of synchronization; set by CPU before the synchronization is performed.
Loop	–
Bit Range	[6 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_11.STATE_EXT == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This offset value is the number of virtual increments to be inserted in the **DPLL_TSF_T[p].TSF_T** for an imminent intended synchronization; the CPU sets its value depending on the gaps until the time of synchronization considering the **DPLL_NUSC_EXT2.NUSE** value to be set and including the future increment (when **DPLL_NUSC_EXT1.SYN_S_OLD** is still 1). When the synchronization takes place, this value is to be added to the **DPLL_APS_EXT.APS_1C2** address pointer (for forward direction, *DIR2* =0) and the **DPLL_APS_EXT.APS_1C2** _status bit is cleared after it. For backward direction subtract **DPLL_APS_SYNC_EXT.APS_1C2_EXT** accordingly. **Note:** When the synchronization is intended and the **DPLL_NUSC_EXT2.NUSE** value is to be set to FULL_SCALE after it, the **DPLL_APS_SYNC_EXT.APS_1C2_EXT** value must be set to **DPLL_CTRL_EXT.SYN_NS** (for **DPLL_CTRL_1.SYSF** =1) or 2* **DPLL_CTRL_EXT.SYN_NS** (for **DPLL_CTRL_1.SYSF** =0) in order to fill all gaps in the extended **DPLL_TSF_S[p].TSF_S** with the corresponding values by the CPU. When all values for FULL_SCALE are still not available, the **DPLL_APS_SYNC_EXT.APS_1C2_EXT** value considers only a share according to the **DPLL_NUSC_EXT2.NUSE** value to be set after the synchronization.

APS_1C2_STATUS	
Description	Address pointer 1c2 status; set by CPU before the synchronization is performed. The value is cleared automatically when the DPLL_APS_SYNC_EXT.APS_1C2_OLD value is written.
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_11.STATE_EXT == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : **DPLL_APS_SYNC_EXT.APS_1C2_EXT** is not to be considered. 1 : **DPLL_APS_SYNC_EXT.APS_1C2_EXT** has to be considered for time stamp field extension.

APS_1C2_OLD	
Description	Address pointer STATE for RAM region 1c2 at synchronization time; this value is set by the current DPLL_APS_EXT.APS_1C2 value when the synchronization takes place for the first active STATE event after writing DPLL_APS_1C3_EXT.APS_1C3 but before adding the offset value DPLL_APS_SYNC_EXT.APS_1C2_EXT (that means: when DPLL_APS_SYNC_EXT.APS_1C2_STATUS=1).

APS_1C2_OLD	
Loop	–
Bit Range	$[22 : 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	DPLL_CTRL_11.STATE_EXT == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Address pointer **DPLL_APS_EXT.APS_1C2** value at the time of synchronization, before the offset value is added, that means the pointer with this value points to the last value before the additional inserted gap

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is set. If **DPLL_CTRL_11.STATE_EXT** is not set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is not set any read access to this register will return *AEI_STATUS* = 0b00. Following the recommendation of section *21.6.12 "Software reset and DPLL deactivation"* a read access, when **DPLL_CTRL_11.STATE_EXT** is not set, delivers data for **DPLL_APS_SYNC_EXT.APS_1C2_EXT** equal to (0 & **DPLL_APS_SYNC.APS_1C2_EXT**), **DPLL_APS_SYNC_EXT.APS_1C2_STATUS** equal to (**DPLL_APS_SYNC.APS_1C2_STATUS**) and **DPLL_APS_SYNC_EXT.APS_1C2_OLD** equal to (0 & **DPLL_APS_SYNC.APS_1C2_OLD**).

21.11.62 DPLL_CTRL_EXT

Description	STATE Time Stamp Field Offset at Synchronization Time
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_EXT
Address	$0x8F34$
C–Name	GTM.CLS[0].DPLL.CTRL_EXT

Interface: MCS[i]

Name	DPLL_CTRL_EXT
Address	$0x8F34$
C–Name	

SNU	
Description	STATE number; SNU+1 is number of nominal STATE events in HALF_SCALE (1...64).
Loop	–
Bit Range	$[5 : 0]$
Access Type	RW

SNU	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$0x17$
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Note: This bit can only be written when the DPLL is disabled. **Note:** The number of nominal *STATE* events is the decimal value plus 1. This value can only be written when (**DPLL_CTRL_0.RMO** =0 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.DEN** =0. Set **DPLL_CTRL_1.SSL** =00 before changing this value and set **DPLL_CTRL_0.RMO** =1 only after FULL_SCALE with **DPLL_CTRL_1.SSL** >0.

SYN_NS	
Description	Synchronization number of STATE; summarized number of virtual increments in HALF_SCALE
Loop	–
Bit Range	$[21 : 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1) || !(DPLL_CTRL_0.RMO == 0 && DPLL_CTRL_1.SMC == 0) || (DPLL_CTRL_11.STATE_EXT == 0)
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	Sum of all systematic missing *STATE* events in HALF_SCALE (for **DPLL_CTRL_1.SYSF** =0) or FULL_SCALE (for **DPLL_CTRL_1.SYSF** =1) ; the **DPLL_CTRL_EXT.SYN_NS** missing *STATE* events can be divided up to an arbitrary number of blocks. The pattern of events and missing events in FULL_SCALE is shown in RAM region 1c3 as value **DPLL_ADT_S[p].NS** in addition to the adapted values. The number of stored increments in FULL_SCALE must be equal to 2*(**DPLL_CTRL_EXT.SNU** +1− **DPLL_CTRL_EXT.SYN_NS**) for **DPLL_CTRL_1.SYSF** =0 or 2*(**DPLL_CTRL_EXT.SNU** +1)− **DPLL_CTRL_EXT.SYN_NS** for **DPLL_CTRL_1.SYSF** =1 . This pattern is written by the CPU beginning from a fixed reference point (maybe beginning of the FULL_SCALE region). The relation to the actual increment is established by setting of the profile RAM pointer **DPLL_APS_1C3_EXT.APS_1C3** in an appropriate relation to the RAM pointer **DPLL_APS_EXT.APS** of the actual increment by the CPU. **Note:** This value can only be written when the DPLL is disabled. **Note:** This value can only be written when (**DPLL_CTRL_0.RMO** =0 and **DPLL_CTRL_1.SMC** =0) or **DPLL_CTRL_1.DEN** =0. Set **DPLL_CTRL_1.SSL** =00 before changing this value and set **DPLL_CTRL_0.RMO** =1 only after FULL_SCALE with **DPLL_CTRL_1.SSL** >0.

Note:

This register is only used when **DPLL_CTRL_11.STATE_EXT** is set. If **DPLL_CTRL_11.STATE_EXT** is not set any write access to this register will return *AEI_STATUS* = 0b10. If **DPLL_CTRL_11.STATE_EXT** is not set any read access to this register will return *AEI_STATUS* = 0b00. Following the recommendation of section *21.6.12 "Software reset and DPLL deactivation"* a read access, when **DPLL_CTRL_11.STATE_EXT** is

not set, delivers data for **DPLL_CTRL_EXT.SNU** equal to (0 & **DPLL_CTRL_0.SNU**) and for **DPLL_CTRL_EXT.SYN_NS** equal to (0 & **DPLL_CTRL_1.SYN_NS**).

21.11.63 DPLL_SW_TRIG

Description	Software triggered input events
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_SW_TRIG
Address	$0x8F7C$
C-Name	GTM.CLS[0].DPLL.SW_TRIG

Interface: MCS[i]

Name	DPLL_SW_TRIG
Address	$0x8F7C$
C-Name	

TRIG_EVENT	
Description	Software triggered input event for TRIGGER
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 1, 1) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : No active *TRIGGER* event requested 1 : Request active *TRIGGER* event
	Note: As time stamp for the *TRIGGER* event the current 24 bit of the 27 bit *TIM_DPLL_TBU_TS0* input signal to the DPLL are used. See table *21.16 "DPLL Port Description"* . **Note:** The *TRIGGER* event is placed independently from the status of the signal **DPLL_CTRL_0.TEN** **Note:** When **DPLL_SW_TRIG.TRIG_EVENT** is set to '1' a *TRIGGER* event is executed with the next internal clock cycle. The signal **DPLL_SW_TRIG.TRIG_EVENT** is automatically set back to 0 such that **DPLL_SW_TRIG.TRIG_EVENT** is always read with '0'. ▽

TRIG_EVENT	
	△ **Note:** This value can only be written when the signal **DPLL_SW_TRIG.WTRIG_EVENT** is set to 1.

WTRIG_EVENT	
Description	Write enable for DPLL_SW_TRIG.TRIG_EVENT
Loop	–
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_SW_TRIG.TRIG_EVENT** is not enabled. 1 : Writing to **DPLL_SW_TRIG.TRIG_EVENT** is enabled.

TRIG_LEVEL	
Description	Input signal level of software triggered input event for TRIGGER
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 3, 3) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : Signal level of 0 for *TRIGGER* input signal requested by signal **DPLL_SW_TRIG.TRIG_EVENT** . 1 : Signal level of 1 for *TRIGGER* input signal requested by signal **DPLL_SW_TRIG.TRIG_EVENT** .
	Note: **DPLL_SW_TRIG.TRIG_LEVEL** replaces the input signal level of the *TRIGGER* input signal for a requested *TRIGGER* event.

WTRIG_LEVEL	
Description	Write enable for DPLL_SW_TRIG.TRIG_LEVEL
Loop	–
Bit Range	$[3:3]$

WTRIG_LEVEL	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_SW_TRIG.TRIG_LEVEL** is not enabled. 1 : Writing to **DPLL_SW_TRIG.TRIG_LEVEL** is enabled.
	Note: The signals **DPLL_SW_TRIG.TRIG_EVENT** and **DPLL_SW_TRIG.TRIG_LEVEL** provide the option to generate active input signals on the *TRIGGER* input channel with either active or inactive signal slope. These events are able to replace the *TRIGGER* input signal of the DPLL. **Attention:** The application has to take care that the sequence of input events with their input signals on the *TRIGGER* input channel does not cause any kind of inconsistency or timing conflicts within the DPLL module. The input signal of the DPLL *TRIGGER* input signal has higher priority compared to *TRIGGER* event request by **DPLL_SW_TRIG.TRIG_EVENT** . When both events are activated in the same system clock cycle the event on the *TRIGGER* input is executed if **DPLL_CTRL_0.TEN** =1.

STATE_EVENT	
Description	Software triggered input event for STATE
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 5, 5) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : No active *STATE* event requested 1 : Request active *STATE* event
	Note: As time stamp for the state event the current 24bit of the 27 bit *CCM[0]_TBU_TS0* of the TIM[0] input signal to the DPLL are used. See table *21.16 "DPLL Port Description"* . **Note:** The *STATE* event is placed independently from the status of the signal **DPLL_CTRL_0.SEN** **Note:** ▽

STATE_EVENT	
	△ When **DPLL_SW_TRIG.STATE_EVENT** is set to '1' a *STATE* event is executed with the next internal clock cycle. The signal **DPLL_SW_TRIG.STATE_EVENT** is automatically set back to 0 such that **DPLL_SW_TRIG.STATE_EVENT** is always read with '0'. **Note:** This value can only be written when the signal **DPLL_SW_TRIG.WSTATE_EVENT** is set to 1.

WSTATE_EVENT	
Description	Write enable for DPLL_SW_TRIG.STATE_EVENT
Loop	–
Bit Range	$[5:5]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : Writing to **DPLL_SW_TRIG.STATE_EVENT** is not enabled. 1 : Writing to **DPLL_SW_TRIG.STATE_EVENT** is enabled.

STATE_LEVEL	
Description	Input signal level of software triggered input event for STATE
Loop	–
Bit Range	$[6:6]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 7, 7) == 0
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW–Coding	0 : Signal level of 0 for *STATE* input signal requested by signal **DPLL_SW_TRIG.STATE_EVENT** . 1 : Signal level of 1 for *STATE* input signal requested by signal **DPLL_SW_TRIG.STATE_EVENT** .
	Note: **DPLL_SW_TRIG.STATE_LEVEL** replaces the input signal level of the *STATE* input signal for a requested *STATE* event.

WSTATE_LEVEL	
Description	Write enable for DPLL_SW_TRIG.STATE_LEVEL
Loop	–

WSTATE_LEVEL	
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W–Coding	0 : Writing to **DPLL_SW_TRIG.STATE_LEVEL** is not enabled. 1 : Writing to **DPLL_SW_TRIG.STATE_LEVEL** is enabled.
	Note: The signals **DPLL_SW_TRIG.STATE_EVENT** and **DPLL_SW_TRIG.STATE_LEVEL** provide the option to generate active input signals on the *STATE* input channel with either active or inactive signal slope. These events are able to replace the *STATE* input signal of the DPLL. **Attention:** The application has to take care that the sequence of input events with their input signals on the *STATE* input channel does not cause any kind of inconsistency or timing conflicts within the DPLL module. The input signal of the DPLL *STATE* input signal has higher priority compared to *STATE* event request by **DPLL_SW_TRIG.STATE_EVENT** . When both events are activated in the same system clock cycle the event on the *STATE* input is executed if **DPLL_CTRL_0.SEN** =1.

21.11.64 DPLL_MP_T

Description	Missing pulses of TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_MP_T
Address	$0x8F80$
C–Name	GTM.CLS[0].DPLL.MP_T

Interface: MCS[i]

Name	DPLL_MP_T
Address	$0x8F80$
C–Name	

MP_T	
Description	Number of missing pulses of the SUB_INC1 pulses in automatic end mode (DPLL_CTRL_1.DMO=0).
Loop	–

MP_T	
Bit Range	$[23:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value is internally captured from the **DPLL_INC_CNT1.INC_CNT1** bitfield in normal mode (**DPLL_CTRL_0.RMO** =0) or in double synchronous mode (**DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =1), when the active input signal edge arrives on the *TRIGGER* input. The bitfield **DPLL_MP_T.MP_T** is written together with the **DPLL_PSTM** , **DPLL_PSTM_OLD** values (See bitfield **DPLL_STA.STA_T** in chapter **DPLL_STA**). The signal **DPLL_MP_T.MP_T** can be used together with **DPLL_PSTM** / **DPLL_PSTM_OLD** values of RAM1b to check the alignment of the angle clock of *CCM[0]_TBU_TS1* at the active signal edges of the *TRIGGER* input signal (normal mode and double synchronous mode).

21.11.65 DPLL_MP_S

Description	Missing pulses of STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_MP_S
Address	$0x8F84$
C-Name	GTM.CLS[0].DPLL.MP_S

Interface: MCS[i]

Name	DPLL_MP_S
Address	$0x8F84$
C-Name	

MP_S	
Description	Number of missing pulses of the SUB_INC1/SUB_INC2 pulses in automatic end mode (DPLL_CTRL_1.DMO=0).
Loop	–
Bit Range	$[23:0]$
Access Type	R
Volatile	True
Multithread	False

MP_S	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
	This value is internally captured from the **DPLL_INC_CNT1.INC_CNT1** bitfield in emergency mode (**DPLL_CTRL_0.RMO** =1, **DPLL_CTRL_1.SMC** =0) or from the **DPLL_INC_CNT2.INC_CNT2** bitfield in double synchronous mode (**DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =1), when the active input signal edge on the *STATE* input arrives. The bitfield **DPLL_MP_S.MP_S** is written together with the **DPLL_PSSM** , **DPLL_PSSM_OLD** values of RAM1b (See bitfield **DPLL_STA.STA_S** in chapter **DPLL_STA**). The Signal **DPLL_MP_S.MP_S** can be used together with **DPLL_PSSM.PSSM** / **DPLL_PSSM_OLD.PSSM_OLD** values of RAM1b to check the alignment of the angle clock of *CCM[0]_TBU_TS1* (emergency mode) or *CCM[0]_TBU_TS2* (double synchronous mode) at the active signal edges of the *TRIGGER* or *STATE* input signal.

21.11.66 DPLL_CTRL_12

Description	DPLL control register 12
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CTRL_12
Address	$0x8F88$
C–Name	GTM.CLS[0].DPLL.CTRL_12

Interface: MCS[i]

Name	DPLL_CTRL_12
Address	$0x8F88$
C–Name	

SUBINC_MUX_SEL	
Description	Selection of DPLL sub increment source for CCM[0]_TBU_TS1 angle base.
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	bitrange(CLS[0]_AEI_ARB_WDATA, 16, 16) == 0

SUBINC_MUX_SEL	
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
RW-Coding	0 : The output signals of pulse generator 1 (*21.9.3 "Sub pulse generation for DPLL_CTRL_1.SMC=0"*) are used to feed the output signals *SUB_INC1* and *SUB_INC1C* . 1 : The output signals of pulse generator 2 (*21.9.4 "Sub pulse generation for DPLL_CTRL_1.SMC=1"*) are used to feed the output signals *SUB_INC1* and *SUB_INC1C* .
	This control bit allows to multiplex the DPLL output signals to TBU channel 1 (*SUB_INC1* , *SUB_INC1C*) to be driven either by pulse generator 1 (default) output signals or the pulse generator 2 output signals.

WSUBINC_MUX_SEL	
Description	Write enable for DPLL_CTRL_12.SUBINC_MUX_SEL.
Loop	-
Bit Range	[16 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async DPLL_DEEP_RESET: sync
W-Coding	0 : Writing to **DPLL_CTRL_12.SUBINC_MUX_SEL** is not enabled. 1 : Writing to **DPLL_CTRL_12.SUBINC_MUX_SEL** is enabled.
	DPLL_CTRL_12.WSUBINC_MUX_SEL is automatically set back to 0 such that **DPLL_CTRL_12.WSUBINC_MUX_SEL** is always read with '0'.

21.12 DPLL RAM Region 1a value description

Note:

Bits 31 to 24 of RAM region 1a are not implemented and therefore always read as zero (reserved). Other bits which are declared as reserved are not protected against writing. Unused address regions are not protected against writing when implemented.

Attention:

Access restrictions for the RAM region 1a are described in *2 "AEI Status Signal"* .

21.12.1 DPLL_PSA[n]

Description	Position Request for Action [n]
Loop	$n = \{n : 0 <= n <= \mathrm{NOAC} - 1\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PSA[n]
Address	$0x4 * n + 0x8200$
C-Name	GTM.CLS[0].DPLL.PSA[n]

Interface: MCS[i]

Name	DPLL_PSA[n]
Address	$0x4 * n + 0x8200$
C-Name	

PSA	
Description	Position information of a desired Action [n]
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	DPLL_RAM_INIT: sync
	Note: This value can only be written/read when the DPLL is disabled.

21.12.2 DPLL_DLA[n]

Description	Time to React for Action [n]
Loop	$n = \{n : 0 <= n <= \mathrm{NOAC} - 1\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_DLA[n]
Address	$0x4 * n + 0x8280$
C-Name	GTM.CLS[0].DPLL.DLA[n]

Interface: MCS[i]

Name	DPLL_DLA[n]
Address	$0x4 * n + 0x8280$
C-Name	

DLA	
Description	Time to react before the corresponding position value of a desired Action [n] is reached .
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	DPLL_RAM_INIT: sync
	In the case of *LOW_RES* =1 (see table *21.16 "DPLL Port Description"*) this delay value must be also given as low resolution value. **Note:** This value can only be written/read when the DPLL is disabled.

21.12.3 DPLL_NA[n]

Description	Calculated Relative Time to Action [n]
Loop	$n = \{n : 0 <= n <= \mathrm{NOAC} - 1\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NA[n]
Address	$0x4 * n + 0x8300$
C–Name	GTM.CLS[0].DPLL.NA[n]

Interface: MCS[i]

Name	DPLL_NA[n]
Address	$0x4 * n + 0x8300$
C–Name	

DB	
Description	Number of events to Action [n] (fractional part).
Loop	–
Bit Range	$[9 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–

DB	
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	DPLL_RAM_INIT: sync
	Note: This value can only be written/read when the DPLL is disabled.

DW	
Description	Number of events to Action [n] (integer part).
Loop	–
Bit Range	[19 : 10]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	DPLL_RAM_INIT: sync
	Note: Use the maximum value for **DPLL_NA[n].DW** =0x3FF in the case of a calculated value which exceeds the representable value. **Note:** This value can only be written/read when the DPLL is disabled.

NOT_USED	
Description	Not used
Loop	–
Bit Range	[23 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.12.4 DPLL_DTA[n]

Description	Calculated Relative Time to Action [n]
Loop	$n = \{n : 0 <= n <= \text{NOAC} - 1\}$
Condition	$\text{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\text{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_DTA[n]
Address	$0x4 * n + 0x8380$
C-Name	GTM.CLS[0].DPLL.DTA[n]

Interface: MCS[i]

Name	DPLL_DTA[n]
Address	$0x4 * n + 0x8380$
C-Name	

DTA	
Description	Calculated relative time to Action [n]
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	(DPLL_CTRL_1.DEN == 1)
Reset group	DPLL_RAM_INIT: sync
	Note: This value can only be written/read when the DPLL is disabled. The **DPLL_DTA[n].DTA** value is a positive integer value. When calculations using equations *[eq_87]* , *[eq_113]* or *[eq_126]* result in a negative value, it is replaced by zero.

21.13 DPLL RAM Region 1b and 1c value description

Bits [31:24] of RAM region 1b and 1c are not implemented and therefore always read as zero (reserved). Other bits which are declared as reserved are not protected against writing. Unused address regions are not protected against writing when implemented.

21.13.1 DPLL_TS_T

Description	Actual TRIGGER Time Stamp Value
Loop	
Condition	$\text{NDPLL} > 0$

Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TS_T
Address	$0x8400$
C-Name	GTM.CLS[0].DPLL.TS_T

Interface: MCS[i]

Name	DPLL_TS_T
Address	$0x8400$
C-Name	

TRIGGER_TS	
Description	Time stamp value of the last active TRIGGER input.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	measured TRIGGER time stamp **Note:** The LSB address is determined using the **DPLL_OSW.SWON_T** value (see chapter **DPLL_ACT_STA**).

21.13.2 DPLL_TS_T_OLD

Description	Previous TRIGGER Time Stamp Value
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TS_T_OLD
Address	$0x8404$
C-Name	GTM.CLS[0].DPLL.TS_T_OLD

Interface: MCS[i]

Name	DPLL_TS_T_OLD
Address	$0x8404$
C-Name	

TRIGGER_TS_OLD	
Description	Time stamp value of the last but one active TRIGGER input.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	previous measured *TRIGGER* time stamp **Note:** The LSB address is determined using the **DPLL_OSW.SWON_T** .

21.13.3 DPLL_FTV_T

Description	Actual TRIGGER Filter value
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_FTV_T
Address	$0x8408$
C-Name	GTM.CLS[0].DPLL.FTV_T

Interface: MCS[i]

Name	DPLL_FTV_T
Address	$0x8408$
C-Name	

TRIGGER_FT	
Description	Filter value of the last active TRIGGER input.
Loop	-
Bit Range	$[23:0]$
Access Type	RW

TRIGGER_FT	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Transmitted filter value

21.13.4 DPLL_RAM1B_RSVD_0

Description	DPLL RAM1B reserved data
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RAM1B_RSVD_0
Address	$0x840C$
C-Name	GTM.CLS[0].DPLL.RAM1B_RSVD_0

Interface: MCS[i]

Name	DPLL_RAM1B_RSVD_0
Address	$0x840C$
C-Name	

RSVD	
Description	Reserved Data word, no DPLL internal use.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync

21.13.5 DPLL_TS_S

Description	Actual STATE Time Stamp Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TS_S
Address	$0x8410$
C-Name	GTM.CLS[0].DPLL.TS_S

Interface: MCS[i]

Name	DPLL_TS_S
Address	$0x8410$
C-Name	

STATE_TS	
Description	Time stamp value of the last active STATE input.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Note: The LSB address is determined using the **DPLL_OSW.SWON_S** .

21.13.6 DPLL_TS_S_OLD

Description	Previous STATE Time Stamp Register
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TS_S_OLD

Address	$0x8414$
C-Name	GTM.CLS[0].DPLL.TS_S_OLD

Interface: MCS[i]

Name	DPLL_TS_S_OLD
Address	$0x8414$
C-Name	

STATE_TS_OLD	
Description	Time stamp value of the last active STATE input.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Note: The LSB address is determined using the **DPLL_OSW.SWON_S** .

21.13.7 DPLL_FTV_S

Description	Actual STATE Filter Value
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_FTV_S
Address	$0x8418$
C-Name	GTM.CLS[0].DPLL.FTV_S

Interface: MCS[i]

Name	DPLL_FTV_S
Address	$0x8418$
C-Name	

STATE_FT	
Description	Filter value of the last active STATE input.

STATE_FT	
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	transmitted filter value

21.13.8 DPLL_RAM1B_RSVD_1

Description	DPLL RAM1B reserved data
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RAM1B_RSVD_1
Address	$0x841C$
C-Name	GTM.CLS[0].DPLL.RAM1B_RSVD_1

Interface: MCS[i]

Name	DPLL_RAM1B_RSVD_1
Address	$0x841C$
C-Name	

RSVD	
Description	Reserved Data word, no DPLL internal use.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

RSVD	
Reset group	DPLL_RAM_INIT: sync

21.13.9 DPLL_THMI

Description	TRIGGER Hold Time Min. Value
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_THMI
Address	$0x8420$
C-Name	GTM.CLS[0].DPLL.THMI

Interface: MCS[i]

Name	DPLL_THMI
Address	$0x8420$
C-Name	

THMI	
Description	Minimal time between active and inactive TRIGGER slope; the time value corresponds to the time stamp clock counts: This means the clock selected for the TBU_CH0_BASE (see TBU_CH0_CTRL register)
Loop	-
Bit Range	$[15:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Set min. value; generate the *DPLL_TINI* interrupt in the case of a violation for 0xFFFF> **DPLL_THMI.THMI** >0. **Note:** Typical retention time values after an active slope can be, for example between 45 µs (forward) and 90 µs (backward). When **DPLL_THMI.THMI** is zero, consider always a **DPLL_THMI.THMI** violation (forwards). When **DPLL_THMI.THMI** is 0xFFFF, consider always no **DPLL_THMI.THMI** violation (backwards direction).

NOT_USED	
Description	Not used
Loop	-
Bit Range	$[23:16]$

NOT_USED	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.10 DPLL_THMA

Description	TRIGGER Hold Time Max. Value
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_THMA
Address	$0x8424$
C–Name	GTM.CLS[0].DPLL.THMA

Interface: MCS[i]

Name	DPLL_THMA
Address	$0x8424$
C–Name	

THMA	
Description	Maximal time between active and inactive TRIGGER slope; the time value corresponds to the time stamp clock counts: This means the clock selected for the TBU_CH0_BASE (see TBU_CH0_CTRL register)
Loop	–
Bit Range	$[15:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

THMA	
Reset group	DPLL_RAM_INIT: sync

NOT_USED	
Description	Not used
Loop	–
Bit Range	$[23 : 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.11 DPLL_THVAL

Description	Measured TRIGGER Hold Time Value
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_THVAL
Address	$0x8428$
C–Name	GTM.CLS[0].DPLL.THVAL

Interface: MCS[i]

Name	DPLL_THVAL
Address	$0x8428$
C–Name	

THVAL	
Description	Measured time from the last active slope to the next inactive TRIGGER slope in time stamp clock counts: This means the clock selected for the TBU_CH0_BASE;
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True

THVAL	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	The measured value considers all input slope filter delays. From the received input the corresponding filter delays are subtracted before the time stamp difference of active and inactive slope is calculated.

Note:
In the case of *LOW_RES* =1 and **DPLL_CTRL_1.TS0_HRT** =0 the difference between the time stamps of active and inactive slope is multiplied by 8. The register contains this value.

21.13.12 DPLL_RAM1B_RSVD_2

Description	DPLL RAM1B reserved data
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RAM1B_RSVD_2
Address	$0x842C$
C-Name	GTM.CLS[0].DPLL.RAM1B_RSVD_2

Interface: MCS[i]

Name	DPLL_RAM1B_RSVD_2
Address	$0x842C$
C-Name	

RSVD	
Description	Reserved Data word, no DPLL internal use.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync

21.13.13 DPLL_TOV

Description	Time Out Value of Active TRIGGER Slope (for missing TRIGGER generation)
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TOV
Address	$0x8430$
C-Name	GTM.CLS[0].DPLL.TOV

Interface: MCS[i]

Name	DPLL_TOV
Address	$0x8430$
C-Name	

TOV_DB	
Description	Decision value (fractional part) for missing TRIGGER interrupt.
Loop	-
Bit Range	$[9 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

TOV_DW	
Description	Decision value (integer part) for missing TRIGGER interrupt.
Loop	-
Bit Range	$[15 : 10]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

TOV_DW	
	DPLL_TOV [15:0] is to be multiplied with the duration of the last increment and divided by 1024 in order to get the time-out time value for a missing *TRIGGER* event **Note:** For the case of *LOW_RES* =1 (see **DPLL_STATUS** register as well), to calculate the time out value consider the following cases: *LOW_RES* =1 and **DPLL_CTRL_1** / **DPLL_CTRL_1.TS0_HRT** =1: multiply the *CCM[0]_TBU_TS0* value by 8 *LOW_RES* =1 and **DPLL_CTRL_1** / **DPLL_CTRL_1.TS0_HRT** =0: multiply the *CCM[0]_TBU_TS0* value by 8 multiply the estimated time point value (using **DPLL_TS_T** , **DPLL_DT_T_ACT.DT_T_ACT** and **DPLL_TOV**) by 8 *LOW_RES* =0 and **DPLL_CTRL_1** / **DPLL_CTRL_1.TS0_HRT** =0: use *CCM[0]_TBU_TS0* and the estimated time value unchanged.

NOT_USED	
Description	Not used
Loop	-
Bit Range	$[23:16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.14 DPLL_TOV_S

Description	Time Out Value of active STATE Slope (for missing STATE generation)
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TOV_S
Address	$0x8434$
C-Name	GTM.CLS[0].DPLL.TOV_S

Interface: MCS[i]

Name	DPLL_TOV_S
Address	$0x8434$

C-Name	

DB	
Description	Decision value (fractional part) for missing STATE interrupt.
Loop	-
Bit Range	[9 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

DW	
Description	Decision value (integer part) for missing STATE interrupt.
Loop	-
Bit Range	[15 : 10]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	DPLL_TOV_S [15:0] is to be multiplied with the duration of the last increment and divided by 1024 in order to get the time-out time value for a missing *STATE* event. **Note:** For the case of *LOW_RES* =1 (see **DPLL_STATUS** register as well) consider for the calculation of the time out value the following cases: *LOW_RES* =1 and **DPLL_CTRL_1** / **DPLL_CTRL_1.TS0_HRS** =1: multiply the *CCM[0]_TBU_TS0* value by 8 *LOW_RES* =1 and **DPLL_CTRL_1** / **DPLL_CTRL_1.TS0_HRS** =0: multiply the *CCM[0]_TBU_TS0* value by 8 multiply the estimated time point value (using **DPLL_TS_T** , **DPLL_DT_S_ACT.DT_S_ACT** and **DPLL_TOV_S**) by 8 *LOW_RES* =0 and **DPLL_CTRL_1** / **DPLL_CTRL_1.TS0_HRS** =0: use *CCM[0]_TBU_TS0* and the estimated time point value unchanged.

NOT_USED	
Description	Not used
Loop	-
Bit Range	[23 : 16]
Access Type	RW
Volatile	False
Multithread	False

NOT_USED	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.15 DPLL_ADD_IN_CAL1

Description	Calculated ADD_IN Value for SUB_INC1 Generation
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_ADD_IN_CAL1
Address	$0x8438$
C-Name	GTM.CLS[0].DPLL.ADD_IN_CAL1

Interface: MCS[i]

Name	DPLL_ADD_IN_CAL1
Address	$0x8438$
C-Name	

ADD_IN_CAL1	
Description	Calculated input value for SUB_INC1 generation, calculated by the DPLL.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync

ADD_IN_CAL1	
	Calculated value. The update of the *ADD_IN* value by the new calculated value **DPLL_ADD_IN_CAL1.ADD_IN_CAL1** is suppressed for one increment when an unexpected missing *TRIGGER* (**DPLL_CTRL_1.SMC** =1 or **DPLL_CTRL_0.RMO** =0) or an unexpected *STATE* (**DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =0) is detected.

21.13.16 DPLL_ADD_IN_CAL2

Description	Calculated ADD_IN Value for SUB_INC2 Generation
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_ADD_IN_CAL2
Address	$0x843C$
C-Name	GTM.CLS[0].DPLL.ADD_IN_CAL2

Interface: MCS[i]

Name	DPLL_ADD_IN_CAL2
Address	$0x843C$
C-Name	

ADD_IN_CAL2	
Description	Input value for SUB_INC2 generation, calculated by the DPLL for DPLL_CTRL_1.SMC=DPLL_CTRL_0.RMO=1.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated value. The update of the ADD_IN value by the calculated value **DPLL_ADD_IN_CAL2.ADD_IN_CAL2** is suppressed for one increment when an unexpected missing *STATE* (**DPLL_CTRL_0.RMO** = **DPLL_CTRL_1.SMC** =1) is detected.

21.13.17 DPLL_MPVAL1

Description	Missing Pulses to be Added or Subtracted Directly
Loop	

Condition	$NDPLL > 0$
Storage Type	RAM
Clock Active Cond	$DPLL\_CLK\_ENABLE == 1$

Interface: CPU

Name	DPLL_MPVAL1
Address	$0x8440$
C-Name	GTM.CLS[0].DPLL.MPVAL1

Interface: MCS[i]

Name	DPLL_MPVAL1
Address	$0x8440$
C-Name	

MPVAL1	
Description	Missing pulses for direct correction of SUB_INC1 pulses by the CPU;
Loop	-
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Used only for **DPLL_CTRL_0.RMO** =0 or **DPLL_CTRL_1.SMC** =1 for the case **DPLL_CTRL_1.PCM1** =1. Add **DPLL_MPVAL1.MPVAL1** once to **DPLL_INC_CNT1.INC_CNT1** and reset **DPLL_CTRL_1.PCM1** after applying once

SIX1	
Description	Sign extension for DPLL_MPVAL1.MPVAL1
Loop	-
Bit Range	$[23 : 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
RW-Coding	0x00 : **DPLL_MPVAL1.MPVAL1** is a positive number 0xFF : **DPLL_MPVAL1.MPVAL1** is a negative number

SIX1	
	Note: All bits must be written to either all zeros or all ones.

Note:

Do not provide negative values which exceed the number of **DPLL_ADT_T[p].NT** *(**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) or **DPLL_MLS1.MLS1** respectively; when considered negative **DPLL_ADT_T[p].PD** values the sum of both (**DPLL_MPVAL1.MPVAL1** + **DPLL_ADT_T[p].NT** * **DPLL_ADT_T[p].PD**) should not exceed the number of **DPLL_ADT_T[p].NT** *(**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) or **DPLL_MLS1.MLS1** respectively.

21.13.18 DPLL_MPVAL2

Description	Missing Pulses to be Added or Subtracted Directly
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_MPVAL2
Address	$0x8444$
C-Name	GTM.CLS[0].DPLL.MPVAL2

Interface: MCS[i]

Name	DPLL_MPVAL2
Address	$0x8444$
C-Name	

MPVAL2	
Description	Missing pulses for direct correction of SUB_INC2 pulses by the CPU;
Loop	-
Bit Range	$[15:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Used only for **DPLL_CTRL_1.SMC** = **DPLL_CTRL_0.RMO** =1 for the case **DPLL_CTRL_1.PCM2** =1. Add **DPLL_MPVAL2.MPVAL2** once to **DPLL_INC_CNT2.INC_CNT2** and reset **DPLL_CTRL_1.PCM2** after applying once **Note:** Do not provide negative values which exceed the number of **DPLL_MLS2.MLS2** ; when considered negative **DPLL_ADT_S[p].PD_S** values the sum of both (**DPLL_MPVAL2.MPVAL2** + **DPLL_ADT_S[p].NS** * **DPLL_ADT_S[p].PD_S**) should not exceed the number of **DPLL_MLS2.MLS2** .

SIX2	
Description	Sign extension for DPLL_MPVAL2.MPVAL2
Loop	–
Bit Range	$[23:16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
RW-Coding	0x00 : **DPLL_MPVAL2.MPVAL2** is a positive number 0xFF : **DPLL_MPVAL2.MPVAL2** is a negative number
	Note: All bits must be written to either all zeros or all ones.

21.13.19 DPLL_NMB_T_TAR

Description	Target Number of Pulses to be sent in Normal Mode
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NMB_T_TAR
Address	$0x8448$
C-Name	GTM.CLS[0].DPLL.NMB_T_TAR

Interface: MCS[i]

Name	DPLL_NMB_T_TAR
Address	$0x8448$
C-Name	

NMB_T_TAR	
Description	Target number of pulses for TRIGGER; calculated target number of pulses in normal mode for the current TRIGGER increment without missing pulses.
Loop	–
Bit Range	$[15:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–

NMB_T_TAR	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: The LSB address is determined using the **DPLL_OSW.SWON_T** .

NOT_USED	
Description	Not used
Loop	–
Bit Range	[23 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.20 DPLL_NMB_T_TAR_OLD

Description	Last but one Target Number of Pulses to be sent in Normal Mode
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NMB_T_TAR_OLD
Address	$0x844C$
C-Name	GTM.CLS[0].DPLL.NMB_T_TAR_OLD

Interface: MCS[i]

Name	DPLL_NMB_T_TAR_OLD
Address	$0x844C$
C-Name	

NMB_T_TAR_OLD	
Description	Target number of pulses for TRIGGER; calculated number of pulses in normal mode for the current TRIGGER increment without missing pulses.
Loop	–
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Calculated target pulse number **Note:** The LSB address is determined using the **DPLL_OSW.SWON_T** .

NOT_USED	
Description	Not used
Loop	–
Bit Range	$[23 : 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.21 DPLL_NMB_S_TAR

Description	Target Number of Pulses to be sent in Emergency Mode
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NMB_S_TAR

Address	$0x8450$
C-Name	GTM.CLS[0].DPLL.NMB_S_TAR

Interface: MCS[i]

Name	DPLL_NMB_S_TAR
Address	$0x8450$
C-Name	

NMB_S_TAR	
Description	Target number of pulses for STATE; calculated number of pulses in emergency mode for the current STATE increment without missing pulses.
Loop	-
Bit Range	$[19:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated target pulse number **Note:** The LSB address is determined using the **DPLL_OSW.SWON_S** .

NOT_USED	
Description	Not used
Loop	-
Bit Range	$[23:20]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.22 DPLL_NMB_S_TAR_OLD

Description	Last but one Target Number of Pulses to be sent in Emergency Mode

Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NMB_S_TAR_OLD
Address	$0x8454$
C-Name	GTM.CLS[0].DPLL.NMB_S_TAR_OLD

Interface: MCS[i]

Name	DPLL_NMB_S_TAR_OLD
Address	$0x8454$
C-Name	

NMB_S_TAR_OLD	
Description	Target number of pulses for STATE; calculated number of pulses in emergency mode for the current STATE increment without missing pulses.
Loop	-
Bit Range	$[19 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated target pulse number **Note:** The LSB address is determined using the **DPLL_OSW.SWON_S** .

NOT_USED	
Description	Not used
Loop	-
Bit Range	$[23 : 20]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

NOT_USED	
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.23 DPLL_RAM1B_RSVD_3_[k]

Description	DPLL RAM1B reserved data [k]
Loop	$k = \{n : 0 <= n <= 1\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RAM1B_RSVD_3_[k]
Address	$0x4 * k + 0x8458$
C-Name	GTM.CLS[0].DPLL.RAM1B_RSVD_3[k]

Interface: MCS[i]

Name	DPLL_RAM1B_RSVD_3_[k]
Address	$0x4 * k + 0x8458$
C-Name	

RSVD	
Description	Reserved Data word, no DPLL internal use.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

21.13.24 DPLL_RCDT_TX

Description	Reciprocal Value of the Expected Increment Duration of TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM

Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RCDT_TX
Address	$0x8460$
C-Name	GTM.CLS[0].DPLL.RCDT_TX

Interface: MCS[i]

Name	DPLL_RCDT_TX
Address	$0x8460$
C-Name	

RCDT_TX	
Description	Reciprocal value of expected increment duration *2^32 while only the lower 24 bits are used.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated value; when an overflow occurs in calculation the value is set to 0xFFFFFF.

21.13.25 DPLL_RCDT_SX

Description	Reciprocal Value of the Expected Increment Duration of STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RCDT_SX
Address	$0x8464$
C-Name	GTM.CLS[0].DPLL.RCDT_SX

Interface: MCS[i]

Name	DPLL_RCDT_SX
Address	$0x8464$

C-Name	

RCDT_SX	
Description	Reciprocal value of expected increment duration *2^32 while only the lower 24 bits are used.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated value; when an overflow occurs in calculation the value is set to 0xFFFFFF.

21.13.26 DPLL_RCDT_TX_NOM

Description	Reciprocal Value of the Expected Nominal Increment Duration of TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RCDT_TX_NOM
Address	$0x8468$
C-Name	GTM.CLS[0].DPLL.RCDT_TX_NOM

Interface: MCS[i]

Name	DPLL_RCDT_TX_NOM
Address	$0x8468$
C-Name	

RCDT_TX_NOM	
Description	Reciprocal value of expected increment duration *2^32 while only the lower 24 bits are used.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-

RCDT_TX_NOM	
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Calculated value; when an overflow occurs in calculation the value is set to 0xFFFFFF.

21.13.27 DPLL_RCDT_SX_NOM

Description	Reciprocal Value of the Expected Nominal Increment Duration of STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RCDT_SX_NOM
Address	$0x846C$
C-Name	GTM.CLS[0].DPLL.RCDT_SX_NOM

Interface: MCS[i]

Name	DPLL_RCDT_SX_NOM
Address	$0x846C$
C-Name	

RCDT_SX_NOM	
Description	Reciprocal value of expected increment duration *2^32 while only the lower 24 bits are used.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Calculated value; when an overflow occurs in calculation the value is set to 0xFFFFFF.

Note:

DPLL_RCDT_TX_NOM.RCDT_TX_NOM and **DPLL_RCDT_SX_NOM.RCDT_SX_NOM** are calculated by the values **DPLL_RCDT_TX.RCDT_TX** and **DPLL_RCDT_SX.RCDT_SX** to be multiplied with **DPLL_NUTC.SYN_T** or DPLL_NUSC_SYN_S respectively.

21.13.28 DPLL_RDT_T_ACT

Description	Reciprocal Value of the Last Increment of TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RDT_T_ACT
Address	$0x8470$
C-Name	GTM.CLS[0].DPLL.RDT_T_ACT

Interface: MCS[i]

Name	DPLL_RDT_T_ACT
Address	$0x8470$
C-Name	

RDT_T_ACT	
Description	Reciprocal value of last TRIGGER increment *2^32, only the lower 24 bits are used; the LSB is rounded up when the next truncated bit is 1.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated value; when an overflow occurs in calculation the value is set to 0xFFFFFF and the **DPLL_STATUS.CRO** bit is set (see chapter **DPLL_STATUS**).

21.13.29 DPLL_RDT_S_ACT

Description	Reciprocal Value of the Last Increment of STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RDT_S_ACT

Address	$0x8474$
C-Name	GTM.CLS[0].DPLL.RDT_S_ACT

Interface: MCS[i]

Name	DPLL_RDT_S_ACT
Address	$0x8474$
C-Name	

RDT_S_ACT	
Description	Reciprocal value of last STATE increment *2^32, only the lower 24 bits are used; the LSB is rounded up when the next truncated bit is 1.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated value; when an overflow occurs in calculation the value is set to 0xFFFFFF and the **DPLL_STATUS.CRO** bit is set (see chapter **DPLL_STATUS**).

21.13.30 DPLL_DT_T_ACT

Description	Duration of the Last TRIGGER Increment
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_DT_T_ACT
Address	$0x8478$
C-Name	GTM.CLS[0].DPLL.DT_T_ACT

Interface: MCS[i]

Name	DPLL_DT_T_ACT
Address	$0x8478$
C-Name	

DT_T_ACT	
Description	Calculated duration of the last TRIGGER increment.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Calculated duration of the last increment; Value will be written into the corresponding RAM field, when all calculations for the considered increment are done and **DPLL_APT.APT** is valid.

21.13.31 DPLL_DT_S_ACT

Description	Duration of the Last STATE Increment
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_DT_S_ACT
Address	$0x847C$
C-Name	GTM.CLS[0].DPLL.DT_S_ACT

Interface: MCS[i]

Name	DPLL_DT_S_ACT
Address	$0x847C$
C-Name	

DT_S_ACT	
Description	Calculated duration of the last STATE increment.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

DT_S_ACT	
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Calculated increment duration Value will be written into the corresponding RAM field, when all calculations for the considered increment are done and DPLL_APS_APS is valid.

21.13.32 DPLL_EDT_T

Description	Difference of Prediction to Actual Value of the Last TRIGGER Increment
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_EDT_T
Address	$0x8480$
C-Name	GTM.CLS[0].DPLL.EDT_T

Interface: MCS[i]

Name	DPLL_EDT_T
Address	$0x8480$
C-Name	

EDT_T	
Description	Signed difference between actual value and a simple prediction of the last TRIGGER increment:
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Calculated error value. See chapter *21.7.2.5 "Calculate the error of last prediction"* .

21.13.33 DPLL_MEDT_T

Description	Weighted Difference of Prediction Errors of TRIGGER

Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_MEDT_T
Address	$0x8484$
C-Name	GTM.CLS[0].DPLL.MEDT_T

Interface: MCS[i]

Name	DPLL_MEDT_T
Address	$0x8484$
C-Name	

MEDT_T	
Description	Signed middle weighted difference between actual value and prediction of the last TRIGGER increments; only calculated for DPLL_STATUS.SYT=1
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated medium error value, see chapter *21.7.2.6 "Calculate the weighted average error"* . The value is calculated only after synchronization (**DPLL_STATUS.SYT** =1) and the update is suppressed for one increment when an unexpected missing *TRIGGER* is detected.

21.13.34 DPLL_EDT_S

Description	Difference of Prediction to Actual Value of the Last STATE Increment
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_EDT_S
Address	$0x8488$

C-Name	GTM.CLS[0].DPLL.EDT_S

Interface: MCS[i]

Name	DPLL_EDT_S
Address	$0x8488$
C-Name	

EDT_S	
Description	Signed difference between actual value and prediction of the last STATE increment
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated error value, see chapter *21.7.3.5 "Calculate the error of last prediction"*

21.13.35 DPLL_MEDT_S

Description	Weighted Difference of Prediction Errors of STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_MEDT_S
Address	$0x848C$
C-Name	GTM.CLS[0].DPLL.MEDT_S

Interface: MCS[i]

Name	DPLL_MEDT_S
Address	$0x848C$
C-Name	

MEDT_S	
Description	Signed middle weighted difference between actual value and prediction of the last STATE increments; only calculated for DPLL_STATUS.SYS=1
Loop	-

MEDT_S	
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Calculated medium error value, see chapter *21.7.3.6 "Calculate the weighted average error"* . The value is calculated only after synchronization (**DPLL_STATUS.SYS** =1) and the update is suppressed for one increment when an unexpected missing *STATE* is detected.

21.13.36 DPLL_CDT_TX

Description	Prediction of the Actual TRIGGER Increment Duration
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CDT_TX
Address	$0x8490$
C–Name	GTM.CLS[0].DPLL.CDT_TX

Interface: MCS[i]

Name	DPLL_CDT_TX
Address	$0x8490$
C–Name	

CDT_TX	
Description	Calculated duration of the current TRIGGER increment.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

CDT_TX	
Reset group	DPLL_RAM_INIT: sync
	Calculated value

21.13.37 DPLL_CDT_SX

Description	Prediction of the Actual STATE Increment Duration
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CDT_SX
Address	$0x8494$
C-Name	GTM.CLS[0].DPLL.CDT_SX

Interface: MCS[i]

Name	DPLL_CDT_SX
Address	$0x8494$
C-Name	

CDT_SX	
Description	Calculated duration of the current STATE increment.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated value

21.13.38 DPLL_CDT_TX_NOM

Description	Prediction of the Nominal TRIGGER Increment Duration
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM

Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CDT_TX_NOM
Address	$0x8498$
C-Name	GTM.CLS[0].DPLL.CDT_TX_NOM

Interface: MCS[i]

Name	DPLL_CDT_TX_NOM
Address	$0x8498$
C-Name	

CDT_TX_NOM	
Description	Calculated duration of the current nominal TRIGGER event.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated value

21.13.39 DPLL_CDT_SX_NOM

Description	Prediction of the Nominal STATE Increment Duration
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CDT_SX_NOM
Address	$0x849C$
C-Name	GTM.CLS[0].DPLL.CDT_SX_NOM

Interface: MCS[i]

Name	DPLL_CDT_SX_NOM
Address	$0x849C$

C-Name	

CDT_SX_NOM	
Description	Calculated duration of the current nominal STATE event.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated value

21.13.40 DPLL_TLR

Description	TRIGGER Locking Range
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TLR
Address	$0x84A0$
C-Name	GTM.CLS[0].DPLL.TLR

Interface: MCS[i]

Name	DPLL_TLR
Address	$0x84A0$
C-Name	

TLR	
Description	Value is to be multiplied with the last nominal TRIGGER duration in order to get the range for the next TRIGGER event without setting DPLL_STATUS.TOR.
Loop	-
Bit Range	$[7:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-

TLR	
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Multiply value with the last nominal increment duration and check violation; when **DPLL_TLR.TLR** = 0 don't perform the check

NOT_USED	
Description	Not used
Loop	–
Bit Range	[23 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.41 DPLL_SLR

Description	STATE Locking Range
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_SLR
Address	$0x84A4$
C-Name	GTM.CLS[0].DPLL.SLR

Interface: MCS[i]

Name	DPLL_SLR
Address	$0x84A4$
C-Name	

SLR	
Description	Value is to be multiplied with the last nominal STATE duration in order to get the range for the next STATE event without setting DPLL_STATUS.SOR

SLR	
Loop	–
Bit Range	$[7:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Multiply value with the last nominal increment duration and check violation; when **DPLL_SLR.SLR** = 0 don't perform the check.

NOT_USED	
Description	Not used
Loop	–
Bit Range	$[23:8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.42 DPLL_RAM1B_RSVD_4_[k]

Description	DPLL RAM1B reserved data [k]
Loop	$k = \{n : 0 <= n <= 21\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RAM1B_RSVD_4_[k]
Address	$0x4 * k + 0x84A8$
C-Name	GTM.CLS[0].DPLL.RAM1B_RSVD_4[k]

Interface: MCS[i]

Name	DPLL_RAM1B_RSVD_4_[k]
Address	$0x4 * k + 0x84A8$
C-Name	

RSVD	
Description	Reserved data word, no DPLL internal use.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

21.13.43 DPLL_PDT_[n]

Description	Projected Increment Sum Relations for Action [n]
Loop	$n = \{n : 0 <= n <= \text{NOAC} - 1\}$
Condition	$\text{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\text{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PDT_[n]
Address	$0x4 * n + 0x8500$
C-Name	GTM.CLS[0].DPLL.PDT[n]

Interface: MCS[i]

Name	DPLL_PDT_[n]
Address	$0x4 * n + 0x8500$
C-Name	

DB	
Description	fractional part of relation between TRIGGER and STATE increments.
Loop	-
Bit Range	$[13:0]$
Access Type	RW
Volatile	True
Multithread	False

DB	
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync

DW	
Description	Integer part of relation between TRIGGER and STATE increments.
Loop	–
Bit Range	[23 : 14]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Definition of relation values between *TRIGGER* and *STATE* increments **DPLL_PDT_[n]** according to equation *21.8.1.1 "For DPLL_NUTC.NUTE – DPLL_NUTC.VTN > NA[n] (part w)"* or equation *21.8.3.1 "For DPLL_NUSC_NUSE – DPLL_NUSC_VSN > DPLL_NA[i] (part w)"* or similar

21.13.44 DPLL_RAM1B_RSVD_5_[k]

Description	DPLL RAM1B reserved data [k]
Loop	$k = \{n : 0 <= n <= 15\}$
Condition	NDPLL > 0
Storage Type	RAM
Clock Active Cond	DPLL_CLK_ENABLE $== 1$

Interface: CPU

Name	DPLL_RAM1B_RSVD_5_[k]
Address	$0x4 * k + 0x8580$
C–Name	GTM.CLS[0].DPLL.RAM1B_RSVD_5[k]

Interface: MCS[i]

Name	DPLL_RAM1B_RSVD_5_[k]
Address	$0x4 * k + 0x8580$
C–Name	

RSVD	
Description	Reserved data word, no DPLL internal use.

RSVD	
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync

21.13.45 DPLL_MLS1

Description	Calculated Number of Sub-Pulses between two nominal STATE Events for DPLL_CTRL_1.SMC = 0
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_MLS1
Address	$0x85C0$
C-Name	GTM.CLS[0].DPLL.MLS1

Interface: MCS[i]

Name	DPLL_MLS1
Address	$0x85C0$
C-Name	

MLS1	
Description	Number of pulses between two STATE events (to be set and updated by the CPU).
Loop	–
Bit Range	$[17:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync

MLS1	
	For **DPLL_CTRL_1.SMC** =0 the value of **DPLL_MLS1.MLS1** is calculated once by the CPU for fixed values in the **DPLL_CTRL_0** register by the formula **DPLL_MLS1.MLS1** = ((**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1)*(**DPLL_CTRL_0.TNU** +1)/(**DPLL_CTRL_0.SNU** +1)) and set accordingly FOR **DPLL_CTRL_1.SMC** =1 the value of **DPLL_MLS1.MLS1** represents the number of pulses between two nominal *TRIGGER* events (to be set and updated by the CPU)

NOT_USED	
Description	Not used
Loop	–
Bit Range	$[23 : 18]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.46 DPLL_MLS2

Description	Calculated Number of Sub−Pulses between two nominal STATE Events for DPLL_CTRL_1.SMC=1 and RMO=1
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_MLS2
Address	$0x85C4$
C−Name	GTM.CLS[0].DPLL.MLS2

Interface: MCS[i]

Name	DPLL_MLS2
Address	$0x85C4$
C−Name	

MLS2	
Description	Number of pulses between two STATE events (to be set and updated by the CPU).
Loop	–

MLS2	
Bit Range	$[17:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Using adapt information and the missing *STATE* event information DPLL_NUSC_SYN_S, this value can be corrected for each increment automatically.

NOT_USED	
Description	Not used
Loop	–
Bit Range	$[23:18]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.47 DPLL_CNT_NUM_1

Description	Number of SUB_INC1 pulses in continuous mode
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CNT_NUM_1
Address	$0x85C8$
C-Name	GTM.CLS[0].DPLL.CNT_NUM_1

Interface: MCS[i]

Name	DPLL_CNT_NUM_1
Address	$0x85C8$
C-Name	

CNT_NUM_1	
Description	Counter for number of SUB_INC1 pulses; Number of pulses in continuous mode for a nominal increment in normal and emergency mode for SUB_INC1, given and updated by CPU only.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Count value for continuous mode

21.13.48 DPLL_CNT_NUM_2

Description	Number of SUB_INC2 pulses in continuous mode
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_CNT_NUM_2
Address	$0x85CC$
C-Name	GTM.CLS[0].DPLL.CNT_NUM_2

Interface: MCS[i]

Name	DPLL_CNT_NUM_2
Address	$0x85CC$
C-Name	

CNT_NUM_2	
Description	Counter for number of SUB_INC2 pulses; Number of pulses in continuous mode for a nominal increment in normal and emergency mode for SUB_INC2, given and updated by CPU only.
Loop	-
Bit Range	$[23:0]$
Access Type	RW

CNT_NUM_2	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Count value for continuous mode

21.13.49 DPLL_PVT

Description	Plausibility Value of Next TRIGGER Slope
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PVT
Address	$0x85D0$
C-Name	GTM.CLS[0].DPLL.PVT

Interface: MCS[i]

Name	DPLL_PVT
Address	$0x85D0$
C-Name	

PVT	
Description	Plausibility value of next active TRIGGER slope.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync

PVT	
	The meaning of the value depends on the value of **DPLL_CTRL_1_SHADOW_TRIGGER.PIT** . For **DPLL_CTRL_1.PIT** =0: the number of *SUB_INC1* pulses to be waited for until a next active *TRIGGER* event is accepted. For **DPLL_CTRL_1_SHADOW_TRIGGER.PIT** =1: **DPLL_PVT.PVT** is to be multiplied with the last nominal increment time **DPLL_DT_T_ACT.DT_T_ACT** and divided by 1024 and reduced to a 24 bit value in order to get the time to be waited for until the next active *TRIGGER* event is accepted. The wait time must be exceeded for an active slope. **Note:** When an active *TRIGGER* slope is detected while the wait condition is not fulfilled the interrupt *DPLL_PWI* is generated. Please note, that the **DPLL_CTRL_1.SGE1** must be set, when **DPLL_CTRL_1.PIT** =0 in order to provide the necessary *SUB_INC1* pulses for checking. After an unexpected missing *TRIGGER* the plausibility check is suppressed for the following increment. In case of direction change the **DPLL_PVT.PVT** value is automatically set to zero in order to deactivate the check.

21.13.50 DPLL_RAM1B_RSVD_6_[k]

Description	DPLL RAM1B reserved data [k]
Loop	$k = \{n : 0 <= n <= 2\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RAM1B_RSVD_6_[k]
Address	$0x4 * k + 0x85D4$
C-Name	GTM.CLS[0].DPLL.RAM1B_RSVD_6[k]

Interface: MCS[i]

Name	DPLL_RAM1B_RSVD_6_[k]
Address	$0x4 * k + 0x85D4$
C-Name	

RSVD	
Description	Reserved data word, no DPLL internal use.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

21.13.51 DPLL_PSTC

Description	Actual Calculated Position Stamp of TRIGGER
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PSTC
Address	$0x85E0$
C-Name	GTM.CLS[0].DPLL.PSTC

Interface: MCS[i]

Name	DPLL_PSTC
Address	$0x85E0$
C-Name	

PSTC	
Description	Calculated position stamp of last TRIGGER input;
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Value is set by the DPLL and can be updated by the CPU when filter values are to be considered for the exact position (see DPLL Status and DPLL Control registers for explanation of the status and control bits used). For each active slope of *TRIGGER* in normal mode When **DPLL_STATUS.FTD** =0: **DPLL_PSTC.PSTC** is set from actual position value, for the first active *TRIGGER* event (no filter delay considered) the CPU must update the value once, taking into account the filter value When **DPLL_STATUS.FTD** =1: **DPLL_PSTC.PSTC** is incremented at each *TRIGGER* event by **DPLL_CTRL_1.SMC** =0: ((**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) + **DPLL_ADT_T[p].PD**) * **DPLL_NUTC.SYN_T** ; while **DPLL_ADT_T[p].PD** =0 for **DPLL_CTRL_0_SHADOW_TRIGGER.AMT** =0 **DPLL_CTRL_1.SMC** =1: (**DPLL_MLS1.MLS1** + **DPLL_ADT_T[p].PD**)*(**DPLL_NUTC.SYN_T**); while **DPLL_ADT_T[p].PD** =0 for **DPLL_CTRL_0_SHADOW_TRIGGER.AMT** =0

21.13.52 DPLL_PSSC

Description	Actual Calculated Position Stamp of STATE
Loop	
Condition	$\mathrm{NDPLL} > 0$

Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PSSC
Address	$0x85E4$
C-Name	GTM.CLS[0].DPLL.PSSC

Interface: MCS[i]

Name	DPLL_PSSC
Address	$0x85E4$
C-Name	

PSSC	
Description	Calculated position stamp for the last STATE input;
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	First value is set by the DPLL and can be updated by the CPU when the filter delay is to be considered. For each active slope of *STATE* in emergency mode: When **DPLL_STATUS.FSD** =0: **DPLL_PSSC.PSSC** is set from actual position value(no filter delay considered), the CPU must update the value once, taking into account the filter value When **DPLL_STATUS.FSD** =1: at each active slope of *STATE* (PD_S_store=0 for **DPLL_CTRL_0_SHADOW_STATE.AMS** =0): **DPLL_CTRL_1.SMC** =0: add (**DPLL_MLS1.MLS1** + PD_S_store)*(DPLL_NUSC_SYN_S); **DPLL_CTRL_1.SMC** =1: add (**DPLL_MLS2.MLS2** + PD_S_store)*(DPLL_NUSC_SYN_S);

21.13.53 DPLL_PSTM

Description	Measured Position Stamp at Last TRIGGER Input
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PSTM
Address	$0x85E8$

C–Name	GTM.CLS[0].DPLL.PSTM

Interface: MCS[i]

Name	DPLL_PSTM
Address	$0x85E8$
C–Name	

PSTM	
Description	Position stamp of TRIGGER, measured; Measured position stamp of last active TRIGGER input.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Store the value *CCM[0]_TBU_TS1* when an active *TRIGGER* event occurs. The value of **DPLL_PSTM.PSTM** is invalid for (**DPLL_CTRL_0.RMO** =1 and **DPLL_CTRL_1.SMC** =0).

Note:
The LSB address is determined using the **DPLL_OSW.SWON_T** .

21.13.54 DPLL_PSTM_OLD

Description	Measured Position Stamp at Last but one TRIGGER Input
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PSTM_OLD
Address	$0x85EC$
C–Name	GTM.CLS[0].DPLL.PSTM_OLD

Interface: MCS[i]

Name	DPLL_PSTM_OLD
Address	$0x85EC$
C–Name	

PSTM_OLD	
Description	Last but one position stamp of TRIGGER, measured; Measured position stamp of last but one active TRIGGER input.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Last **DPLL_PSTM.PSTM** value: see explanation of **DPLL_PSTM.PSTM**

Note:
The LSB address is determined using the **DPLL_OSW.SWON_T** .

21.13.55 DPLL_PSSM

Description	Measured Position Stamp at Last STATE Input
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PSSM
Address	$0x85F0$
C-Name	GTM.CLS[0].DPLL.PSSM

Interface: MCS[i]

Name	DPLL_PSSM
Address	$0x85F0$
C-Name	

PSSM	
Description	Position stamp of STATE, measured; Measured position stamp of last active STATE input.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

PSSM	
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Store the value *CCM[0]_TBU_TS1* or *CCM[0]_TBU_TS2* respectively at the moment when an active *STATE* event occurs. The value of **DPLL_PSSM.PSSM** is invalid for (**DPLL_CTRL_0.RMO** =0 and **DPLL_CTRL_1.SMC** =0).

Note:
The LSB address is determined using the **DPLL_OSW.SWON_S** .

21.13.56 DPLL_PSSM_OLD

Description	Measured Position Stamp at Last but one STATE Input
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_PSSM_OLD
Address	$0x85F4$
C–Name	GTM.CLS[0].DPLL.PSSM_OLD

Interface: MCS[i]

Name	DPLL_PSSM_OLD
Address	$0x85F4$
C–Name	

PSSM_OLD	
Description	Last but one position stamp of STATE, measured; Measured position stamp of last but one active STATE input.
Loop	–
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Last **DPLL_PSSM.PSSM** value: see explanation of **DPLL_PSSM.PSSM**

Note:

The LSB address is determined using the **DPLL_OSW.SWON_S** .

21.13.57 DPLL_NMB_T

Description	Number of Pulses to be sent in Normal Mode
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NMB_T
Address	$0x85F8$
C-Name	GTM.CLS[0].DPLL.NMB_T

Interface: MCS[i]

Name	DPLL_NMB_T
Address	$0x85F8$
C-Name	

NMB_T	
Description	Number of pulses for TRIGGER; calculated number of pulses in normal mode for the current TRIGGER increment.
Loop	-
Bit Range	$[15:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Calculated pulse number

NOT_USED	
Description	Not used
Loop	-
Bit Range	$[23:16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

NOT_USED	
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.58 DPLL_NMB_S

Description	Number of Pulses to be sent in Emergency Mode
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_NMB_S
Address	$0x85FC$
C-Name	GTM.CLS[0].DPLL.NMB_S

Interface: MCS[i]

Name	DPLL_NMB_S
Address	$0x85FC$
C-Name	

NMB_S	
Description	NMB_S: Number of pulses for STATE; calculated number of pulses in emergency mode for the current STATE increment.
Loop	–
Bit Range	$[19 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Calculated pulse number

NOT_USED	
Description	Not used
Loop	–

NOT_USED	
Bit Range	$[23 : 20]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

21.13.59 DPLL_RDT_S[p]

Description	Reciprocal Values of the Nominal STATE Increment Duration in FULL_SCALE for maximum [p] entries in profile for STATE (DPLL_ADT_S[p])
Loop	$p = \{n : 0 <= n <= 63\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_RDT_S[p]
Address	$0x4 * p + 0x8600$
C–Name	GTM.CLS[0].DPLL.RDT_S[p]

Interface: MCS[i]

Name	DPLL_RDT_S[p]
Address	$0x4 * p + 0x8600$
C–Name	

RDT_S	
Description	Reciprocal difference time of STATE; nominal reciprocal value of the number of time stamp clocks measured in the corresponding increment *2^32 while only the lower 24 bits are used; no gap considered. The LSB is rounded up when the next truncated bit is 1.
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

RDT_S	
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Note: There are 2*(**DPLL_CTRL_0.SNU** +1- **DPLL_CTRL_1.SYN_NS**) entries for **DPLL_CTRL_1.SYSF** =0 or 2*(**DPLL_CTRL_0.SNU** +1)- **DPLL_CTRL_1.SYN_NS** entries for **DPLL_CTRL_1.SYSF** =1 respectively.

Note:
If **DPLL_CTRL_11.STATE_EXT** is set, this memory range is not used by the DPLL, but emulated outside the DPLL. The DPLL will access the MCS to DPLL interface chapter *22.3 " MCS to DPLL Registers Description"* and will expect data to be correctly stored there. This means in fact, that the handling of the **DPLL_RDT_S[p].RDT_S** values has to be done outside, in the MCS integrated in the same cluster.

21.13.60 DPLL_TSF_S[p]

Description	Time Stamp Values of the Nominal STATE Events in FULL_SCALE for maximum [p] entries in profile for STATE (DPLL_ADT_S[p])
Loop	$p = \{n : 0 <= n <= 63\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_TSF_S[p]
Address	$0x4 * p + 0x8700$
C-Name	GTM.CLS[0].DPLL.TSF_S[p]

Interface: MCS[i]

Name	DPLL_TSF_S[p]
Address	$0x4 * p + 0x8700$
C-Name	

TSF_S	
Description	Time stamp field of STATE; Time stamp value of each active STATE event.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

TSF_S	
	Note: There are 2* (**DPLL_CTRL_0.SNU** +1) entries.

Note:

If **DPLL_CTRL_11.STATE_EXT** is set, this memory range is not used by the DPLL, but emulated outside the DPLL. The DPLL will access the MCS to DPLL interface chapter *22.3 " MCS to DPLL Registers Description"* and will expect data to be correctly stored there. This means in fact, that the handling of the **DPLL_TSF_S[p].TSF_S** values has to be done outside, in the MCS integrated in the same cluster.

21.13.61 DPLL_ADT_S[p]

Description	Adapt and Profile Values of the STATE Increments in FULL_SCALE for maximum [p] entries
Loop	$p = \{n : 0 <= n <= 63\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_ADT_S[p]
Address	$0x4 * p + 0x8800$
C-Name	GTM.CLS[0].DPLL.ADT_S[p]

Interface: MCS[i]

Name	DPLL_ADT_S[p]
Address	$0x4 * p + 0x8800$
C-Name	

PD_S	
Description	Physical deviation of STATE; Adapt values for each nominal STATE increment in FULL_SCALE;
Loop	-
Bit Range	$[15 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	This value represents the number of pulses to be added to the corresponding nominal increment. The absolute value of a negative **DPLL_ADT_S[p].PD_S** must not exceed **DPLL_MLS1.MLS1** or **DPLL_MLS2.MLS2** respectively. The **DPLL_ADT_T[p].PD** value means the number of *SUB_INC1* pulses per nominal tooth to be added to **DPLL_ADT_S[p].NS** *((**DPLL_MLS1.MLS1** / **DPLL_MLS2.MLS2** +1) + **DPLL_ADT_S[p].PD_S**);

NS	
Description	Number of STATE events; number of nominal STATE parts in the corresponding increment.
Loop	–
Bit Range	$[21 : 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: There are 2*(**DPLL_CTRL_0.SNU** +1− **DPLL_CTRL_1.SYN_NS**) entries for **DPLL_CTRL_1.SYSF** =0 or 2*(**DPLL_CTRL_0.SNU** +1)− **DPLL_CTRL_1.SYN_NS** entries for **DPLL_CTRL_1.SYSF** =1 respectively.

NOT_USED	
Description	Not used
Loop	–
Bit Range	$[23 : 22]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

Note:
If **DPLL_CTRL_11.STATE_EXT** is set, this memory range is not used by the DPLL, but emulated outside the DPLL. The DPLL will access the MCS to DPLL interface chapter *22.3 " MCS to DPLL Registers Description"* and will expect data to be correctly stored there. This means in fact, that the handling of the **DPLL_ADT_S[p]** values has to be done outside, in the MCS integrated in the same cluster.

21.13.62 DPLL_DT_S[p]

Description	Nominal STATE Increment Duration in FULL_SCALE for maximum [p] entries in profile for STATE (DPLL_ADT_S[p])
Loop	$p = \{n : 0 <= n <= 63\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM
Clock Active Cond	$\mathrm{DPLL\_CLK\_ENABLE} == 1$

Interface: CPU

Name	DPLL_DT_S[p]
Address	$0x4 * p + 0x8900$
C-Name	GTM.CLS[0].DPLL.DT_S[p]

Interface: MCS[i]

Name	DPLL_DT_S[p]
Address	$0x4 * p + 0x8900$
C-Name	

DT_S	
Description	Difference time of STATE; nominal increment duration values for each STATE increment in FULL_SCALE (considering no gap).
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync
	Note: There are 2*(**DPLL_CTRL_0.SNU** +1- **DPLL_CTRL_1.SYN_NS**) entries for **DPLL_CTRL_1.SYSF** =0 or 2*(**DPLL_CTRL_0.SNU** +1)- **DPLL_CTRL_1.SYN_NS** entries for **DPLL_CTRL_1.SYSF** =1 respectively.

Note:
If **DPLL_CTRL_11.STATE_EXT** is set, this memory range is not used by the DPLL, but emulated outside the DPLL. The DPLL will access the MCS to DPLL interface chapter *22.3 " MCS to DPLL Registers Description"* and will expect data to be correctly stored there. This means in fact, that the handling of the **DPLL_DT_S[p].DT_S** values has to be done outside, in the MCS integrated in the same cluster.

21.14 DPLL RAM Region 2 value description

Note:
Bits 31 to 24 of RAM region 2 are not implemented and therefore always read as zero (reserved). Other bits which are declared as reserved are not protected against writing. Unused address regions are not protected against writing when implemented.

21.14.1 DPLL_RR2

Description	DPLL memory RR2
Loop	
Condition	$NDPLL > 0$
Size	DPLL_RR2_MEM_SIZE
Clock Active Cond	$DPLL\_CLK\_ENABLE == 1$

Interface: CPU

Name	DPLL_RR2
Address	$0xC000$
C-Name	GTM.CLS[0].DPLL.RR2

Interface: MCS[i]

Name	DPLL_RR2
Address	$0xC000$
C-Name	

DATA	
Description	Data
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

21.14.2 DPLL_RDT_T[p]

Description	Reciprocal Values of the Nominal TRIGGER Increments Duration in FULL_SCALE for maximum [p] entries in profile for TRIGGER (DPLL_ADT_T[p])
Loop	$p = \{n : 0 <= n <= 1023\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM

RDT_T	
Description	Reciprocal difference time of TRIGGER; 2* (TNU+1- SYN_NT) stored values nominal reciprocal value of the number of time stamp clocks measured in the corresponding increment (which is divided by the number of nominal increments); multiplied by *2^32 while only the lower 24 bits are used; the LSB is rounded up, when the next truncated bit is 1.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	DPLL_RAM_INIT: sync

RDT_T	
	Note: There are 2* (**DPLL_CTRL_0.TNU** +1− **DPLL_CTRL_1.SYN_NT**) entries. The maximum number of entries is restricted to a value corresponding to the **DPLL_OSW.OSS** value. The maximum number of entries can be calculated as 128 * (2** **DPLL_OSW.OSS**).

Note:
The starting index for Memory **DPLL_RDT_T[p]** in RAM2 is defined by the parameter **DPLL_AOSV_2.AOSV_2A** .

21.14.3 DPLL_TSF_T[p]

Description	Time Stamp Values of the Nominal TRIGGER Increments in FULL_SCALE for maximum [p] entries in profile for TRIGGER (DPLL_ADT_T[p])
Loop	$p = \{n : 0 <= n <= 1023\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM

TSF_T	
Description	Time stamp field of active TRIGGER slopes
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: There are 2* (**DPLL_CTRL_0.TNU** +1) entries. The maximum number of entries is restricted to a value corresponding to the **DPLL_OSW.OSS** value. The maximum number of entries can be calculated as 128 * (2** **DPLL_OSW.OSS**).

Note:
The starting index for Memory **DPLL_TSF_T[p]** in RAM2 is defined by the parameter **DPLL_AOSV_2.AOSV_2B**

21.14.4 DPLL_ADT_T[p]

Description	Adapt and Profile Values of the TRIGGER Increments in FULL_SCALE for maximum [p] entries
Loop	$p = \{n : 0 <= n <= 1023\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM

PD	
Description	Physical deviation; Adapt values for each nominal TRIGGER increment in FULL_SCALE;
Loop	–

PD	
Bit Range	[12 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	The **DPLL_ADT_T[p].PD** value means the number of *SUB_INC1* pulses to be added to **DPLL_ADT_T[p].NT** *((**DPLL_CTRL_0_SHADOW_TRIGGER.MLT** +1) + **DPLL_ADT_T[p].PD**); the absolute value of a negative **DPLL_ADT_T[p].PD** must not exceed (**DPLL_CTRL_0_SHADOW_TRIGGER.-MLT** +1) or **DPLL_MLS1.MLS1** respectively; systematic missing *TRIGGER* events must be considered for the value of **DPLL_ADT_T[p].PD** ;

TINT	
Description	TRIGGER Interrupt information;
Loop	–
Bit Range	[15 : 13]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Depending on the value up to 7 different interrupts can be generated. In the current version the 5 interrupts *DPLL_TE0_IRQ* ... *DPLL_TE4_IRQ* are supported by **DPLL_ADT_T[p].TINT** ="001", "010", "011", "100", "101" respectively. For the values "000", "110" and "111" no interrupt is generated and no other reaction is performed. The corresponding interrupt is activated, when the **DPLL_ADT_T[p].TINT** value is read by the DPLL together with the other values (**DPLL_ADT_T[p].PD** , **DPLL_ADT_T[p].NT**) according to the profile.

NT	
Description	Number of TRIGGER events; number of nominal TRIGGER parts in the corresponding increment.
Loop	–
Bit Range	[18 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

NT	
Reset group	DPLL_RAM_INIT: sync
	Note: There are 2* (**DPLL_CTRL_0.TNU** +1− **DPLL_CTRL_1.SYN_NT**) entries. The maximum number of entries is restricted to a value corresponding to the **DPLL_OSW.OSS** value. The maximum number of entries can be calculated as 128 * (2** **DPLL_OSW.OSS**).

NOT_USED	
Description	Not used
Loop	–
Bit Range	$[23 : 19]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	DPLL_RAM_INIT: sync
	Note: Bit field is implemented in DPLL memory due to memory bit width of 24. It can be written to any value, but it must be ensured that the value is zero before DPLL is enabled.

Note:
The starting index for Memory **DPLL_ADT_T[p]** in RAM2 is defined by the parameter **DPLL_AOSV_2.AOSV_2C**

21.14.5 DPLL_DT_T[p]

Description	Nominal TRIGGER Increments Duration in FULL_SCALE for maximum [p] entries in profile for TRIGGER (DPLL_ADT_T[p])
Loop	$p = \{n : 0 <= n <= 1023\}$
Condition	$\mathrm{NDPLL} > 0$
Storage Type	RAM

DT_T	
Description	Difference time of TRIGGER; increment duration values for each TRIGGER increment in FULL_SCALE divided by the number of nominal increments (nominal value).
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

DT_T	
Reset group	DPLL_RAM_INIT: sync
	Note: There are 2* (**DPLL_CTRL_0.TNU** +1− **DPLL_CTRL_1.SYN_NT**) entries. The maximum number of entries is restricted to a value corresponding to the **DPLL_OSW.OSS** value. The maximum number of entries can be calculated as 128 * (2** **DPLL_OSW.OSS**).

Note:
The starting index for Memory **DPLL_DT_T[p]** in RAM2 is defined by the parameter **DPLL_AOSV_2.AOSV_2D**

21.15 DPLL Signal Description

21.15.1 DPLL reset

Loop	–
Condition	–
Format	–

DPLL_DEEP_RESET	
Description	DPLL deep reset active signal. Resets all bit fields in DPLL.
Loop	–
Condition	–
Signal Type	INT
Assignment	DPLL_CTRL_1.SWR == 1 && DPLL_CTRL_1.DEN == 0

DPLL_RESET	
Description	DPLL reset active signal. Resets a subset of DPLL bit fields.
Loop	–
Condition	–
Signal Type	INT
Assignment	DPLL_CTRL_1.DEN == 0

21.15.2 DPLL signals

Loop	–
Condition	–
Format	–

PMTR[n]	
Description	Data input of "Position Minus Time Request" of channel [n]
Loop	$n = \{n : 1 <= n <= \mathrm{NOAC}\}$
Condition	–
Signal Type	ARRAY[32]
Assignment	–

ACT_D[m]	
Description	ACB bits of ACT_D (see DPLL_51020)
Loop	$m = \{n : 1 <= n <= \mathrm{NOAC}\}$

ACT_D[m]	
Condition	–
Signal Type	ARRAY[32]
Assignment	–

AEN[n]	
Description	Action enable of action channel [n]
Loop	$n = \{n : 1 <= n <= \mathrm{NOAC}\}$
Condition	–
Signal Type	ARRAY[32]
Assignment	–

RPCU_CH[x]	
Description	Number of rapid pulse corrections of pulse generator channel [x]
Loop	$x = \{n : 1 <= n <= 2\}$
Condition	–
Signal Type	ARRAY[2]
Assignment	–

RESET_SIG[x]	
Description	Signal of the pulse generator (see DPLL_1450) is activated for each new active input slope in order to reset the register values for pulse generator channel [x]
Loop	$x = \{n : 1 <= n <= 2\}$
Condition	–
Signal Type	ARRAY[2]
Assignment	–

EN_C2U	
Description	Enable of sub increment generator 2
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EN_C2C	
Description	Gating of sub increment pulses of sub increment generator 2 when target number of pulses is reached in automatic end mode
Loop	–
Condition	–
Signal Type	INT
Assignment	–

EN_C1U	
Description	Enable of sub increment generator 1
Loop	–
Condition	–
Signal Type	INT

EN_C1U	
Assignment	–

EN_C1C	
Description	Gating of sub increment pulses of sub increment generator 1 when target number of pulses is reached in automatic end mode
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ADT_T	
Description	Adapt and profile values of TRIGGER increments
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ADT_S	
Description	Adapt and profile values of STATE increments
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ADD_IN_CAL1	
Description	The value stored in DPLL_ADD_IN_CAL1.ADD_IN_CAL1
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ADD_IN	
Description	Input signal of pulse generator according to
Loop	–
Condition	–
Signal Type	INT
Assignment	–

21.15.3 DPLL_RAM_INIT

Loop	$j = \{n : 0 <= n <= \mathrm{NDPLL} - 1\}$
Condition	–
Format	–

DPLL_RAM_INIT	
Description	DPLL memory initialization
Loop	–

DPLL_RAM_INIT	
Condition	–
Signal Type	INT
Assignment	DPLL_RAM_INI.INIT_RAM == 1 && DPLL_CTRL_1.DEN == 0

21.15.4 Action enable signals

21.15.4.1 dpll_Action_enable

Loop	$j = \{n : 0 <= n <= \mathrm{NOAC} - 1\}$
Condition	–
Format	–

DPLL_ACTION_ENABLE[-m]	
Description	DPLL Action [m] enable
Loop	$x = \{n : 0 <= n <= 3\}; n = \{n : 0 <= n <= 7\}; m = \{n : x * 8 + n <= n <= x * 8 + n\}$
Condition	–
Signal Type	ARRAY[NOAC]
Assignment	(x == 0 && DPLL_CTRL_2.AEN[n]) || (x == 1 && DPLL_CTRL_3.AEN[n]) || (x == 2 && DPLL_CTRL_4.AEN[n]) || (x == 3 && DPLL_CTRL_5.AEN[n])

21.16 DPLL Port Description

21.16.1 DPLL Interrupt ports

Loop	–
Condition	–
Format	–

DPLL_W1I	
Description	Interrupt port: Write access to RAM region 1b or 1c interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_W2I	
Description	Interrupt port: Write access to RAM region 2 interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC

DPLL_W2I	
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_CDSI	
Description	Interrupt port: STATE duration calculated for last increment
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_CDTI	
Description	Interrupt port: TRIGGER duration calculated for last increment
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TORI	
Description	Interrupt port: TRIGGER is out of range
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TISI	
Description	Interrupt port: TRIGGER inactive slope interrupt request

DPLL_TISI	
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TINI	
Description	Interrupt port: TRIGGER minimum hold time violation interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TAXI	
Description	Interrupt port: TRIGGER maximum hold time violation interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TASI	
Description	Interrupt port: TRIGGER active slope while DPLL_NTI_CNT.NTI_CNT is zero interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET

DPLL_TASI	
Reset Value	–

DPLL_SORI	
Description	Interrupt port: STATE is out of range
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_SISI	
Description	Interrupt port: STATE inactive slope interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_SASI	
Description	Interrupt port: STATE active slope interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_PWI	
Description	Interrupt port: Plausibility window violation interrupt of TRIGGER request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT

DPLL_PWI	
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_PEI	
Description	Interrupt port: DPLL enable interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_PDI	
Description	Interrupt port: DPLL disable interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_MTI	
Description	Interrupt port: Missing TRIGGER interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_MSI	
Description	Interrupt port: Missing STATE interrupt request
Loop	–

DPLL_MSI	
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_LL2I	
Description	Interrupt port: Loss of lock interrupt for SUB_INC2 request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_LL1I	
Description	Interrupt port: Loss of lock interrupt for SUB_INC1 request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_GL2I	
Description	Interrupt port: Get lock interrupt for SUB_INC2 request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET

DPLL_GL2I	
Reset Value	–

DPLL_GL1I	
Description	Interrupt port: Get lock interrupt for SUB_INC1 request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_EI	
Description	Interrupt port: Error interrupt request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_DCGI	
Description	Interrupt port: Direction change
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TE4_IRQ	
Description	Interrupt port: TRIGGER event interrupt 4 request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT

DPLL_TE4_IRQ	
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TE3_IRQ	
Description	Interrupt port: TRIGGER event interrupt 3 request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TE2_IRQ	
Description	Interrupt port: TRIGGER event interrupt 2 request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TE1_IRQ	
Description	Interrupt port: TRIGGER event interrupt 1 request
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DPLL_TE0_IRQ	
Description	Interrupt port: TRIGGER event interrupt 0 request
Loop	–

DPLL_TE0_IRQ	
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

21.16.2 DPLL_ARU_INTERFACES

Loop	–
Condition	–
Format	–

ACT_D	
Description	Output of a time stamp, a position and a control signal for a calculated Action;SV_i=ACT_D[52:48] : ACB bits, directly written from the corresponding PMTR_D signals; ACT_D[47:24] is the calculated position value DPLL_-PSAC[n].PSAC for the Action in relation to CCM[0]_TBU_TS1 or CCM[0]_TBU_TS2 and ACT_D[23:0] is the time stamp value DPLL_TSAC[n].TSAC for the Action in relation to CCM[0]_TBU_TS0.Future time stamp with the re-solution as CCM[0]_TBU_TS0 input, additional position information and additional control bits;
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	4 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

ACT_RA	
Description	Action read address;address bits for selection of all NOAC Action channels.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	8 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

ACT_RR	
Description	read request of selected Action.the Action data is demanded from another module.
Loop	–

ACT_RR	
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

ACT_V	
Description	ACT_D value is available and valid;blocking read access. For a valid Action address: ACT_V reflects the shadow value of DPLL_ACT_STA.ACT_N[n](DPLL_ACT_STA.ACT_N[n] = 1 when new PMTR values are available and the shadow register is updated, when a calculation of the actual PMTR values was done);reset after reading of the ACT_D values.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

ARU_CA	
Description	ARU Channel address, for PMT data requests
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	8 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

PMTR_D	
Description	Position minus time request data, delivered by ARU on request for up to NOAC requests PMTR_RR;SV_i = PMTR_D[52:48] : ACB bits, directly written to the corresponding DPLL_ACB_[n] registersDPLL_PSA[n] = PMTR_D[47:24] : position value for ActionDPLL_DLA[n] = PMTR_D[23:0] time delay value for Action.Data values for calculation of actual Actions; the values are requested by AEN[n]=1 and DPLL_STATUS.CAIP1 =0 and DPLL_STATUS.CAIP2 =0 ; a served request is shown by PMTR_V which signals that valid PMTR data arrived and they are written immediately after that to the corresponding RAM regions and registers; The DPLL_DLA[n] values must have the same resolution as the CCM[0]_TBU_TS0 input.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR

PMTR_D	
Port Direction	IN
Port Range	4 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

PMTR_RAR	
Description	Read address of PMTR access.reflects DPLL_ID_PMTR_[n].ID_PMTR according to the selected channel address
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	8 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

PMTR_RR	
Description	Read request of PMTR access; suppressed for DPLL_STATUS.CAIP1=1 (see DPLL_STATUS register)read request of PMTR access; suppressed for DPLL_STATUS.CAIP2=1 (see DPLL_STATUS register).Reflects the value of the corresponding bit AEN[n]=1 for while the corresponding bit DPLL_STATUS.CAIP1=0reflects the value of the corresponding bit AEN[n]=1 while the corresponding bit DPLL_STATUS.CAIP2=0
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

PMTR_V	
Description	Signals a valid PMTR_D value, that means data is delivered on request.when valid: PMTR_D overwrites data in the DPLL_PSA[n] and DPLL_DLA[n] registers, also when the corresponding bit DPLL_ACT_STA.ACT_N[n]=1;
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

21.16.3 DPLL Ports

Loop	–
Condition	–
Format	–

T_VALID	
Description	The values of TRIGGER are valid. Announces the arrival of a new TRIGGER value
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TS_CLK	
Description	Time stamp clock. Used for generation of the time stamps; this clock is used to generate the SUB_INC1/SUB_INC2 pulses.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TRIGGER	
Description	Normal Signal for triggering DPLL by positions/values: TRIGGER[48:48] = TRIGGER_S, TRIGGER[47:24] = TRIGGER_FT, TRIGGER[23:0] = TRIGGER_TS DPLL_16191. One bit signal value (SV), 24 bits filter delay value info and 24 bits time stamp, filtered in different modes.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	52 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TRIGGER_TS	
Description	Time stamp in TRIGGER input port: TRIGGER_TS=TRIGGER[23:0]
Loop	–

TRIGGER_TS	
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TRIGGER_S	
Description	Signal level in TRIGGER input port: TRIGGER_S=TRIGGER[48:48]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TRIGGER_FT	
Description	Filter value in TRIGGER input port: TRIGGER_S=TRIGGER[47:24]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TDIR	
Description	Direction of TRIGGER input values (TDIR =0 means a forward direction and TDIR =1 means backward direction).Direction information from multiple sensors valid only for DPLL_CTRL_1.SMC=1 or DPLL_CTRL_1.IDDS=1.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET

TDIR	
Reset Value	–

S_VALID	
Description	The values of STATE are valid. Announces the arrival of a new STATE value
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

SUB_INC	
Description	Sub increment pulses
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

SUB_INC2C	
Description	Pulse output for time base unit 2 in compensation mode (can stop in automatic end mode). sub-position increment related to STATE input (when TRIGGER and STATE are used for 2 independent devices).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

SUB_INC2	
Description	Pulse output for STATE input filter(when TRIGGER and STATE are used for 2 independent devices). sub-position increment provided continuously
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC

SUB_INC2	
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

SUB_INC1	
Description	Pulse output for TRIGGER input filter. sub-position increment provided continuously.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

SUB_INC1C	
Description	Pulse output for time base unit 1 in compensation mode (can stop in automatic end mode). sub-position increment related to TRIGGER input
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

STA_T	
Description	Status of state machine TRIGGER.Output to MCS0. Signals accessible via uC interface as well (DPLL_STA).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

STA_S	
Description	Status of state machine STATE.Output to MCS0.Signals accessible via uC interface as well (DPLL_STA).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

STATE_TS	
Description	Time stamp in STATE input port: STATE_TS=STATE[23:0]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

STATE_S	
Description	Signal level in STATE input port: STATE_S=STATE[48:48]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

STATE	
Description	Assistance signal for synchronization STATE[48:48]= STATE_S;STATE[47:24]= STATE_FT;STATE[23:0]= STATE_-TS;Replacement of signal TRIGGER for emergency situations, or signal from an independent device; bits like above, corresponding
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK

STATE	
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

STATE_FT	
Description	Filter value in STATE input port: STATE_S=STATE[47:24]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

SDIR	
Description	Direction of STATE input values (SDIR =0 means forward direction and SDIR =1 means backward direction). Direction information from multiple sensors valid only for DPLL_CTRL_1.SMC=1.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

RESET_N	
Description	Asynchronous reset signal.Low active;After reset the DPLL is available only after performing the RAM reset procedures by the DPLL hardware.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

LOW_RES	
Description	low resolution of CCM[0]_TBU_TS0 selected;shows which of the 27 bits of CCM[0]_TBU_TS0 are connected to the DPLL.LOW_RES =0: CCM[0]_TBU_TS0(DPLL)= lower 24 Bits of CCM[0]_TBU_TS0(TBU); LOW_RES =1: CCM[0]_TBU_TS0(DPLL)= higher 24 Bits of CCM[0]_TBU_TS0(TBU); In the case LOW_RES =1 the bits DPLL_CTRL_1.TS0_HRT and/or DPLL_CTRL_1.TS0_HRS can be set

LOW_RES	
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DIR2	
Description	Direction information of SUB_INC2 (count forwards for DIR2 =0 and backwards for DIR2=1).Count direction of TBU_CH[2]_BASE; DIR2 changes always after the evaluation of the corresponding valid STATE slope and after incrementing/decrementing the address pointer.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

DIR1	
Description	Direction information of SUB_INC1 (count forwards for DIR1 =0 and backwards for DIR1 =1).Count direction of TBU_CH[1]_BASE; DIR1 changes always after the evaluation of the corresponding valid TRIGGER slope and after incrementing/decrementing the address pointer.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CNT_T2	
Description	Count TRIGGER.Output to MCS0.This reflects the count of active TRIGGER slopes (mod8).Signals accessible via uC interface as well (DPLL_STA).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0

CNT_T2	
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CNT_S	
Description	Count STATE.Output to MCS0.This reflects the count of active STATE slopes (mod8).Signals accessible via uC interface as well (DPLL_STA).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	2 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

IRQ	
Description	Interrupt request output
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIM_DPLL_TBU_TS0	
Description	27bit time stamp from TBU; needed to generate input time stamp for input events triggered by DPLL_SW_TRIG.TRIG_EVENT/DPLL_SW_TRIG.STATE_EVENT.When LOW_RES=1: IF DPLL_CTRL_1.TS0_HRT/DPLL_CTRL_1.TS0_HRS=0 use TIM_DPLL_TBU_TS0[26:3] as input time stamp. IF DPLL_CTRL_1.TS0_HRT/DPLL_CTRL_1.TS0_HRS=1 use TIM_DPLL_TBU_TS0[23:0] as input time stamp. When LOW_RES=0: Use TIM_DPLL_TBU_TS0[23:0] as input time stamp.
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	26 DOWNTO 0
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

INC_CNT1	
Description	Increment counter of pulse generator 1 (automatic end mode).Output to MCS0.Signals accessible via uC interface as well (DPLL_INC_CNT1).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

INC_CNT2	
Description	Increment counter of pulse generator 2 (automatic end mode).Output to MCS0.Signals accessible via uC interface as well (DPLL_INC_CNT2).
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[0]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

22 MCS to DPLL Interface of DPLL Module (MCS2DPLL)

22.1 Overview

To be modified in a suitable manner

The MCS2DPLL Interface has to perform the following tasks:

Indices as used inside this chapter are:

- NOAC= Number Of Action Channels
- n:={0, 1,..., 31} index of action channels
- m = loop index for registers (belonging to action channels)
- p = index of data out of memory according to profile of flywheel (*TRIGGER* and *STATE* channel)
- pm, pn, pt, pq, ps, pr = (indices used for profile pointers in the context of the signal processing equations. They are introduced formally in the chapters of use)

22.2 MCS to DPLL Interface Description

A reduced AEI interface is implemented in the DPLL, which can only be accessed by the MCS Bus Master interface in the same cluster. The purpose of this interface is to enable a faster interchange of data between the MCS and the DPLL, while enabling a certain control over the DPLL internal state machine.

22.2.1 Architecture and Organization

The implemented interface has an address width of 4 bits, while the size of the data interface is 24 bits.

The interface comprises 14 registers. The bit fields (**MCS2DPLL_DEB0.DATA** , **MCS2DPLL_DEB2.DATA** , **MCS2DPLL_DEB3.DATA** , **MCS2DPLL_DEB4.DATA** , **MCS2DPLL_DEB5.DATA** , **MCS2DPLL_DEB7.DATA**) are updated by the DPLL and read/written by MCS[0]. The bit fields (**MCS2DPLL_DEB1.DATA** , **MCS2DPLL_DEB6.DATA** , **MCS2DPLL_DEB8.DATA** , **MCS2DPLL_DEB9.DATA** , **MCS2DPLL_DEB10.DATA** , **MCS2DPLL_DEB11.DATA** , **MCS2DPLL_DEB12.DATA** , **MCS2DPLL_DEB15.DATA**) are not updated by the DPLL and written by MCS[0].

22.2.2 General Functionality

In order to have a better understanding of the implications when this interface is used, the following working concepts are informally defined here. They refer to the *STATE* engine operation when **DPLL_CTRL_11.STATE_EXT** is set.

Update of RAM: Operation which stores **DPLL_TSF_S[p].TSF_S** , **DPLL_DT_S_ACT.DT_S_ACT** and **DPLL_RDT_S_ACT.RDT_S_ACT** back to the RAM and reads the profile.
Calculation of sub-increments: Calculation of **DPLL_DT_S_ACT.DT_S_ACT** , **DPLL_RDT_S_ACT.RDT_S_ACT** , **DPLL_NMB_S.NMB_S** and *ADD_IN* .
Change of direction: Update of profile and its increment or decrement (only in the *STATE* processor).
Calculation of actions (PMT): Where the calculation of **DPLL_PSAC[n].PSAC** and **DPLL_TSAC[n].TSAC** is performed (only in state processor).

If **DPLL_CTRL_11.STATE_EXT** is not set , the DPLL will ignore the data written to this interface from the MCS. The DPLL will not update the interface either and a read done to this interface from the MCS can obtain out-of-date information.

If **DPLL_CTRL_11.STATE_EXT** is set , some modifications are done to the way that the DPLL module works when using the *STATE* engine.
Up to 128 *STATE* events can be handled.
RAM1c is not used anymore. Instead, the data needed to perform each of the already described operations is fetched from registers in this interface. The data that would have to be written back to RAM1c is also written to this interface.
In each of the procedures described above, the DPLL will enter in one or more stalled states, in which it will wait for one or more words to be written to **MCS2DPLL_DEB15.DATA** (STATUS_INFO)

Table 52 Correspondence Between DPLL_STA.STA_S Values and Their Unlocking Keywords

Operation	DPLL_STA.STA_S value	First keyword	Second keyword
Update of RAM	0x09	0xE	0x1
Calculation of sub-increments	0x10	0xD	0x2
Calculation of actions	0x70	0xC	0x3
Change of direction	0x04	0xB	0x4
Change of direction	0x06	0xB	0x4

The stalled *STATE* in the DPLL is freed by writing the upper keywords to **MCS2DPLL_DEB15.DATA** . Please note the requirements on DPLL level (described in chapter *22.3 " MCS to DPLL Registers Description"*) regarding the data on the interface that has to be written by the MCS program. If this data is incorrectly delivered or the *STATE* state machine is unlocked before delivery, the proper signal processing of the DPLL cannot be assured.

For the particular case of an update of RAM after a virtual increment, the data field **DPLL_TSF_S[p].TSF_S** is not calculated completely by the DPLL *STATE* processing unit. Instead, the values needed in order to fill this data field are provided (chapter *21.8.5.6 "Extend the time stamp values for STATE signal"* and chapter *21.8.5.7 "Extend the time stamp values for STATE signal for backward direction"*)

22.3 MCS to DPLL Registers Description

As already mentioned in section *22.2 "MCS to DPLL Interface Description"* , the following registers of the MCS2DPLL interface are exclusively accessible by the MCS instance of cluster 0 and cannot be accessed directly via CPU.

22.3.1 MCS2DPLL_DEB0

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB0
Address	$0x5600$
C–Name	

DATA	
Description	Data exchange buffer 0. Actual content depends on whether it is a Read or Write operation from MCS.
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Duration of the last increment **DPLL_DT_S_ACT.DT_S_ACT** (see chapter *21.8.5 "Update of RAM in Normal and Emergency Mode"*). This value is updated by the DPLL during the update of RAM (**DPLL_STA.STA_S** = 0x09) and is ready to be read when **DPLL_STA.STA_S** is modified to 0x0A.
- WRITE access from MCS:
- The DPLL expects **DPLL_DT_S[p–1].DT_S** (see chapter *21.7.3.5 "Calculate the error of last prediction"*) or **DPLL_DT_S[p+1].DT_S** (see chapter *21.7.5.2 "Calculate the error of the last prediction"*) during the increment prediction (**DPLL_STA.STA_S** = 0x10).

Note:

The data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO)

22.3.2 MCS2DPLL_DEB1

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB1
Address	$0x5604$
C-Name	

DATA	
Description	Data exchange buffer 1
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

READ access from MCS:
Reads as 0.
WRITE access from MCS:
The DPLL expects **DPLL_RDT_S[p-1].RDT_S** (chapter *21.7.3.4 "Calculate QDT_S_ACT"*) or **DPLL_RDT_S[p+1].RDT_S** (chapter *21.7.5.1 "Calculate QDT_S_ACT backwards"*) during the increment prediction (**DPLL_STA.STA_S** = 0x10) and **DPLL_RDT_S[pt-1].RDT_S** (chapter *21.8.3 "Action calculations for STATE signal forwards"*) or **DPLL_RDT_S[pt+1].RDT_S** (chapter *21.8.4 "Action calculations for STATE signal backwards "*) during the action calculation (**DPLL_STA.STA_S** = 0x70)

Note:

In both cases, the data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO).

22.3.3 MCS2DPLL_DEB2

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB2

Address	$0x5608$
C-Name	

DATA	
Description	Data exchange buffer 2. Actual content depends on whether it is a Read or Write operation from MCS.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- TS_Sx (chapter *21.8.5.5 "Update the time stamp values for STATE signal"*). Use to compute/update the time stamp values for *STATE* during the update of RAM (**DPLL_STA.STA_S** = 0x09)
- WRITE access from MCS:
- The DPLL expects **DPLL_RDT_S[p-pq-1].RDT_S** (chapter *21.7.3.5 "Calculate the error of last prediction"*) or **DPLL_RDT_S[p+pq+1].RDT_S** (chapter *21.7.5.2 "Calculate the error of the last prediction"*) during the increment prediction (**DPLL_STA.STA_S** = 0x10).

Note:
The data read from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO).

Note:
The data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO)

22.3.4 MCS2DPLL_DEB3

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB3
Address	$0x560C$
C-Name	

DATA	
Description	Data exchange buffer 3. Actual content depends on whether it is a Read or Write operation from MCS.
Loop	-
Bit Range	$[23:0]$

DATA	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- DT_S_ax (chapter *21.8.5.5 "Update the time stamp values for STATE signal"*). Use to compute/update the time stamp values for *STATE* during the update of RAM (**DPLL_STA.STA_S** = 0x09)
- WRITE access from MCS:
- The DPLL expects **DPLL_DT_S[p−pq].DT_S** (chapter *21.7.3.5 "Calculate the error of last prediction"*) or **DPLL_DT_S[p+pq].DT_S** (chapter *21.7.5.2 "Calculate the error of the last prediction"*) during the increment prediction (**DPLL_STA.STA_S** = 0x10).

Note:
The data read from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO).

22.3.5 MCS2DPLL_DEB4

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB4
Address	$0x5610$
C−Name	

DATA	
Description	Data exchange buffer 4. Actual content depends on whether it is a Read or Write operation from MCS.
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

DATA	
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- **DPLL_NUSC_EXT1.SYN_S_OLD** (chapter *21.8.5.5 "Update the time stamp values for STATE signal"*). Use to compute/update the time stamp values for *STATE* during the update of RAM (**DPLL_STA.STA_S** = 0x09)
- WRITE access from MCS:
- The DPLL expects **DPLL_RDT_S[p-pq].RDT_S** (chapter *21.7.3.7 "Calculate the current increment (nominal value)"*) or **DPLL_RDT_S[p+pq].RDT_S** (chapter *21.7.5.4 "Calculate the current increment value"*) during the increment prediction (**DPLL_STA.STA_S** = 0x10) and **DPLL_RDT_S[pt-pq].RDT_S** (chapter *21.8.3 "Action calculations for STATE signal forwards"*) or **DPLL_RDT_S[pt+pq].RDT_S** (chapter *21.8.4 "Action calculations for STATE signal backwards "*) during the action calculation (**DPLL_STA.STA_S** = 0x70)

Note:
The data read from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO).

Note:
The data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO)

22.3.6 MCS2DPLL_DEB5

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB5
Address	$0x5614$
C-Name	

DATA	
Description	Data exchange buffer 5. Actual content depends on whether it is a Read or Write operation from MCS.
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- M_DW (pm in chapter *21.8.3 "Action calculations for STATE signal forwards"* and chapter *21.8.4 "Action calculations for STATE signal backwards "*). Use to provide the proper time stamp field value during the action calculation (**DPLL_STA.STA_S** = 0x70)

- WRITE access from MCS:
- The DPLL expects **DPLL_DT_S[p-pq+1].DT_S** (chapter *21.7.3.7 "Calculate the current increment (nominal value)"*) or **DPLL_DT_S[p+pq--1].DT_S** (chapter *21.7.5.4 "Calculate the current increment value"*) during the increment prediction (**DPLL_STA.STA_S** = 0x10) and **DPLL_DT_S[pt-pq+1].DT_S** (chapter *21.8.3 "Action calculations for STATE signal forwards"*) or **DPLL_DT_S[pt+pq-1].DT_S** (chapter *21.8.4 "Action calculations for STATE signal backwards "*) during the action calculation (**DPLL_STA.STA_S** = 0x70)

Note:
The data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO).

22.3.7 MCS2DPLL_DEB6

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB6
Address	$0x5618$
C-Name	

DATA	
Description	Data exchange buffer 6
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Reads as 0.
- WRITE access from MCS:
- The DPLL expects **DPLL_ADT_S[p]** (with p = **DPLL_APS_EXT.APS_1C2**) during the update of RAM (**DPLL_STA.STA_S** = 0x09) and change of direction (**DPLL_STA.STA_S** = 0x04 and **DPLL_STA.STA_S** = 0x06)

Note:
In both cases, the current **DPLL_ADT_S[p]** (with p = **DPLL_APS_EXT.APS_1C2**) value should be stored in the register before unlocking the state machine the second time, i.e. between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO).

22.3.8 MCS2DPLL_DEB7

Description	MCS2DPLL exchange buffer register.
Loop	

Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB7
Address	$0x561C$
C-Name	

DATA	
Description	Data exchange buffer 7. Actual content depends on whether it is a Read or Write operation from MCS.
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Duration of the reciprocal of the last increment **DPLL_RDT_S_ACT.RDT_S_ACT** (chapter *21.8.5 "Update of RAM in Normal and Emergency Mode"*). This value is written by the DPLL during the update of RAM (**DPLL_STA.STA_S** = 0x09) and is ready to be read when **DPLL_STA.STA_S** is modified to 0x0A.
- WRITE access from MCS:
- The DPLL expects the reciprocal of the last increment **DPLL_RDT_S[p].RDT_S** (with p = **DPLL_APS_EXT.APS**) (chapter *21.8.5 "Update of RAM in Normal and Emergency Mode"*) before it is overwritten with **DPLL_RDT_S_ACT.RDT_S_ACT** during the update of RAM (**DPLL_STA.STA_S** = 0x09). For the action calculation, this value is needed as well as **DPLL_RDT_S[pt].RDT_S** in chapter *21.8.3 "Action calculations for STATE signal forwards"* and chapter *21.8.4 "Action calculations for STATE signal backwards "* .

Note:

During an update of RAM, perform the write before unlocking the state machine (**DPLL_STA.STA_S** = 0x09), i.e.: after the first write to **MCS2DPLL_DEB15** (STATUS_INFO) but before the second write. During the action calculation, the data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO).

22.3.9 MCS2DPLL_DEB8

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB8

Address	$0x5620$
C-Name	

DATA	
Description	Data exchange buffer 8
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Reads as 0.
- WRITE access from MCS:
- The DPLL expects **DPLL_TSF_S[p].TSF_S** (chapter *21.8.3 "Action calculations for STATE signal forwards"* and chapter *21.8.4 "Action calculations for STATE signal backwards "*) during the action calculation (**DPLL_STA.STA_S** = 0x70)

Note:
The data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO)

22.3.10 MCS2DPLL_DEB9

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB9
Address	$0x5624$
C-Name	

DATA	
Description	Data exchange buffer 9
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False

DATA	
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Reads as 0.
- WRITE access from MCS:
- The DPLL expects **DPLL_TSF_S[p-pn].TSF_S** (chapter *21.8.3 "Action calculations for STATE signal forwards"*) or **DPLL_TSF_S[p+pn].TSF_S** (chapter *21.8.4 "Action calculations for STATE signal backwards "*) during the action calculation (**DPLL_STA.STA_S** = 0x70)

Note:
The data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO)

22.3.11 MCS2DPLL_DEB10

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB10
Address	$0x5628$
C-Name	

DATA	
Description	Data exchange buffer 10
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Reads as 0.

- WRITE access from MCS:
- The DPLL expects **DPLL_TSF_S[p+pm-pn].TSF_S** (chapter *21.8.3 "Action calculations for STATE signal forwards"*) or **DPLL_TSF_S[p--pm+pn].TSF_S** (chapter *21.8.4 "Action calculations for STATE signal backwards "*) during the action calculation (**DPLL_STA.STA_S** = 0x70)

Note:
The data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO)

22.3.12 MCS2DPLL_DEB11

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB11
Address	$0x562C$
C-Name	

DATA	
Description	Data exchange buffer 11
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Reads as 0.
- WRITE access from MCS:
- The DPLL expects **DPLL_TSF_S[p+pm].TSF_S** (chapter *21.8.3 "Action calculations for STATE signal forwards"*) or **DPLL_TSF_S[p-pm].TSF_S** (chapter *21.8.4 "Action calculations for STATE signal backwards "*) during the action calculation (**DPLL_STA.STA_S** = 0x70)

Note:
The data write from MCS should be performed between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO)

22.3.13 MCS2DPLL_DEB12

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$

Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB12
Address	$0x5630$
C-Name	

DATA	
Description	Data exchange buffer 12
Loop	-
Bit Range	$[23:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Reads as 0.
- WRITE access from MCS:
- The DPLL expects the future adapt information **DPLL_ADT_S[p]** (with p = **DPLL_APS_EXT.APS_1C2** +1) (when in forwards) or **DPLL_ADT_S[p]** (with p = **DPLL_APS_EXT.APS_1C2** -1) (when in backwards) (chapter *21.7.3.5 "Calculate the error of last prediction"*) during the update of RAM (**DPLL_STA.STA_S** = 0x09) and change of direction (**DPLL_STA.STA_S** = 0x04 and **DPLL_STA.STA_S** = 0x06)

Note:

In both cases, the current **DPLL_ADT_S[p]** (with p = **DPLL_APS_EXT.APS_1C2** +1) or **DPLL_ADT_S[p]** (with p = **DPLL_APS_EXT.APS_1C2** -1) value should be stored in the register before unlocking the state machine the second time, i.e. between the two writes to **MCS2DPLL_DEB15** (STATUS_INFO).

22.3.14 MCS2DPLL_DEB15

Description	MCS2DPLL exchange buffer register.
Loop	
Condition	$\mathrm{NDPLL} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MCS2DPLL\_CLK\_ENABLE} == 1$

Interface: MCS[i]

Name	MCS2DPLL_DEB15
Address	$0x563C$
C-Name	

DATA	
Description	Data exchange buffer 15
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

- READ access from MCS:
- Reads as 0.
- WRITE access from MCS:
- Unlocks the DPLL *STATE* state machine. See chapter *22.2.2 "General Functionality"* .

23 Sensor Pattern Evaluation (SPE)

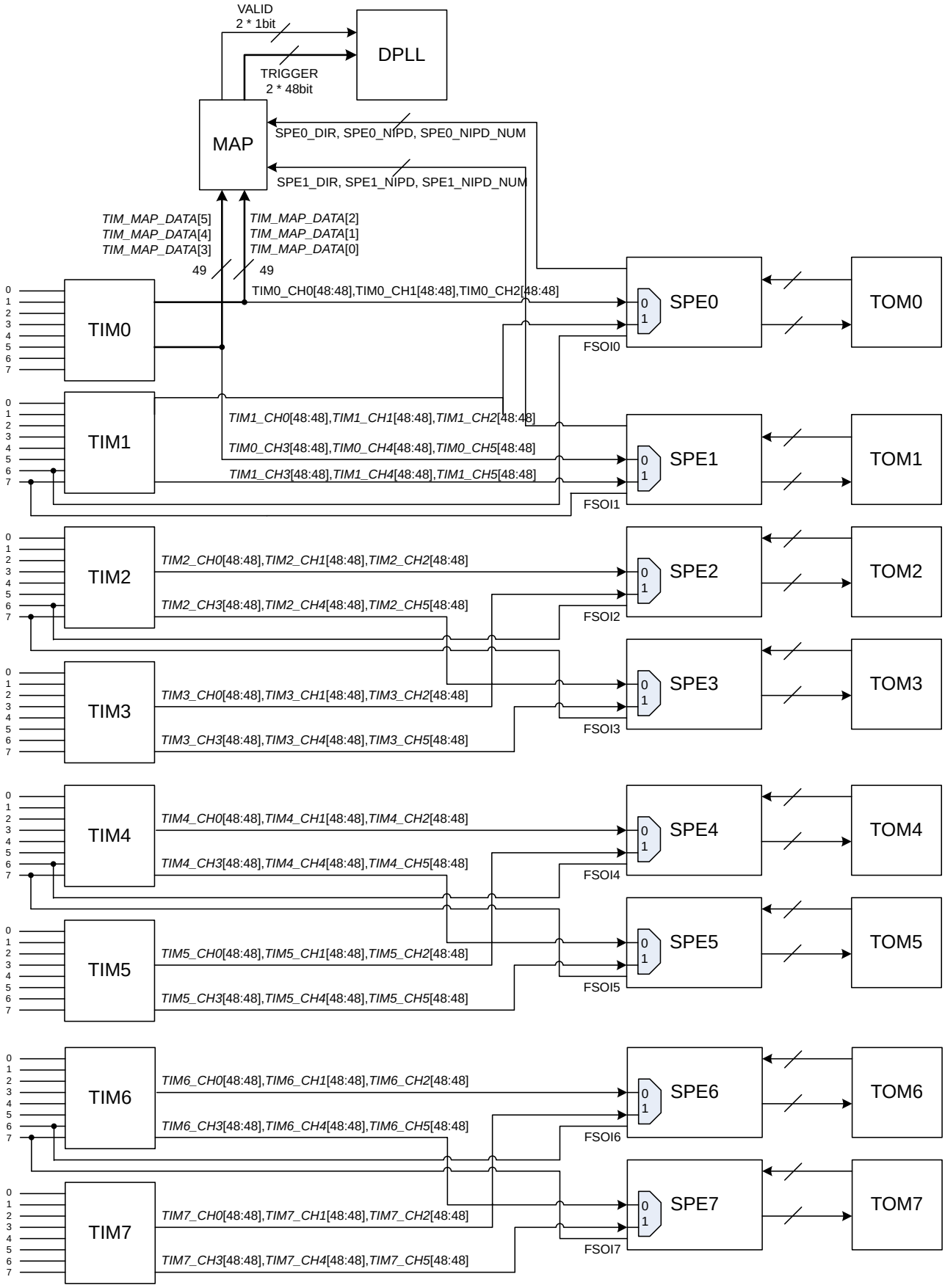
23.1 Overview

The Sensor Pattern Evaluation (SPE) submodule can be used to evaluate three hall sensor inputs and together with the TOM module to support the drive of BLDC engines. Thus, the input signals are filtered already in the connected TIM channels. In addition, the SPE submodule can be used as an input stage to the MAP submodule if the DPLL should be used to calculate the rotation speed of one or two electric engine(s). The integration of the SPE submodule into the overall GTM−IP architecture concept is shown in figure *157 "SPE Submodule Integration Concept into GTM−IP"* .

Indices and their range as used inside this chapter are:

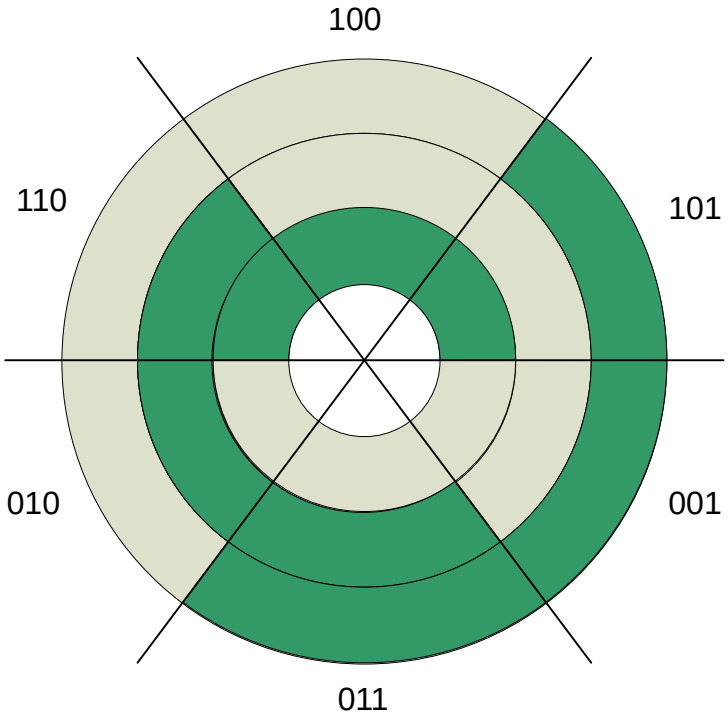
- i:={0, 1,..., NSPE−1} instance index of cluster / SPE module
- x, y, z channel index (x !=y!=z)
- t:={0, 1,..., 7} index of input pattern
- p:={0, 1,..., 7} index for pattern
- o:={0, 1,..., 7} index for output pattern

Figure 157 SPE Submodule Integration Concept into GTM-IP



The SPE submodule can determine a rotation direction out of the combined *TIM[i]_CH* [x:x], *TIM[i]_CH* [y:y] and *TIM[i]_CH* [z:z] signals. On this input signals a pattern match algorithm is applied to generate the *SPE[i]_DIR* signal on behalf of the temporal relation between these input patterns. A possible sample pattern of the three input signals is shown in figure *158 "SPE Sample Input Pattern"* . In general, the input pattern is programmable within the SPE submodule.

Figure 158 SPE Sample Input Pattern



In figure *158 "SPE Sample Input Pattern"* the input signals define the pattern from the input sensors which have a 50% high and 50% low phase. The pattern according to figure *158 "SPE Sample Input Pattern"* is as follows:

100 – 110 – 010 – 011 – 001 – 101 – 100

Where the first bit (smallest circle) represents *TIM[i]_CH* [x:x], the second bit represents *TIM[i]_CH* [y:y], and the third bit (greatest circle) represents *TIM[i]_CH* [z:z].

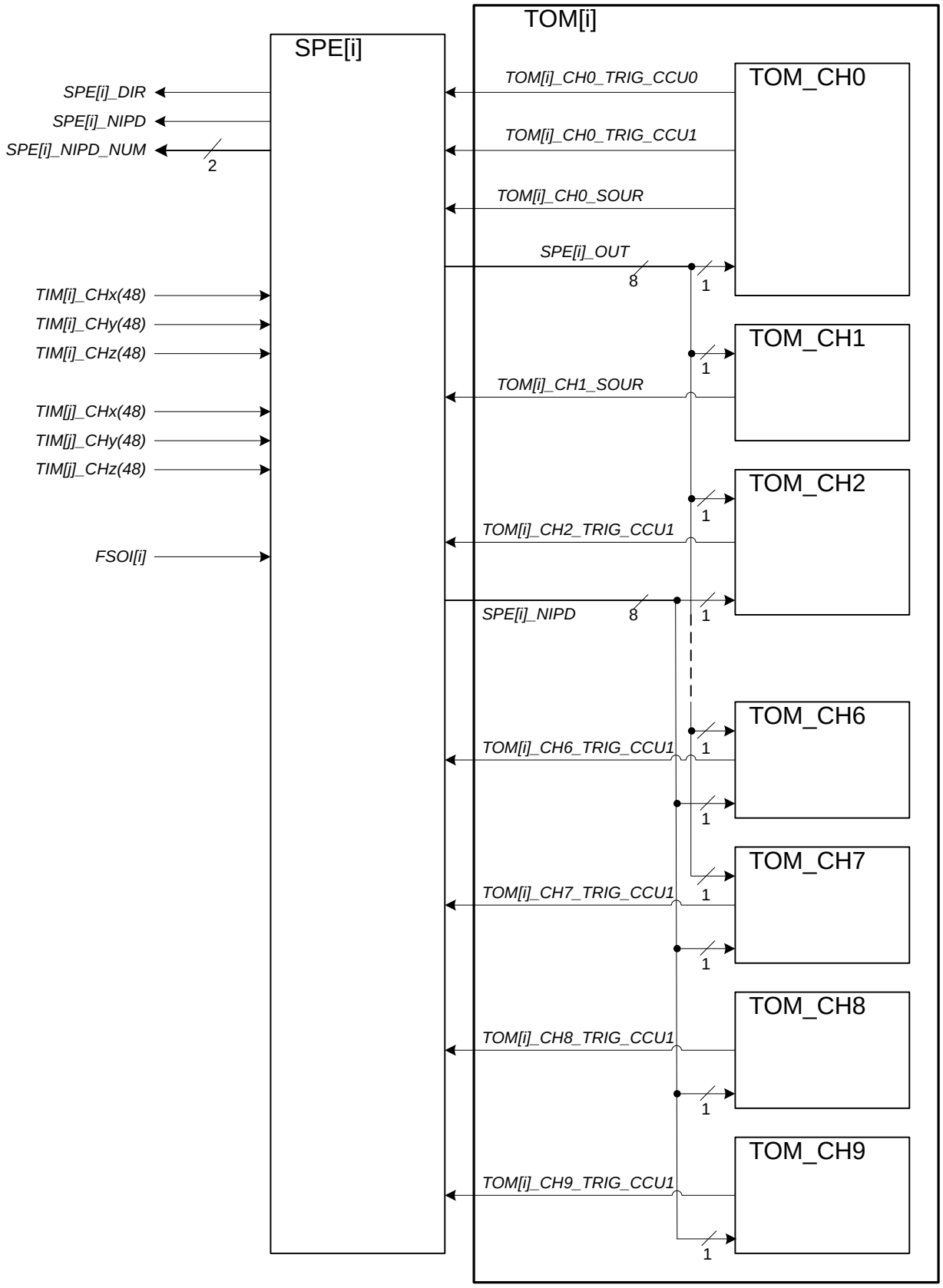
Note:

The SPE module expects that with every new pattern only one of the three input signals changes its value.

23.2 SPE Submodule Description

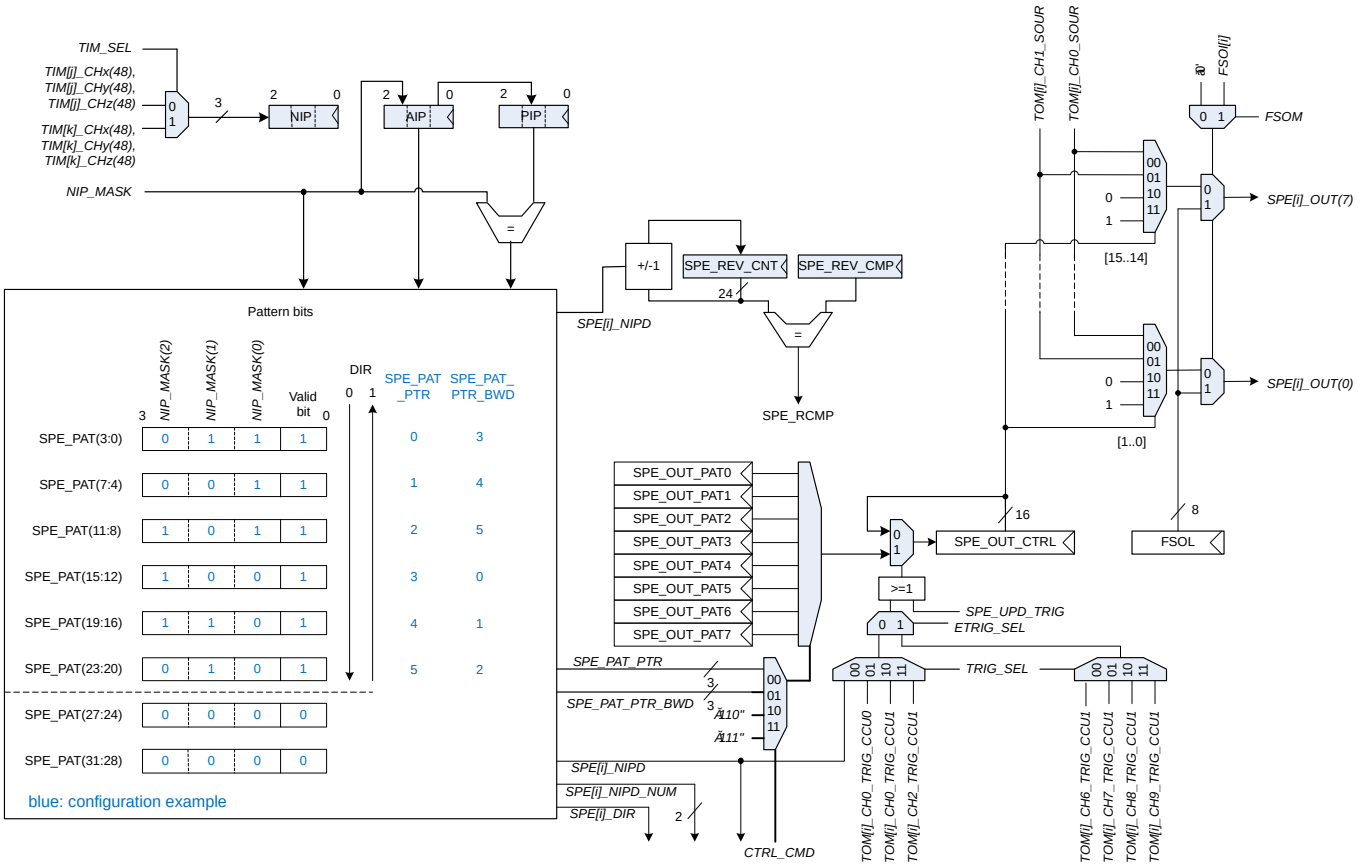
The SPE submodule can handle sensor pattern inputs. Every time if one of the input signals *TIM[i]_CH* [x:x], *TIM[i]_CH* [y:y] or *TIM[i]_CH* [z:z] changes its value, a sample of all three input signals is made. Derived from the sample of the three inputs the encoded rotation direction and the validity of the input pattern sequence is determined and signaled. When a valid input pattern is detected, the SPE submodule can control the outputs of a dedicated connected TOM submodule. This connection is shown in figure *159 "SPE to TOM Connections"* .

Figure 159 SPE to TOM Connections



The *TOM_CH0_TRIG_CCU0* , *TOM_CH[0]_TRIG_CCU1* and *TOM_CH[x]_SOUR* signal lines are used to evaluate the current state of the TOM outputs, whereas the *SPE[i]_OUT* output vector is used to control the TOM output depending on the new input pattern. The *SPE[i]_OUT* output vector is defined inside the SPE submodule in a pattern definition table **SPE[i]_OUT_PAT[p]** . The internal SPE submodule architecture is shown in figure *160 "SPE Submodule Architecture"* .

Figure 160 SPE Submodule Architecture

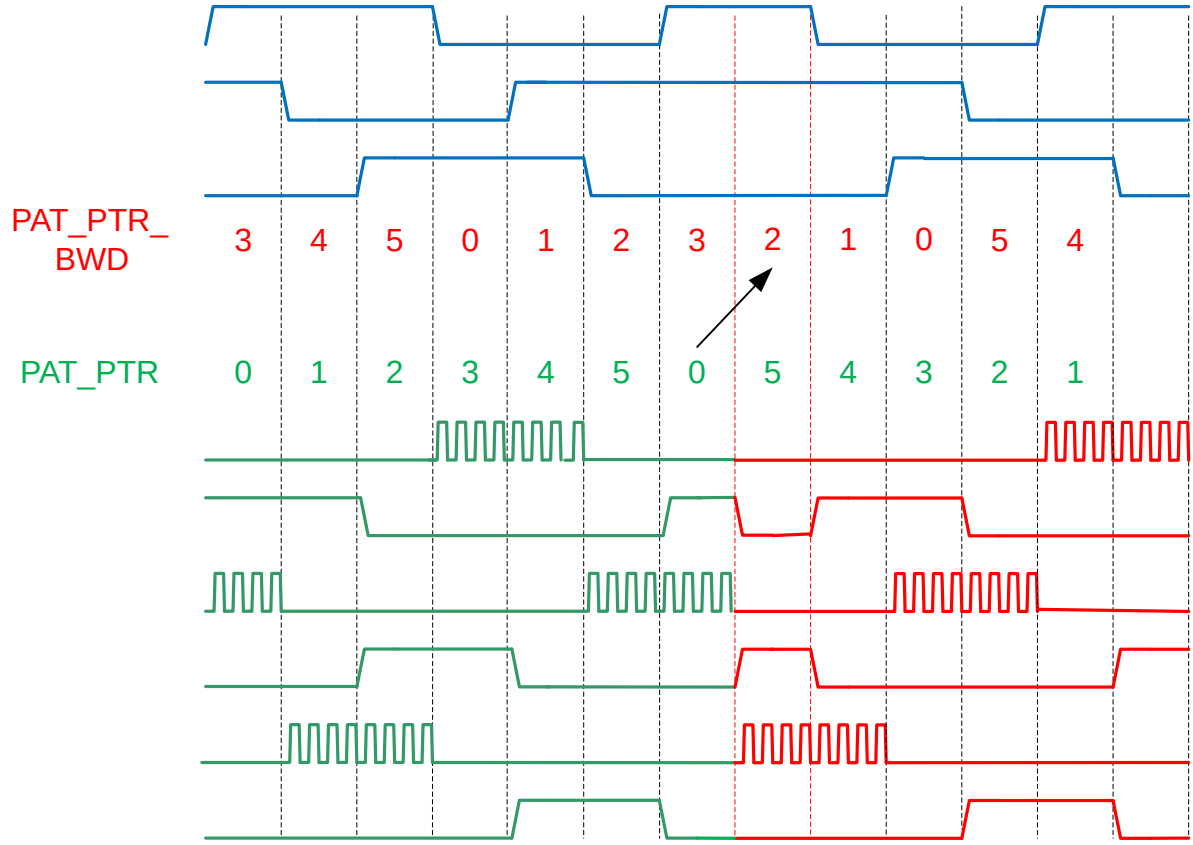


The **SPE[i]_PAT** register holds the valid input pattern for the three input patterns *TIM[i]_CH* [x:x], *TIM[i]_CH* [y:y] and *TIM[i]_CH* [z:z]. The input pattern is programmable. The valid bit shows if the programmed pattern is a valid one. Figure *160 "SPE Submodule Architecture"* shows the programming of the **SPE[i]_PAT** register for the input pattern defined in figure *158 "SPE Sample Input Pattern"* .

The rotation direction is determined by the order of the valid input pattern. This rotation direction defines if **SPE[i]_CTRL_STAT.SPE_-PAT_PTR** is incremented (*DIR* = 0) or decremented (*DIR* = 1). Whenever a valid input pattern is detected, the *SPE[i]_NIPD* signal is raised, the **SPE[i]_CTRL_STAT.SPE_PAT_PTR** is incremented/decremented and a new output control signal *SPE[i]_OUT* [x:x] is sent to the corresponding TOM submodule.

To command directly the forward or backward rotation the SPE provides with **SPE[i]_CTRL_STAT.SPE_PAT_PTR** and **SPE[i]_CTRL_STAT2.-SPE_PAT_PTR_BWD** two pointers to array **SPE[i]_OUT_PAT[p]** . Both can point to different values of **SPE[i]_OUT_PAT[p]** at the same point in time. **SPE[i]_CTRL_STAT.SPE_PAT_PTR** is intended to point to the pattern for forward commanding and **SPE[i]_CTRL_STAT2.SPE_PAT_-PTR_BWD** is intended to point to the pattern for backward commanding.
On startup, both pointers have to be configured to an initial value that corresponds to different direction depending on the start pattern of **SPE[i]_OUT_PAT[p]** .
With each valid new input pattern indicated by *SPE[i]_NIPD*, both pointers will be incremented or decremented according to the detected direction.
Switching from forward command to backward command can then be done by changing the selected pointer to **SPE[i]_OUT_PAT[p]** array, i.e. changing **SPE[i]_CMD.SPE_CTRL_CMD** from selecting **SPE[i]_CTRL_STAT.SPE_PAT_PTR** to selecting **SPE[i]_CTRL_STAT2.SPE_PAT_P-TR_BWD** or vice versa.
The intended behavior is depicted in the following figure:

Figure 161 SPE Forward – Backward Commanding



With command **SPE[i]_CMD.SPE_CTRL_CMD** = 0b10 or 0b11 a dedicated configurable output pattern configured to the pattern **SPE[i]_OUT_PAT[p].SPE_OUT_PAT[6]** or **SPE[i]_OUT_PAT[p].SPE_OUT_PAT[7]** can be commanded to the outputs.
An example is the introduction of a SW dead time if switching from pointer **SPE[i]_CTRL_STAT.SPE_PAT_PTR** to **SPE[i]_CTRL_STAT2.SPE_PAT_PTR_BWD** or vice versa. E.g. if in **SPE[i]_OUT_PAT[6]** the value 0b10 is programmed for each output (i.e. set *SPE[i]_OUT* [n:n] to 0), this can be used as an intermediate step to introduce this 'all off' when switching between **SPE[i]_CTRL_STAT.SPE_PAT_PTR** to **SPE[i]_CTRL_STAT2.SPE_PAT_PTR_BWD** or vice versa.

Selectable by **SPE[i]_CTRL_STAT.TRIG_SEL** and **SPE[i]_CTRL_STAT.ETRIG_SEL** the CCU1 trigger of either the TOM channel z (z∈{2, 6, 7, 8, 9}) can be used together with the SPE module to trigger a delayed update of the **SPE[i]_OUT_CTRL.SPE_OUT_CTRL[o]** register after new input pattern detected by SPE (signaled by *SPE[i]_NIPD*).
To do this, the TOM channel z (z∈{2, 6, 7, 8, 9}) has to be configured to work in one-shot mode (set bit **TOM[i]_CH[z]_CTRL.OSM**). The SPE trigger of this channel has to be enabled too (set description of bit **TOM[i]_CH[z]_CTRL.SPEM** and bit **TOM[i]_CH[z]_CTRL.SPE_TRIG**). The SPE module has to be configured to update **SPE[i]_OUT_CTRL.SPE_OUT_CTRL[o]** on *TOM_CH[z]_TRIG_CCU1* (set in **SPE[i]_CTRL_STAT** bits **SPE[i]_CTRL_STAT.TRIG_SEL** to 0b11). Then, on new input detected by SPE, the signal *SPE[i]_NIPD* triggers the start of the TOM channel z to generate one PWM period by resetting **TOM[i]_CH[x]_CN0.CN0** to 0. On second PWM edge triggered by CCU1 of TOM channel z, the signal *TOM_CH[z]_TRIG_CCU1* triggers the update of **SPE[i]_OUT_CTRL.SPE_OUT_CTRL[o]** . Be aware, that setting TOM CCU1 compare register **TOM[i]_CH[x]_CM1.CM1** to a value of '0' will not trigger an update of the **SPE[i]_OUT_CTRL.SPE_OUT_CTRL[o]** register.

The update of **SPE[i]_OUT_CTRL** with the content of one of the **SPE[i]_OUT_PAT[p].SPE_OUT_PAT[o]** register field can be triggered at any time by writing a 1 to bit **SPE[i]_CMD.SPE_UPD_TRIG** .

The regular trigger for update of **SPE[i]_OUT_CTRL** (commutation trigger) is selected by **SPE[i]_CTRL_STAT.TRIG_SEL** and **SPE[i]_CTRL_STAT.ETRIG_SEL** .

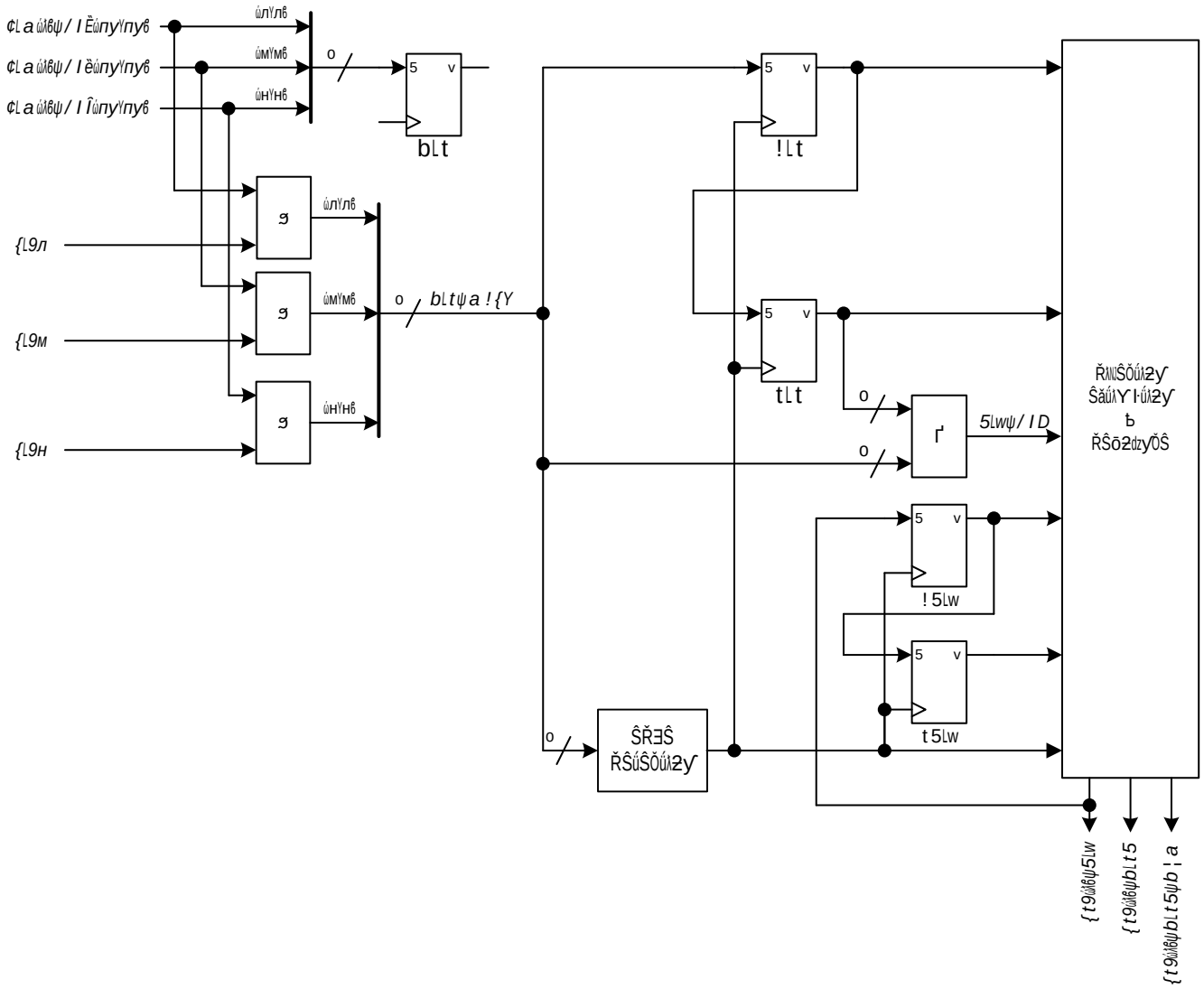
According to figure *160 "SPE Submodule Architecture"* , the two input patterns 0b000 and 0b111 are not allowed combinations and will end in a *SPE[i]_PERR* interrupt. These two patterns can be used to determine a sensor input error. A *SPE[i]_PERR* interrupt will also be raised, if the input patterns occur in a wrong order, e.g. if the pattern 0b010 does not follow the pattern 0b110 or 0b011.

The **SPE[i]_CTRL_STAT** bit field is implemented inside the **SPE[i]_CTRL_STAT** register, where the input pattern history is stored by the SPE submodule. The CPU can determine a broken sensor when the *SPE[i]_PERR* interrupt occurs by analyzing the bit pattern readable via bit field **SPE[i]_CTRL_STAT.NIP** inside the **SPE[i]_CTRL_STAT** register. The input pattern in the **SPE[i]_CTRL_STAT** register is updated whenever a valid edge is detected on one of the input lines *TIM[i]_CH* [x:x], *TIM[i]_CH* [y:y] or *TIM[i]_CH* [z:z]. The pattern bit fields are then shifted. The input pattern history generation inside the **SPE[i]_CTRL_STAT** register is shown in figure *162 "SPE[i]_CTRL_STAT Register Representation"* .

In addition to the sensor pattern evaluation, the SPE module also provides the feature of fast shutoff for all TOM channels controlled by the SPE module. The feature is enabled by setting bit **SPE[i]_CTRL_STAT.FSOM** . The fast shutoff level itself is defined in the bit field **SPE[i]_CTRL_STAT.FSOL** of register **SPE[i]_CTRL_STAT** . The TIM input used to trigger the fast shutoff is either TIM channel 6 or TIM channel

7 depending on the TIM instance connected to the SPE module. For details of connections please refer to figure *157 "SPE Submodule Integration Concept into GTM-IP"* .

Figure 162 SPE[i]_CTRL_STAT Register Representation



The CPU can disable one of the three input signals, e.g. when a broken input sensor was detected, by disabling the input with the three input enable bits **SPE[i]_CTRL_STAT.SIE[k]** .

Whenever at least one of the input signals *TIM[i]_CH* [x:x], *TIM[i]_CH* [y:y] or *TIM[i]_CH* [z:z] changes, the SPE submodule stores the new bit pattern in an internal register bit field **SPE[i]_CTRL_STAT.NIP** (New Input Pattern). If the current input pattern in **SPE[i]_CTRL_STAT.NIP** is the same as in the previous input pattern (**SPE[i]_CTRL_STAT.PIP**) the direction of the engine is changed, the *SPE[i]_DCHG* interrupt is raised, the direction change is stored internally and the pattern in the **SPE[i]_CTRL_STAT.PIP** bit field is filled with the **SPE[i]_CTRL_STAT.AIP** bit field and the **SPE[i]_CTRL_STAT.AIP** bit field is filled with the **SPE[i]_CTRL_STAT.NIP** bit field. The **SPE[i]_CTRL_STAT.ADIR** bit inside the **SPE[i]_CTRL_STAT** register is toggled and the *SPE[i]_DIR* signal is changed.

If the SPE encounters that with the next input pattern, new input pattern **SPE[i]_CTRL_STAT.NIP** again detects the change in direction, the input signal is categorized as bouncing and the bouncing input signal interrupt *SPE[i]_BIS* is raised.

Immediately after update of register bit field **SPE[i]_CTRL_STAT.NIP** , when the new detected input pattern doesn't match the **SPE[i]_CTRL_STAT.PIP** pattern (i.e. no direction change was detected), the SPE shifts the value of register **SPE[i]_CTRL_STAT.AIP** to register **SPE[i]_CTRL_STAT.PIP** and the value of register **SPE[i]_CTRL_STAT.NIP** to register **SPE[i]_CTRL_STAT.AIP** . The *SPE[i]_NIPD* interrupt is raised.

The number of the channel that has been changed and thus leads to the new input pattern is encoded in the signal *SPE[i]_NIPD_NUM* .

If a sensor error was detected, the CPU has to define the pattern in the **SPE[i]_CTRL_STAT** register, which input line comes from the broken sensor. The faulty signal line has to be masked by the CPU and the SPE submodule determines the rotation direction on behalf of the two remaining *TIM[i]_CH* [x:x] input lines.

The pattern history can be determined by the CPU by reading the two bit fields **SPE[i]_CTRL_STAT.AIP** and **SPE[i]_CTRL_STAT.PIP** of the **SPE[i]_CTRL_STAT** register. The **SPE[i]_CTRL_STAT.AIP** register field holds the current input pattern at *TIM[i]_CH* [x:x], *TIM[i]_CH* [y:y] and *TIM[i]_CH* [z:z] and the **SPE[i]_CTRL_STAT.PIP** holds the previous pattern.

After reset the register fields **SPE[i]_CTRL_STAT.NIP** , **SPE[i]_CTRL_STAT.AIP** and **SPE[i]_CTRL_STAT.PIP** as well as the registers **SPE[i]_CTRL_STAT.SPE_PAT_PTR** and **SPE[i]_OUT_CTRL** will not contain valid startup values which would allow correct behavior after enabling SPE and detecting the first input patterns.
Thus, it is necessary to initialize these registers to correct values.
To do this, before enabling the SPE, the bit field **SPE[i]_CTRL_STAT.NIP** can be read and depending on this value the initialization values for the register fields **SPE[i]_CTRL_STAT.AIP** , **SPE[i]_CTRL_STAT.PIP** , **SPE[i]_CTRL_STAT.SPE_PAT_PTR** and **SPE[i]_OUT_CTRL** can be determined.

23.2.1 SPE Revolution Detection

The SPE submodule is able to detect and count the number of valid input patterns detected at the specified input ports. This is done with a 24 bit revolution counter **SPE[i]_REV_CNT.REV_CNT** . The counter is incremented by a value of one (1) when a new valid input pattern indicating forward direction is detected. The counter is decremented by a value of one (1) when a new valid input pattern indicating backward direction is detected.

In addition, there exists a 24 bit **SPE[i]_REV_CMP** register. The user can initialize this register with a compare value, where an interrupt *SPE[i]_RCMP* is raised, when the revolution counter equals the compare value either in forward or backward direction.

Both registers may be written by software at any time.

23.3 SPE Configuration Registers Description

23.3.1 SPE[i]_CTRL_STAT

Description	SPE[i] Control Status Register
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{SPE}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_CTRL_STAT
Address	$0x20000 * i + 0x4C00$
C-Name	GTM.CLS[i].SPE.CTRL_STAT

Interface: MCS[i]

Name	SPE[i]_CTRL_STAT
Address	$0x4C00$
C-Name	

EN	
Description	SPE Submodule enable
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

EN	
RW-Coding	0 : SPE disabled. 1 : SPE enabled.

SIE[k]	
Description	SPE Input [k] enable for TIM[i]_CH[x:x], TIM[i]_CH[y:y], TIM[i]_CH[z:z].
Loop	$k = \{n : 0 <= n <= 2\}$
Bit Range	$[k+1 : k+1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : SPE Input is disabled. 1 : SPE Input is enabled.
	Note: When the input is disabled, a 0 signal is sampled for this input. However, the bit field **SPE[i]_CTRL_STAT.NIP** shows the true value of the input signal.

TRIG_SEL	
Description	Select trigger input signal
Loop	–
Bit Range	$[5 : 4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	SPE[i]_CTRL_STAT.ETRIG_SEL == 0 0b00 : *SPE[i]_NIPD* selected 0b01 : *TOM_CH0_TRIG_CCU0* selected 0b10 : *TOM_CH[0]_TRIG_CCU1* selected 0b11 : *TOM_CH[2]_TRIG_CCU1* selected
RW-Coding	SPE[i]_CTRL_STAT.ETRIG_SEL == 1 0b00 : *TOM_CH[6]_TRIG_CCU1* selected 0b01 : *TOM_CH[7]_TRIG_CCU1* selected 0b10 : *TOM_CH[8]_TRIG_CCU1* selected 0b11 : *TOM_CH[9]_TRIG_CCU1* selected

TRIG_SEL	
	Note: In case **SPE[i]_CTRL_STAT.ETRIG_SEL** = 1, according to the selected *TOM_CH[x]_TRIG_CCU1* signal, the configuration bits **TOM[i]_CH[x]_CTRL.SPE_TRIG** and **TOM[i]_CH[x]_CTRL.OSM** have to be set in the same TOM channel x to enable the trigger signal generation in one-shot mode.

TIM_SEL	
Description	Select TIM input signal
Loop	–
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	i == 0 0 : *TIM[0]_CH* [2:0] 1 : *TIM[1]_CH* [2:0]
RW-Coding	i == 1 0 : *TIM[0]_CH* [5:3] 1 : *TIM[1]_CH* [5:3]
RW-Coding	i == 2 0 : *TIM[2]_CH* [2:0] 1 : *TIM[3]_CH* [2:0]
RW-Coding	i == 3 0 : *TIM[2]_CH* [5:3] 1 : *TIM[3]_CH* [5:3]
RW-Coding	i == 4 0 : *TIM[4]_CH* [2:0] 1 : *TIM[5]_CH* [2:0]
RW-Coding	i == 5 0 : *TIM[4]_CH* [5:3] 1 : *TIM[5]_CH* [5:3]

FSOM	
Description	Fast Shutoff Mode
Loop	–
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

FSOM	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Fast Shutoff mode disabled 1 : Fast Shutoff mode enabled

SPE_PAT_PTR	
Description	Pattern selector for TOM output signals.
Loop	–
Bit Range	[10 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b000 : **SPE[i]_OUT_PAT[0]** selected 0b001 : **SPE[i]_OUT_PAT[1]** selected 0b010 : **SPE[i]_OUT_PAT[2]** selected 0b011 : **SPE[i]_OUT_PAT[3]** selected 0b100 : **SPE[i]_OUT_PAT[4]** selected 0b101 : **SPE[i]_OUT_PAT[5]** selected 0b110 : **SPE[i]_OUT_PAT[6]** selected 0b111 : **SPE[i]_OUT_PAT[7]** selected
	Index into the **SPE[i]_OUT_PAT[p]** register table. Each register **SPE[i]_OUT_PAT[p]** is fixed assigned to one bit field **SPE[i]_PAT.IP[t]_PAT** . Thus, the pointer **SPE[i]_CTRL_STAT.SPE_PAT_PTR** represents an index to the selected **SPE[i]_OUT_PAT[p]** register as well as the actual detected input pattern **SPE[i]_PAT.IP[t]_PAT** .

AIP	
Description	Input pattern that was detected by a regular input pattern change.
Loop	–
Bit Range	[14 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

ADIR	
Description	Rotation direction. Will be reflected in the signal SPE[i]_DIR.
Loop	–
Bit Range	[15 : 15]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Rotation direction is 0 according to **SPE[i]_PAT** register. 1 : Rotation direction is 1 according to **SPE[i]_PAT** register.

PIP	
Description	Previous input pattern that was detected by a regular input pattern change.
Loop	–
Bit Range	[18 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

PDIR	
Description	Previous rotation direction
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Rotation direction is 0 according to **SPE[i]_PAT** register. 1 : Rotation direction is 1 according to **SPE[i]_PAT** register.

NIP	
Description	New input pattern that was detected.
Loop	–
Bit Range	[22 : 20]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: This bit field mirrors the new input pattern. SPE internal functionality is triggered on each change of this bit field.

ETRIG_SEL	
Description	Extended trigger selection of signal TRIG_SEL
Loop	–
Bit Range	[23 : 23]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Selection of *TRIG_SEL* signal (For explanation see note at **SPE[i]_CTRL_STAT.TRIG_SEL**) 1 : Selection of *TRIG_SEL* signal (For explanation see note at **SPE[i]_CTRL_STAT.TRIG_SEL**)

FSOL	
Description	Fast Shutoff Level for TOM[i] channel 0 to 7
Loop	–
Bit Range	[31 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

FSOL	
Reset group	HW_RESET: async GTM_RESET: async

Note:
SPE[i]_CTRL_STAT holds the current and the previous states of the hall pattern and can be written by the CPU for initialization purposes after reset before enabling SPE. Any CPU write to **SPE[i]_CTRL_STAT** has precedence over internal updates of this register and will also disable the generation of all SPE interrupts during this access.

23.3.2 SPE[i]_PAT

Description	SPE[i] Input Pattern Definition Register
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{SPE}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_PAT
Address	$0x20000 * i + 0x4C04$
C-Name	GTM.CLS[i].SPE.PAT

Interface: MCS[i]

Name	SPE[i]_PAT
Address	$0x4C04$
C-Name	

IP[t]_VAL	
Description	Input pattern [t] is a valid pattern.
Loop	$t = \{n : 0 <= n <= 7\}$
Bit Range	$[4 * t : 4 * t]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Pattern invalid. 1 : Pattern valid.

IP[t]_PAT	
Description	Input pattern [t]

IP[t]_PAT	
Loop	$t = \{n : 0 <= n <= 7\}$
Bit Range	$[4 * t + 3 : 4 * t + 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Bit field defines the first input pattern of the SPE input signals. Bit 1 defines the *TIM[i]_CH* [x:x] input signal. Bit 2 defines the *TIM[i]_CH* [y:y] input signal. Bit 3 defines the *TIM[i]_CH* [z:z] input signal.

Note:
Only the first block of valid input patterns defines the commutation. All input pattern following the first marked invalid input pattern are ignored.

23.3.3 SPE[i]_OUT_PAT[p]

Description	SPE[i] Output Definition Register
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}; p = \{n : 0 <= n <= 7\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{SPE}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_OUT_PAT0
Address	$0x20000 * i + 0x4C08$
C–Name	GTM.CLS[i].SPE.OUT_PAT0
Name	SPE[i]_OUT_PAT1
Address	$0x20000 * i + 0x4C0C$
C–Name	GTM.CLS[i].SPE.OUT_PAT1
Name	SPE[i]_OUT_PAT2
Address	$0x20000 * i + 0x4C10$
C–Name	GTM.CLS[i].SPE.OUT_PAT2
Name	SPE[i]_OUT_PAT3
Address	$0x20000 * i + 0x4C14$
C–Name	GTM.CLS[i].SPE.OUT_PAT3
Name	SPE[i]_OUT_PAT4
Address	$0x20000 * i + 0x4C18$

C-Name	GTM.CLS[i].SPE.OUT_PAT4
Name	SPE[i]_OUT_PAT5
Address	$0x20000 * i + 0x4C1C$
C-Name	GTM.CLS[i].SPE.OUT_PAT5
Name	SPE[i]_OUT_PAT6
Address	$0x20000 * i + 0x4C20$
C-Name	GTM.CLS[i].SPE.OUT_PAT6
Name	SPE[i]_OUT_PAT7
Address	$0x20000 * i + 0x4C24$
C-Name	GTM.CLS[i].SPE.OUT_PAT7

Interface: MCS[i]

Name	SPE[i]_OUT_PAT0
Address	$0x4C08$
C-Name	
Name	SPE[i]_OUT_PAT1
Address	$0x4C0C$
C-Name	
Name	SPE[i]_OUT_PAT2
Address	$0x4C10$
C-Name	
Name	SPE[i]_OUT_PAT3
Address	$0x4C14$
C-Name	
Name	SPE[i]_OUT_PAT4
Address	$0x4C18$
C-Name	
Name	SPE[i]_OUT_PAT5
Address	$0x4C1C$
C-Name	
Name	SPE[i]_OUT_PAT6
Address	$0x4C20$
C-Name	
Name	SPE[i]_OUT_PAT7
Address	$0x4C24$
C-Name	

SPE_OUT_PAT[o]	
Description	SPE output control value for TOM channel0 to TOM channel7
Loop	$o = \{n : 0 <= n <= 7\}$

SPE_OUT_PAT[o]	
Bit Range	$[2 * o + 1 : 2 * o]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Set *SPE[i]_OUT* [o:o] to *TOM_CH[0]_SOUR* 0b01 : Set *SPE[i]_OUT* [o:o] to *TOM_CH[1]_SOUR* 0b10 : Set *SPE[i]_OUT* [o:o] to 0 0b11 : Set *SPE[i]_OUT* [o:o] to 1
	SPE[i]_OUT_PAT[p].SPE_OUT_PAT[o] defines output select signal of TOM instance i channel n.

Note:

Register bit field **SPE[i]_OUT_PAT[p].SPE_OUT_PAT[o]** defines the output selection for TOM instance i channel 0 to TOM instance i channel 7 depending on the input pattern **SPE[i]_PAT.IP[t]_PAT** .

23.3.4 SPE[i]_OUT_CTRL

Description	SPE[i] Output Control Register
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{SPE}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_OUT_CTRL
Address	$0x20000 * i + 0x4C28$
C-Name	GTM.CLS[i].SPE.OUT_CTRL

Interface: MCS[i]

Name	SPE[i]_OUT_CTRL
Address	$0x4C28$
C-Name	

SPE_OUT_CTRL[o]	
Description	SPE output control value for TOM channel0 to TOM channel7
Loop	$o = \{n : 0 <= n <= 7\}$
Bit Range	$[2 * o + 1 : 2 * o]$
Access Type	RW

SPE_OUT_CTRL[o]	
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Set *SPE[i]_OUT* [o:o] to *TOM_CH[0]_SOUR* 0b01 : Set *SPE[i]_OUT* [o:o] to *TOM_CH[1]_SOUR* 0b10 : Set *SPE[i]_OUT* [o:o] to 0 0b11 : Set *SPE[i]_OUT* [o:o] to 1
	SPE[i]_OUT_CTRL.SPE_OUT_CTRL[o] defines output select signal of TOM instance i channel n. **Note:** Current output control selection for *SPE[i]_OUT* [7:0].

23.3.5 SPE[i]_REV_CNT

Description	SPE[i] Input Revolution Counter
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{SPE}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_REV_CNT
Address	$0x20000 * i + 0x4C40$
C-Name	GTM.CLS[i].SPE.REV_CNT

Interface: MCS[i]

Name	SPE[i]_REV_CNT
Address	$0x4C40$
C-Name	

REV_CNT	
Description	Input signal revolution counter
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–

REV_CNT	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	The counter is running if SPE module is enabled (bit **SPE[i]_CTRL_STAT.EN**). **SPE[i]_REV_CNT.REV_CNT** is incrementing if **SPE[i]_CTRL_STAT.SPE_PAT_PTR** is incrementing **SPE[i]_REV_CNT.REV_CNT** is decrementing if **SPE[i]_CTRL_STAT.SPE_PAT_PTR** is decrementing

23.3.6 SPE[i]_REV_CMP

Description	SPE[i] Revolution Counter Compare Value
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{SPE}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_REV_CMP
Address	$0x20000 * i + 0x4C44$
C-Name	GTM.CLS[i].SPE.REV_CMP

Interface: MCS[i]

Name	SPE[i]_REV_CMP
Address	$0x4C44$
C-Name	

REV_CMP	
Description	Input signal revolution counter compare value
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

REV_CMP	
	The interrupt *SPE[i]_RCMP* is raised when the **SPE[i]_REV_CNT.REV_CNT** value equals the **SPE[i]_REV_CMP.REV_CMP** register field. **Note:** *SPE[i]_RCMP* is only raised if an incrementation or decrementation of **SPE[i]_REV_CNT** is applied, due to an input signal change. Any update of **SPE[i]_REV_CNT** or **SPE[i]_REV_CMP** via AEI does not raise a *SPE[i]_RCMP* interrupt.

23.3.7 SPE[i]_IRQ_NOTIFY

Description	SPE[i] Interrupt Notification Register
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{SPE}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_IRQ_NOTIFY
Address	$0x20000 * i + 0x4C2C$
C-Name	GTM.CLS[i].SPE.IRQ_NOTIFY

Interface: MCS[i]

Name	SPE[i]_IRQ_NOTIFY
Address	$0x4C2C$
C-Name	

SPE_NIPD	
Description	New input pattern interrupt occurred
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

SPE_NIPD	
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

SPE_DCHG	
Description	SPE_DIR bit changed on behalf of new input pattern.
Loop	–
Bit Range	$[1:1]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

SPE_PERR	
Description	Wrong or invalid pattern detected at input.
Loop	–
Bit Range	$[2:2]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

SPE_BIS	
Description	Bouncing input signal detected
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	True

SPE_BIS	
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

SPE_RCMP	
Description	SPE revolution counter match event
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

23.3.8 SPE[i]_IRQ_EN

Description	SPE[i] Interrupt Enable Register
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{SPE}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_IRQ_EN
Address	$0x20000 * i + 0x4C30$
C-Name	GTM.CLS[i].SPE.IRQ_EN

Interface: MCS[i]

Name	SPE[i]_IRQ_EN

Address	$0x4C30$
C-Name	

SPE_NIPD_IRQ_EN	
Description	SPE_NIPD_IRQ interrupt enable
Loop	-
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP. 1 : Enable interrupt, interrupt is visible outside GTM-IP.

SPE_DCHG_IRQ_EN	
Description	SPE_DCHG_IRQ interrupt enable
Loop	-
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt, interrupt is not visible outside GTM-IP. 1 : Enable interrupt, interrupt is visible outside GTM-IP.

SPE_PERR_IRQ_EN	
Description	SPE_PERR_IRQ interrupt enable
Loop	-
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

SPE_PERR_IRQ_EN	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable interrupt, interrupt is not visible outside GTM–IP. 1 : Enable interrupt, interrupt is visible outside GTM–IP.

SPE_BIS_IRQ_EN	
Description	SPE_BIS_IRQ interrupt enable
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable interrupt, interrupt is not visible outside GTM–IP. 1 : Enable interrupt, interrupt is visible outside GTM–IP.

SPE_RCMP_IRQ_EN	
Description	SPE_RCMP_IRQ interrupt enable
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable interrupt, interrupt is not visible outside GTM–IP. 1 : Enable interrupt, interrupt is visible outside GTM–IP.

23.3.9 SPE[i]_IRQ_FORCINT

Description	SPE[i] Interrupt Generation By Software
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	
Storage Type	REGISTER

Clock Active Cond	SPE[i]_CLK_ENABLE == 1

Interface: CPU

Name	SPE[i]_IRQ_FORCINT
Address	$0x20000 * i + 0x4C34$
C-Name	GTM.CLS[i].SPE.IRQ_FORCINT

Interface: MCS[i]

Name	SPE[i]_IRQ_FORCINT
Address	$0x4C34$
C-Name	

TRG_SPE_NIPD	
Description	Trigger SPE[i]_IRQ_NOTIFY.SPE_NIPD by software
Loop	-
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 4, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **SPE[i]_IRQ_NOTIFY.SPE_NIPD**
	Note: This bit is cleared automatically after interrupt is released **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_SPE_DCHG	
Description	Trigger SPE[i]_IRQ_NOTIFY.SPE_DCHG by software.
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 4, 0) != 0)

TRG_SPE_DCHG	
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **SPE[i]_IRQ_NOTIFY.SPE_DCHG**
	Note: This bit is cleared automatically after interrupt is released **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_SPE_PERR	
Description	Trigger SPE[i]_IRQ_NOTIFY.SPE_PERR by software.
Loop	-
Bit Range	$[2:2]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 4, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **SPE[i]_IRQ_NOTIFY.SPE_PERR**
	Note: This bit is cleared automatically after interrupt is released **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_SPE_BIS	
Description	Trigger SPE[i]_IRQ_NOTIFY.SPE_BIS by software.
Loop	-
Bit Range	$[3:3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 4, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status

TRG_SPE_BIS	
W-Coding	0 : No action 1 : Force setting of **SPE[i]_IRQ_NOTIFY.SPE_BIS**
	Note: This bit is cleared automatically after interrupt is released **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

TRG_SPE_RCMP	
Description	Trigger SPE[i]_IRQ_NOTIFY.SPE_RCMP by software.
Loop	-
Bit Range	$[4:4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[i]_AEI_ARB_WDATA, 4, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **SPE[i]_IRQ_NOTIFY.SPE_RCMP**
	Note: This bit is cleared automatically after interrupt is released **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT**

23.3.10 SPE[i]_IRQ_MODE

Description	SPE[i] Interrupt Mode Configuration Register
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{SPE}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_IRQ_MODE
Address	$0x20000 * i + 0x4C38$
C-Name	GTM.CLS[i].SPE.IRQ_MODE

Interface: MCS[i]

Name	SPE[i]_IRQ_MODE

Address	$0x4C38$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	-
Bit Range	$[1:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

23.3.11 SPE[i]_EIRQ_EN

Description	SPE[i] Error Interrupt Enable Register
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	SPE[i]_CLK_ENABLE $== 1$

Interface: CPU

Name	SPE[i]_EIRQ_EN
Address	$0x20000 * i + 0x4C3C$
C-Name	GTM.CLS[i].SPE.EIRQ_EN

Interface: MCS[i]

Name	SPE[i]_EIRQ_EN
Address	$0x4C3C$
C-Name	

SPE_NIPD_EIRQ_EN	
Description	SPE_NIPD_EIRQ interrupt enable
Loop	-

SPE_NIPD_EIRQ_EN	
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP. 1 : Enable error interrupt, error interrupt is visible outside GTM–IP.

SPE_DCHG_EIRQ_EN	
Description	SPE_DCHG_EIRQ error interrupt enable
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP. 1 : Enable error interrupt, error interrupt is visible outside GTM–IP.

SPE_PERR_EIRQ_EN	
Description	SPE_PERR_EIRQ error interrupt enable
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP. 1 : Enable error interrupt, error interrupt is visible outside GTM–IP.

SPE_BIS_EIRQ_EN	
Description	SPE[i]_BIS_EIRQ error interrupt enable
Loop	–
Bit Range	$[3:3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP. 1 : Enable error interrupt, error interrupt is visible outside GTM–IP.

SPE_RCMP_EIRQ_EN	
Description	SPE[i]_RCMP_EIRQ error interrupt enable
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Disable error interrupt, error interrupt is not visible outside GTM–IP. 1 : Enable error interrupt, error interrupt is visible outside GTM–IP.

23.3.12 SPE[i]_CTRL_STAT2

Description	SPE[i] Control Status Register 2
Loop	$i = \{n : 0 <= n <= \text{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\text{SPE}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_CTRL_STAT2
Address	$0x20000 * i + 0x4C48$
C–Name	GTM.CLS[i].SPE.CTRL_STAT2

Interface: MCS[i]

Name	SPE[i]_CTRL_STAT2
Address	$0x4C48$
C-Name	

SPE_PAT_PTR_BWD	
Description	Pattern selector for TOM output signals in case of SPE[i]_CMD.SPE_CTRL_CMD = 0b01 (e.g. backward direction).
Loop	-
Bit Range	$[10 : 8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	▶ Index into the **SPE[i]_OUT_PAT[p]** register table in case of **SPE[i]_CMD.SPE_CTRL_CMD** = 0b01 which may be used for backward direction. ▶ Each register **SPE[i]_OUT_PAT[p]** is fixed assigned to one bit field **SPE[i]_PAT.IP[t]_PAT** of register **SPE[i]_PAT** . Thus, the pointer **SPE[i]_CTRL_STAT2.SPE_PAT_PTR_BWD** represents an index to the selected **SPE[i]_OUT_PAT[p]** register as well as the detected input pattern **SPE[i]_PAT.IP[t]_PAT** . ▶ The index pointer **SPE[i]_CTRL_STAT2.SPE_PAT_PTR_BWD** is used if **SPE[i]_CMD.SPE_CTRL_CMD** = 0b01. ▶ The index pointer **SPE[i]_CTRL_STAT.SPE_PAT_PTR** is used if **SPE[i]_CMD.SPE_CTRL_CMD** = 0b00 (by default). ▶ 0b000 = **SPE[i]_OUT_PAT[0]** selected

23.3.13 SPE[i]_CMD

Description	SPE[i] Command Register
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{SPE}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	SPE[i]_CMD
Address	$0x20000 * i + 0x4C4C$
C-Name	GTM.CLS[i].SPE.CMD

Interface: MCS[i]

Name	SPE[i]_CMD
Address	$0x4C4C$

C–Name	

SPE_CTRL_CMD	
Description	SPE control command
Loop	–
Bit Range	$[1:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0b00 : Use pi: **SPE[i]_CTRL_STAT.SPE_PAT_PTR** as an index pointer to select **SPE[i]_OUT_PAT[p]** 0b01 : Use pi: **SPE[i]_CTRL_STAT2.SPE_PAT_PTR_BWD** as an index pointer to select **SPE[i]_OUT_PAT[p]** 0b10 : Select **SPE[i]_OUT_PAT[6]** 0b11 : Select **SPE[i]_OUT_PAT[7]**
	Note: On switch between 0b00 and 0b01 the direction flag **SPE[i]_CTRL_STAT.ADIR** will be independently set according to the input pattern **SPE[i]_CTRL_STAT.NIP** and **SPE[i]_CTRL_STAT.AIP** .

SPE_UPD_TRIG	
Description	SPE updater trigger
Loop	–
Bit Range	$[16:16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
W–Coding	1 : Trigger update of **SPE[i]_OUT_CTRL.SPE_OUT_CTRL[o]** with register selected by **SPE[i]_CMD.SPE_CTRL_CMD** multiplexer. 0 : Do nothing.
	Note: This bit is automatically reset to 0.

23.4 SPE Signal Description

23.4.1 SPE Signals

Loop	–

Condition	–
Format	–

SPE[i]_BIS_EIRQ	
Description	SPE Bouncing input signal error interrupt.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Signal Type	ARRAY[NSPE]
Assignment	–

SPE[i]_RCMP_EIRQ	
Description	SPE Revolution counter compare error interrupt.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Signal Type	ARRAY[NSPE]
Assignment	–

TIM[i]_CH	
Description	TIM instance i filtered output F_OUT .
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Signal Type	ARRAY[NTIM]
Assignment	–

TRIG_SEL	
Description	Trigger select signal.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

23.4.2 SPE Interrupt Signals

Loop	–
Condition	–
Format	–

SPE[i]_DCHG	
Description	SPE Rotation direction change detected on behalf of input pattern.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Signal Type	ARRAY[NSPE]
Assignment	–

SPE[i]_NIPD	
Description	SPE New valid input pattern detected.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Signal Type	ARRAY[NSPE]
Assignment	–

SPE[i]_RCMP	
Description	SPE Revolution counter compare value reached.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Signal Type	ARRAY[NSPE]
Assignment	–

SPE[i]_PERR	
Description	SPE Invalid input pattern detected.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Signal Type	ARRAY[NSPE]
Assignment	–

SPE[i]_BIS	
Description	SPE Bouncing input signal detected at input.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Signal Type	ARRAY[NSPE]
Assignment	–

23.5 SPE Port Description

23.5.1 SPE Ports

Loop	–
Condition	–
Format	–

SPE[i]_DIR	
Description	Direction Output to MAP.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	–
Port Range	–
Operational Clock	CLS[i]_CLK

SPE[i]_DIR	
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

SPE[i]_NIPD_NUM	
Description	Index of changed TIM input to MAP.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	–
Port Range	2 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

SPE[i]_OUT	
Description	SPE output signal to TOM.
Loop	$i = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	–
Port Range	–
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

24 Interrupt Concentrator Module (ICM)

24.1 Overview

The Interrupt Concentrator Module (ICM) is used to group (bundle) the GTM-IP interrupt lines of the individual submodules into interrupt groups. By bundling, a smaller number of interrupt lines is visible at the outside of the GTM-IP.

The individual interrupts of the GTM-IP submodules and channels have to be enabled or disabled inside the submodules and channels.

The feed-through architecture of bundled interrupt lines is used for the submodules AEI, ARU, BRC, CMP, SPE, PSM, TIM, DPLL, TOM, ATOM, TIO and MCS.

To determine the detailed interrupt source the microcontroller has to read the submodule/channel interrupt notification register NOTIFY and serve the channel individual interrupt.

Note: The interrupts are only visible inside the ICM and in consequence outside of the GTM-IP, when the interrupt is enabled inside the submodules themselves.

Indices and their range as used inside this chapter are:

- j:={0, 1,..., NModule-1} instance index of module NModule:={NTOM, NATOM, NSPE, NMCS, NPSM, NTIO}
- x = channel index in a module (TOM / ATOM / SPE / MCS / PSM / TIO)
- g = index for dedicated register groups

24.2 Bundling

The GTM-IP submodule individual interrupt sources are connected to the ICM. There, the individual interrupt lines are either fed through and signaled to the outside world or bundled a second time and signaled to the outside world.
The ICM interrupt bundling is described in the following sections:

24.2.1 GTM Infrastructure Interrupt Bundling

The first interrupt group contains interrupts of the infrastructure and safety components of the GTM. This interrupt group therefore includes interrupt lines coming from the AEI, ARU, BRC, PSM, SPE and CMP submodules. In this interrupt group each individual channel of the submodules has its own interrupt line to the outside world.

Thus, the active interrupt line can be used by the CPU to determine the GTM-IP submodule channel that raised the interrupt. The interrupts are also represented in the **ICM_IRQG_0** register. This register is typically not read by the CPU, but it is readable.

In addition, the interrupt line status for the 8 channels of each FIFO is shown in the **ICM_IRQG_PSM_0_CI** register. Typically, the interrupt source determined by the corresponding interrupt line and the **ICM_IRQG_PSM_0_CI** register is typically not read out by the CPU, but is readable.

The interrupt line status for each SPE is shown in the **ICM_IRQG_SPE_CI** register. Typically, the interrupt source determined by the corresponding interrupt line and the **ICM_IRQG_SPE_CI** register is typically not read out by the CPU, but is readable.

24.2.2 DPLL Interrupt Bundling

The DPLL interrupt group handles the interrupts coming from the DPLL submodule of the GTM-IP. Each of the individual DPLL interrupt lines has its own dedicated interrupt line to the outside world. The interrupts are additionally identified in the **ICM_IRQG_1** interrupt group register. This register is typically not read out by the CPU, but it is readable.

24.2.3 TIM Interrupt Bundling

Inside this group, submodules which handle GTM-IP input signals are treated. This is the case for the TIM[j] submodules. Each TIM submodule channel is able to generate 6 individual interrupts if enabled inside the TIM channel. These six interrupts are bundled into one interrupt per TIM channel connected to the ICM.

The ICM performs no further bundling. Thus for the GTM-IP, 8 interrupt lines per TIM module are provided. The channel responsible for the interrupt can be determined by the raised interrupt line.

In addition, the registers **ICM_IRQG_2** and **ICM_IRQG_3** are mirrors for the TIM submodule channel interrupts and typically not read out by the CPU, but they are readable.

24.2.4 TIO Interrupt Bundling

Inside this group, submodules which handle GTM-IP input signals are treated. This is the case for the TIO[j]_G[g] submodules. Each TIO submodule channel is able to generate 6 individual interrupts if enabled inside the TIO channel. These six interrupts are bundled into one interrupt per TIO channel connected to the ICM.

The ICM performs no further bundling. Thus, for the GTM-IP 8 interrupt lines per TIO_CH[g] module are provided. The channel responsible for the interrupt can be determined by the raised interrupt line.

24.2.5 MCS Interrupt Bundling

For complex signal output generation, the MCS submodules are used inside the GTM-IP. Each of these MCS submodules can have 8 channels with one interrupt line. This interrupt line is connected to the ICM submodule and is fed through directly to the outside world.

The interrupt line status for the first 8 channels of each MCS is shown in the **ICM_IRQG_4** and **ICM_IRQG_5** register. The interrupt line status for all used channels of each MCS are shown in the **ICM_IRQG_MCS[j]_CI** register. Typically, the interrupt source is determined by the corresponding interrupt line, **ICM_IRQG_4** / **ICM_IRQG_5** and **ICM_IRQG_MCS[j]_CI** registers are not read out by the CPU, but they are readable.

24.2.6 TOM and ATOM Interrupt Bundling

For the TOM and ATOM submodules, the interrupts are bundled within the ICM submodule a second time to reduce external interrupt lines. The interrupts are ORed in a manner that one GTM-IP external interrupt line represents two adjacent TOM or ATOM channel interrupts. For TOM[j] and ATOM[j] the bundling is as shown in table *53 "ATOM interrupt bundling within ICM"* .

Table 53 ATOM interrupt bundling within ICM

ATOM[j] input IRQs	ATOM-output IRQs (OR-ed)
ATOM[j]_CH0_IRQ or *ATOM[j]_CH1_IRQ*	*GTM_ATOM[j]_IRQ* [0:0]
ATOM[j]_CH2_IRQ or *ATOM[j]_CH3_IRQ*	*GTM_ATOM[j]_IRQ* [1:1]
ATOM[j]_CH4_IRQ or *ATOM[j]_CH5_IRQ*	*GTM_ATOM[j]_IRQ* [2:2]
ATOM[j]_CH6_IRQ or *ATOM[j]_CH7_IRQ*	*GTM_ATOM[j]_IRQ* [3:3]

Table 54 TOM interrupt bundling within ICM

TOM[j]-input IRQs	TOM-output IRQs (OR-ed)
TOM[j]_CH0_IRQ or *TOM[j]_CH1_IRQ*	*GTM_TOM[j]_IRQ* [0:0]
TOM[j]_CH2_IRQ or *TOM[j]_CH3_IRQ*	*GTM_TOM[j]_IRQ* [1:1]
TOM[j]_CH4_IRQ or *TOM[j]_CH5_IRQ*	*GTM_TOM[j]_IRQ* [2:2]
TOM[j]_CH6_IRQ or *TOM[j]_CH7_IRQ*	*GTM_TOM[j]_IRQ* [3:3]
TOM[j]_CH8_IRQ or *TOM[j]_CH9_IRQ*	*GTM_TOM[j]_IRQ* [4:4]
TOM[j]_CH10_IRQ or *TOM[j]_CH11_IRQ*	*GTM_TOM[j]_IRQ* [5:5]
TOM[j]_CH12_IRQ or *TOM[j]_CH13_IRQ*	*GTM_TOM[j]_IRQ* [6:6]
TOM[j]_CH14_IRQ or *TOM[j]_CH15_IRQ*	*GTM_TOM[j]_IRQ* [7:7]

The interrupts coming from the TOM[j] submodules are registered in the **ICM_IRQG_TOM_[g]_CI** register. Two TOMs are always bundled in one ICM register for e.g., TOM0 and TOM1 are bundled in **ICM_IRQG_TOM_[g]_CI** . To identify the TOM submodule channel in which the interrupt occurred, the CPU has to read out the **ICM_IRQG_TOM_[g]_CI** register first before it goes to the TOM submodule channel itself.

The **ICM_IRQG_TOM_[g]_CI** register bits are cleared automatically when their corresponding interrupt in the submodule channels is cleared.

The interrupts coming from the ATOM[j] submodules are registered in the **ICM_IRQG_ATOM_[g]_CI** register. Four ATOMs are bundled in one ICM register for e.g., ATOM0, ATOM1, ATOM2 and ATOM3 are bundled in **ICM_IRQG_ATOM_[g]_CI** . To identify the ATOM submodule channel in which the interrupt occurred, the CPU has to read out the **ICM_IRQG_ATOM_[g]_CI** register first before it goes to the ATOM submodule channel itself.

The **ICM_IRQG_ATOM_[g]_CI** register bits are cleared automatically when their corresponding interrupt in the submodule channels is cleared.

The interrupt line status of two 16 channel TOMs is shown in each **ICM_IRQG_TOM_[g]_CI** register. TOM0 and TOM1 are bundled in **ICM_IRQG_TOM_[0]_CI** . The interrupt source is determined by the corresponding interrupt line and the **ICM_IRQG_TOM_[g]_CI** registers are typically not read out by the CPU, but they are readable.

The interrupt line status of four 8 channel ATOMs is shown in each **ICM_IRQG_ATOM_[g]_CI** register, ATOM0, ATOM1, ATOM2 and ATOM3 are bundled in **ICM_IRQG_ATOM_[0]_CI** . Typically, the interrupt source is determined by the corresponding interrupt line and the **ICM_IRQG_ATOM_[g]_CI** registers are typically not read out by the CPU, but they are readable.

24.2.7 Module Error Interrupt Bundling

The Module Error Interrupt group handles the error interrupts coming from the BRC, FIFO, TIM, MCS, SPE, CMP, and DPLL submodules of the GTM-IP. The Module Error interrupts are additionally identified in the **ICM_IRQG_MEI** error interrupt group register. This register is typically not read out by the CPU, but it is readable.

In addition, the error interrupt line status for each SPE is shown in the **ICM_IRQG_SPE_CEI** register. Typically, the error interrupt source is determined by the corresponding interrupt line and the **ICM_IRQG_SPE_CEI** register are typically not read out by the CPU, but they are readable.

24.2.8 FIFO Channel Error Interrupt Bundling

The FIFO Channel Error Interrupt group handles the error interrupts coming from the FIFO channel of the GTM-IP. The FIFO Channel Error interrupts are additionally identified in the **ICM_IRQG_CEI0** error interrupt group register. This register is typically not read out by the CPU, but it is readable.

The **ICM_IRQG_CEI0** register bits are cleared automatically, when their corresponding error interrupt in the submodule channel is cleared.

In addition, the error interrupt line status for 8 channels of each FIFO are shown in the **ICM_IRQG_PSM_0_CEI** register. Typically, the error interrupt source is determined by the corresponding interrupt line and the **ICM_IRQG_PSM_0_CEI** register are typically not read out by the CPU, but they are readable.

24.2.9 TIM Channel Error Interrupt Bundling

The TIM Channel Error Interrupt group handles the error interrupts coming from the TIM channel of the GTM-IP. The TIM Channel Error interrupts are additionally identified for the submodules TIM0, TIM1, TIM2 and TIM3 in the **ICM_IRQG_CEI1** error interrupt group register and for the submodules TIM4, TIM5 and TIM6 in the **ICM_IRQG_CEI2** error interrupt group register. These registers are typically not read out by the CPU, but they are readable.

The **ICM_IRQG_CEI1** and **ICM_IRQG_CEI2** register bits are cleared automatically, when their corresponding error interrupt in the submodule channel is cleared.

24.2.10 MCS Channel Error Interrupt Bundling

The MCS Channel Error Interrupt group handles the error interrupts coming from the MCS-channel of the GTM-IP. All used 8 MCS Channel Error interrupts are additionally identified for each submodules MCS[j] in the **ICM_IRQG_MCS[j]_CEI** error interrupt group register. The first 8 MCS Channel Error interrupts are additionally identified for the submodules MCS0, MCS1, MCS2 and MCS3 in the **ICM_IRQG_CEI3** error interrupt group register and for the submodules MCS4, MCS5, MCS6 and MCS7 in the **ICM_IRQG_CEI4** error interrupt group register. These registers are typically not read out by the CPU, but they are readable.

The **ICM_IRQG_MCS[j]_CEI** , **ICM_IRQG_CEI3** and **ICM_IRQG_CEI4** register bits are cleared automatically, when their corresponding error interrupt in the submodule channel is cleared.

24.2.11 Error Interrupt Cluster Bundling

The Error Interrupt lines of up to 4 clusters are bundled in each **ICM_IRQG_CLS_[g]_MEI** .
Actually each cluster collects one EIRQ of one TIM, MCS, SPE and FIFO.
These registers are typically not read out by the CPU, but they are readable.

24.3 ICM Toplevel Interrupt Signals

In the ICM module each top level interrupt signal is pipelined with a single stage register, which operates on the GTM-IP clock *CLK* .

The above table shows the GTM-IP interrupt lines that are visible outside of the GTM-IP.

24.4 ICM MCS Interrupt Signals

In the ICM module the MCS interrupt signals can be pipelined with a single stage register, they operate on the corresponding cluster clock *CLS[i]_CLK* depending on the chosen cluster clock divider.

Table 55 ICM MCS Interrupt Signals Table

Signal	Description	Established connection
MCS_TIM[j]_IRQ [x:x]	TIM[j] Shared interrupts (x∈{0, 1,..., 7})	Cluster internal connection TIM[j] to MCS[j], no register stage present, source and target module operate on same cls_clk of instance j frequency
MCS_TOM[j]_IRQ [x:x]	TOM[j] Shared interrupts for x (x∈{0, 1, ..., 7}), further OR bundling of interrupts applied *MCS_TOM[j]_IRQ* [x:x] = *GTM_TOM[j]_IRQ* [2*x:2*x] OR *GTM_TOM[j]_IRQ* [(2*x+1):(2*x+1)]	Cluster internal connection TOM[j] to MCS[j], no register stage present, source and target module operate on same cls_clk of instance j frequency
MCS_ATOM[j]_IRQ [x:x]	ATOM[j] Shared interrupts for x (x∈{0, 1,..., 7})	Cluster internal connection ATOM[j] to MCS[j], no register stage present, source and target module operate on same cls_clk of instance j frequency
MCS_TIMATOM[j]_IRQ [x:x]	TOM[j+1], ATOM[j+1], TIM[j+1] Shared interrupts for x (x∈{0, 1,..., 7}), further OR bundling of interrupts applied *MCS_TIMATOM[j]_IRQ* [x:x] = *GTM_TOM[j+1]_IRQ* [2*x:2*x] OR *GTM_TOM[j+1]_IRQ* [2*x+1:2*x+1] OR *GTM_ATOM[j+1]_IRQ* [x:x] OR *GTM_TIM[j+1]_IRQ* [x:x]	Cluster external connection TIM[j+1], ATOM[j+1], TOM[j+1] to MCS[j], pipeline register stage present, source and target module can operate on different cls_clk of instance j frequencies. Depending on clock divider values latency of 0,1 cls_clk of instance j clock cycles can occur.
MCS_TIO[j]_G[g]_IRQ [x:x]	TIO[j] Shared interrupts for g, x (g∈{0, 1, 2} x∈{0, 1,..., 7})	Cluster internal connection TIO[j] to MCS[j], no register stage present, source and target module operate on same cls_clk of instance j frequency
MCS_NMCS[j]_IRQ [x:x]	MCS[j+1] Interrupt for channel x (x∈{0, 1,..., 7})	Cluster external connection MCS[j+1] to MCS[j], pipeline register stage present, source and target module can operate on different cls_clk of instance j frequencies. Depending on clock divider values latency of 0,1 cls_clk of instance j clock cycles can occur.
MCS_MCS[0]_IRQ [x:x]	MCS[0] Interrupt for channel x (x∈{0, 1,..., 7})	Cluster external connection MCS[0] to MCS[j], pipeline register stage present, source and target module can operate on different cls_clk of instance j frequencies. Depending on clock divider values latency of 0,1 cls_clk of instance j clock cycles can occur.

The above table shows the MCS interrupt lines which exist for interrupt signaling to an MCS[j].

24.5 ICM Configuration Registers Description

24.5.1 ICM_IRQG_0

Description	ICM Interrupt group register covering infrastructural and safety components (ARU, BRC, AEI, PSM0, PSM1, MAP, CMP, SPE)
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_0
Address	$0x400$
C-Name	GTM.CLS[0].ICM.IRQG_0

Interface: MCS[i]

Name	ICM_IRQG_0

Address	$0x400$
C-Name	

ARU_NEW_DATA0_IRQ	
Description	ARU_NEW_DATA0_I interrupt
Loop	-
Bit Range	$[0:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NARU} > 0$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

ARU_NEW_DATA1_IRQ	
Description	ARU_NEW_DATA1_I interrupt
Loop	-
Bit Range	$[1:1]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NARU} > 0$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

ARU_ACC_ACK_IRQ	
Description	ARU_ACC_ACK_I interrupt
Loop	-

ARU_ACC_ACK_IRQ	
Bit Range	$[2:2]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NARU} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

BRC_IRQ	
Description	BRC shared submodule interrupt
Loop	–
Bit Range	$[3:3]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NBRC} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

AEI_IRQ	
Description	ICM_IRQG_0.AEI_IRQ: AEI_IRQ interrupt
Loop	–
Bit Range	$[4:4]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–

AEI_IRQ	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the interrupt sources *AEI_TO_XPT_IRQ* , *AEI_USP_ADDR_IRQ* , *AEI_IM_ADDR_IRQ* , *AEI_USP_BE_IRQ* , *AEIM_USP_ADDR_IRQ* , *AEIM_IM_ADDR_IRQ* , *AEIM_USP_BE_IRQ* , *CLK_EN_ERR_IRQ* or *CLK_PER_ERR_IRQ* .

CMP_IRQ	
Description	CMP shared submodule interrupt
Loop	–
Bit Range	$[5:5]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NCMP} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

SPE[j]_IRQ	
Description	SPE[j] shared submodule interrupt
Loop	$j = \{n : 0 <= n <= 7\}$
Bit Range	$[j+6 : j+6]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$j < \mathrm{NSPE}$

SPE[j]_IRQ	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the five interrupt sources *SPE[j]_NIPD* , *SPE[j]_DCHG* , *SPE[j]_PERR* , *SPE[j]_BIS* or *SPE[j]_RCMP* .

PSM0_CH[x]_IRQ	
Description	PSM0 shared submodule channel [x] interrupt.
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the four interrupt sources *FIFO_EMPTY* , *FIFO_FULL* , *FIFO_LOWER_WM* or *FIFO_UPPER_WM* of FIFO0 channel [x].

PSM1_CH[x]_IRQ	
Description	PSM1 shared submodule channel [x] interrupt.
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–

PSM1_CH[x]_IRQ	
Condition	$NPSM > 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

24.5.2 ICM_IRQG_1

Description	ICM Interrupt group register covering DPLL
Loop	
Condition	$NDPLL > 0$
Storage Type	REGISTER
Clock Active Cond	$ICM\_CLK\_ENABLE == 1$

Interface: CPU

Name	ICM_IRQG_1
Address	$0x404$
C-Name	GTM.CLS[0].ICM.IRQG_1

Interface: MCS[i]

Name	ICM_IRQG_1
Address	$0x404$
C-Name	

DPLL_DCGI_IRQ	
Description	TRIGGER direction change detected.
Loop	–
Bit Range	$[0 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

DPLL_DCGI_IRQ	
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_EDI_IRQ	
Description	DPLL enable/disable interrupt
Loop	–
Bit Range	$[1:1]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the two interrupt sources *DPLL_PDI* or *DPLL_PEI* .

DPLL_TINI_IRQ	
Description	TRIGGER minimum hold time (THMI) violation detected interrupt.
Loop	–
Bit Range	$[2:2]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule

DPLL_TINI_IRQ	
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TAXI_IRQ	
Description	TRIGGER maximum hold time (THMA) violation detected interrupt.
Loop	–
Bit Range	$[3:3]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_SISI_IRQ	
Description	STATE inactive slope detected interrupt.
Loop	–
Bit Range	$[4:4]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TISI_IRQ	
Description	TRIGGER inactive slope detected interrupt.
Loop	–

DPLL_TISI_IRQ	
Bit Range	[5 : 5]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_MSI_IRQ	
Description	Missing STATE interrupt
Loop	–
Bit Range	[6 : 6]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_MTI_IRQ	
Description	Missing TRIGGER interrupt.
Loop	–
Bit Range	[7 : 7]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

DPLL_MTI_IRQ	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_SASI_IRQ	
Description	STATE active slope detected.
Loop	–
Bit Range	[8 : 8]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TASI_IRQ	
Description	TRIGGER active slope detected while DPLL_NTI_CNT.NTI_CNT is zero.
Loop	–
Bit Range	[9 : 9]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule

DPLL_TASI_IRQ	
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_PWI_IRQ	
Description	Plausibility window (PVT) violation interrupt of TRIGGER.
Loop	–
Bit Range	[10 : 10]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_W2I_IRQ	
Description	Write access to RAM region 2 interrupt.
Loop	–
Bit Range	[11 : 11]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_W1I_IRQ	
Description	Write access to RAM region 1b or 1c interrupt.
Loop	–

DPLL_W1I_IRQ	
Bit Range	[12 : 12]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_GL1I_IRQ	
Description	Get lock interrupt for SUB_INC1
Loop	–
Bit Range	[13 : 13]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_LL1I_IRQ	
Description	Loss of lock interrupt for SUB_INC1
Loop	–
Bit Range	[14 : 14]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

DPLL_LL1I_IRQ	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_EI_IRQ	
Description	Error interrupt
Loop	–
Bit Range	[15 : 15]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_GL2I_IRQ	
Description	Get lock interrupt for SUB_INC2
Loop	–
Bit Range	[16 : 16]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule

DPLL_GL2I_IRQ	
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_LL2I_IRQ	
Description	Loss of lock interrupt for SUB_INC2
Loop	–
Bit Range	[17 : 17]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TE0I_IRQ	
Description	TRIGGER event interrupt 0
Loop	–
Bit Range	[18 : 18]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TE1I_IRQ	
Description	TRIGGER event interrupt 1
Loop	–

DPLL_TE1I_IRQ	
Bit Range	[19 : 19]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TE2I_IRQ	
Description	TRIGGER event interrupt 2
Loop	–
Bit Range	[20 : 20]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TE3I_IRQ	
Description	TRIGGER event interrupt 3
Loop	–
Bit Range	[21 : 21]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

DPLL_TE3I_IRQ	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TE4I_IRQ	
Description	TRIGGER event interrupt 4
Loop	–
Bit Range	[22 : 22]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_CDTI_IRQ	
Description	DPLL calculated duration interrupt for trigger.
Loop	–
Bit Range	[23 : 23]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule

DPLL_CDTI_IRQ	
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_CDSI_IRQ	
Description	DPLL calculated duration interrupt for state.
Loop	–
Bit Range	$[24 : 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_TORI_IRQ	
Description	DPLL calculated duration interrupt for state.
Loop	–
Bit Range	$[25 : 25]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

DPLL_SORI_IRQ	
Description	DPLL calculated duration interrupt for state.
Loop	–

DPLL_SORI_IRQ	
Bit Range	$[26 : 26]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

24.5.3 ICM_IRQG_2

Description	ICM Interrupt group register covering TIM0, TIM1, TIM2, TIM3
Loop	
Condition	$\mathrm{NTIM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_2
Address	$0x408$
C-Name	GTM.CLS[0].ICM.IRQG_2

Interface: MCS[i]

Name	ICM_IRQG_2
Address	$0x408$
C-Name	

TIM0_CH[x]_IRQ	
Description	TIM0 shared interrupt channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

TIM0_CH[x]_IRQ	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the six interrupt sources *TIM_NEWVAL_IRQ* [x:x], *TIM_ECNTOFL_IRQ* [x:x], *TIM_CNTOFL_IRQ* [x:x], *TIM_GPROFL_IRQ* [x:x], *TIM_GLITCHDET_IRQ* [x:x] or *TO_DET* [x:x] of TIM instance 0, channel [x].

TIM1_CH[x]_IRQ	
Description	TIM1 shared interrupt channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the six interrupt sources *TIM_NEWVAL_IRQ* [x:x], *TIM_ECNTOFL_IRQ* [x:x], *TIM_CNTOFL_IRQ* [x:x], *TIM_GPROFL_IRQ* [x:x], *TIM_GLITCHDET_IRQ* [x:x] or *TO_DET* [x:x] of TIM instance 1, channel [x].

TIM2_CH[x]_IRQ	
Description	TIM2 shared interrupt channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False

TIM2_CH[x]_IRQ	
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the six interrupt sources *TIM_NEWVAL_IRQ* [x:x], *TIM_ECNTOFL_IRQ* [x:x], *TIM_CNTOFL_IRQ* [x:x], *TIM_GPROFL_IRQ* [x:x], *TIM_GLITCHDET_IRQ* [x:x] or *TO_DET* [x:x] of TIM instance 2, channel [x].

TIM3_CH[x]_IRQ	
Description	TIM3 shared interrupt channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the six interrupt sources *TIM_NEWVAL_IRQ* [x:x], *TIM_ECNTOFL_IRQ* [x:x], *TIM_CNTOFL_IRQ* [x:x], *TIM_GPROFL_IRQ* [x:x], *TIM_GLITCHDET_IRQ* [x:x] or *TO_DET* [x:x] of TIM instance 3, channel [x].

24.5.4 ICM_IRQG_3

Description	ICM Interrupt group register covering TIM4, TIM5, TIM6, TIM7
Loop	
Condition	$\mathrm{NTIM} > 4$

Storage Type	REGISTER
Clock Active Cond	ICM_CLK_ENABLE == 1

Interface: CPU

Name	ICM_IRQG_3
Address	$0x40C$
C-Name	GTM.CLS[0].ICM.IRQG_3

Interface: MCS[i]

Name	ICM_IRQG_3
Address	$0x40C$
C-Name	

TIM4_CH[x]_IRQ	
Description	TIM4 shared interrupt channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the six interrupt sources *TIM_NEWVAL_IRQ* [x:x], *TIM_ECNTOFL_IRQ* [x:x], *TIM_CNTOFL_IRQ* [x:x], *TIM_GPROFL_IRQ* [x:x], *TIM_GLITCHDET_IRQ* [x:x] or *TO_DET* [x:x] of TIM instance 4, channel [x].

TIM5_CH[x]_IRQ	
Description	TIM5 shared interrupt channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-

TIM5_CH[x]_IRQ	
Condition	$\mathrm{NTIM} > 5$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the six interrupt sources *TIM_NEWVAL_IRQ* [x:x], *TIM_ECNTOFL_IRQ* [x:x], *TIM_CNTOFL_IRQ* [x:x], *TIM_GPROFL_IRQ* [x:x], *TIM_GLITCHDET_IRQ* [x:x] or *TO_DET* [x:x] of TIM instance 5, channel [x].

TIM6_CH[x]_IRQ	
Description	TIM6 shared interrupt channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 6$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the six interrupt sources *TIM_NEWVAL_IRQ* [x:x], *TIM_ECNTOFL_IRQ* [x:x], *TIM_CNTOFL_IRQ* [x:x], *TIM_GPROFL_IRQ* [x:x], *TIM_GLITCHDET_IRQ* [x:x] or *TO_DET* [x:x] of TIM instance 6, channel [x].

TIM7_CH[x]_IRQ	
Description	TIM7 shared interrupt channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True

TIM7_CH[x]_IRQ	
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 7$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the six interrupt sources *TIM_NEWVAL_IRQ* [x:x], *TIM_ECNTOFL_IRQ* [x:x], *TIM_CNTOFL_IRQ* [x:x], *TIM_GPROFL_IRQ* [x:x], *TIM_GLITCHDET_IRQ* [x:x] or *TO_DET* [x:x] of TIM instance 7, channel [x].

24.5.5 ICM_IRQG_4

Description	ICM Interrupt group register covering MCS0 to MCS3 submodules
Loop	
Condition	$\mathrm{NMCS} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_4
Address	$0x410$
C-Name	GTM.CLS[0].ICM.IRQG_4

Interface: MCS[i]

Name	ICM_IRQG_4
Address	$0x410$
C-Name	

MCS0_CH[x]_IRQ	
Description	MCS0 channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–

MCS0_CH[x]_IRQ	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** This bit represents an OR function of the two interrupt sources **MCS[0]_CH[x]_IRQ_NOTIFY.ERR_IRQ** , **MCS[0]_CH[x]_IRQ_NOTIFY.MCS_IRQ** of MCS instance 0 channel [x].

MCS1_CH[x]_IRQ	
Description	MCS1 channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** This bit represents an OR function of the two interrupt sources **MCS[1]_CH[x]_IRQ_NOTIFY.ERR_IRQ** , **MCS[1]_CH[x]_IRQ_NOTIFY.MCS_IRQ** of MCS instance 1 channel [x].

MCS2_CH[x]_IRQ	
Description	MCS2 channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–

MCS2_CH[x]_IRQ	
Condition	$\mathrm{NMCS} > 2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** This bit represents an OR function of the two interrupt sources **MCS[2]_CH[x]_IRQ_NOTIFY.ERR_IRQ** , **MCS[2]_CH[x]_IRQ_NOTIFY.MCS_IRQ** of MCS instance 2 channel [x].

MCS3_CH[x]_IRQ	
Description	MCS3 channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > 3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** This bit represents an OR function of the two interrupt sources **MCS[3]_CH[x]_IRQ_NOTIFY.ERR_IRQ** , **MCS[3]_CH[x]_IRQ_NOTIFY.MCS_IRQ** of MCS instance 3 channel [x].

24.5.6 ICM_IRQG_5

Description	ICM Interrupt group register covering MCS4 to MCS7 submodules
Loop	
Condition	$\mathrm{NMCS} > 4$
Storage Type	REGISTER

Clock Active Cond	ICM_CLK_ENABLE == 1

Interface: CPU

Name	ICM_IRQG_5
Address	$0x414$
C-Name	GTM.CLS[0].ICM.IRQG_5

Interface: MCS[i]

Name	ICM_IRQG_5
Address	$0x414$
C-Name	

MCS4_CH[x]_IRQ	
Description	MCS4 channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** This bit represents an OR function of the two interrupt sources **MCS[4]_CH[x]_IRQ_NOTIFY.ERR_IRQ** , **MCS[4]_CH[x]_IRQ_NOTIFY.MCS_IRQ** of MCS instance 4 channel [x].

MCS5_CH[x]_IRQ	
Description	MCS5 channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$NMCS > 5$

MCS5_CH[x]_IRQ	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** This bit represents an OR function of the two interrupt sources **MCS[5]_CH[x]_IRQ_NOTIFY.ERR_IRQ** , **MCS[5]_CH[x]_IRQ_NOTIFY.MCS_IRQ** of MCS instance 5 channel [x].

MCS6_CH[x]_IRQ	
Description	MCS6 channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > 6$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** This bit represents an OR function of the two interrupt sources **MCS[6]_CH[x]_IRQ_NOTIFY.ERR_IRQ** , **MCS[6]_CH[x]_IRQ_NOTIFY.MCS_IRQ** of MCS instance 6 channel [x].

MCS7_CH[x]_IRQ	
Description	MCS7 channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–

MCS7_CH[x]_IRQ	
Condition	$NMCS > 7$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** This bit represents an OR function of the two interrupt sources **MCS[7]_CH[x]_IRQ_NOTIFY.ERR_IRQ** , **MCS[7]_CH[x]_IRQ_NOTIFY.MCS_IRQ** of MCS instance 7 channel [x].

24.5.7 ICM_IRQG_MEI

Description	ICM Interrupt group register for module error interrupt information
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$ICM\_CLK\_ENABLE == 1$

Interface: CPU

Name	ICM_IRQG_MEI
Address	$0x430$
C-Name	GTM.CLS[0].ICM.IRQG_MEI

Interface: MCS[i]

Name	ICM_IRQG_MEI
Address	$0x430$
C-Name	

GTM_EIRQ	
Description	AEI error interrupt request
Loop	–
Bit Range	$[0 : 0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

GTM_EIRQ	
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the interrupt sources *AEI_TO_XPT_EIRQ* , *AEI_USP_ADDR_EIRQ* , *AEI_IM_ADDR_EIRQ* , *AEI_USP_BE_EIRQ* , *AEIM_USP_ADDR_EIRQ* , *AEIM_IM_ADDR_EIRQ* , *AEIM_USP_BE_EIRQ* , *CLK_EN_ERR_IRQ* or *CLK_PER_ERR_IRQ* .

BRC_EIRQ	
Description	BRC error interrupt
Loop	–
Bit Range	$[1:1]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NBRC} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

FIFO[j]_EIRQ	
Description	FIFO[j] error interrupt
Loop	$j = \{n : 0 <= n <= 1\}$
Bit Range	$[j+2 : j+2]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$j < \mathrm{NPSM}$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

FIFO[j]_EIRQ	
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM[j]_EIRQ	
Description	TIM[j] error interrupt
Loop	$j = \{n : 0 <= n <= 7\}$
Bit Range	$[j+4 : j+4]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$j < \mathrm{NTIM}$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS[j]_EIRQ	
Description	MCS[j] error interrupt
Loop	$j = \{n : 0 <= n <= 7\}$
Bit Range	$[j+12 : j+12]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$j < \mathrm{NMCS}$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule

MCS[j]_EIRQ	
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

SPE[j]_EIRQ	
Description	SPE[j] error interrupt
Loop	$j = \{n : 0 <= n <= 3\}$
Bit Range	$[j + 20 : j + 20]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$j < \mathrm{NSPE}$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

CMP_EIRQ	
Description	CMP error interrupt
Loop	–
Bit Range	$[24 : 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NCMP} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

DPLL_EIRQ	
Description	DPLL error interrupt
Loop	–
Bit Range	$[25 : 25]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NDPLL} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

24.5.8 ICM_IRQG_CEI0

Description	ICM Interrupt group register 0 for channel error interrupt information
Loop	
Condition	$\mathrm{NPSM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_CEI0
Address	$0x434$
C-Name	GTM.CLS[0].ICM.IRQG_CEI0

Interface: MCS[i]

Name	ICM_IRQG_CEI0
Address	$0x434$
C-Name	

FIFO0_CH[x]_EIRQ	
Description	FIFO0 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True

FIFO0_CH[x]_EIRQ	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

FIFO1_CH[x]_EIRQ	
Description	FIFO1 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

FIFO2_CH[x]_EIRQ	
Description	FIFO2 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > 2$
Initial value	0

FIFO2_CH[x]_EIRQ	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

24.5.9 ICM_IRQG_CEI1

Description	ICM Interrupt group register 1 for channel error interrupt information
Loop	
Condition	$\mathrm{NTIM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_CEI1
Address	$0x438$
C-Name	GTM.CLS[0].ICM.IRQG_CEI1

Interface: MCS[i]

Name	ICM_IRQG_CEI1
Address	$0x438$
C-Name	

TIM0_CH[x]_EIRQ	
Description	TIM0 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule

TIM0_CH[x]_EIRQ	
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM1_CH[x]_EIRQ	
Description	TIM1 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM2_CH[x]_EIRQ	
Description	TIM2 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM3_CH[x]_EIRQ	
Description	TIM3 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

24.5.10 ICM_IRQG_CEI2

Description	ICM Interrupt group register 2 for channel error interrupt information
Loop	
Condition	$\mathrm{NTIM} > 4$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_CEI2
Address	$0x43C$
C-Name	GTM.CLS[0].ICM.IRQG_CEI2

Interface: MCS[i]

Name	ICM_IRQG_CEI2
Address	$0x43C$
C-Name	

TIM4_CH[x]_EIRQ	
Description	TIM4 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R

TIM4_CH[x]_EIRQ	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM5_CH[x]_EIRQ	
Description	TIM5 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 5$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM6_CH[x]_EIRQ	
Description	TIM6 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 6$

TIM6_CH[x]_EIRQ	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM7_CH[x]_EIRQ	
Description	TIM7 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > 7$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

24.5.11 ICM_IRQG_CEI3

Description	ICM Interrupt group register 3 for channel error interrupt information
Loop	
Condition	$\mathrm{NMCS} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_CEI3
Address	$0x440$
C-Name	GTM.CLS[0].ICM.IRQG_CEI3

Interface: MCS[i]

Name	ICM_IRQG_CEI3
Address	$0x440$
C-Name	

MCS0_CH[x]_EIRQ	
Description	MCS0 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS1_CH[x]_EIRQ	
Description	MCS1 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NMCS} > 1$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS2_CH[x]_EIRQ	
Description	MCS2 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NMCS} > 2$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS3_CH[x]_EIRQ	
Description	MCS3 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NMCS} > 3$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

24.5.12 ICM_IRQG_CEI4

Description	ICM Interrupt group register 4 for channel error interrupt information
Loop	
Condition	$\mathrm{NMCS} > 4$

Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_CEI4
Address	$0x444$
C-Name	GTM.CLS[0].ICM.IRQG_CEI4

Interface: MCS[i]

Name	ICM_IRQG_CEI4
Address	$0x444$
C-Name	

MCS4_CH[x]_EIRQ	
Description	MCS4 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS5_CH[x]_EIRQ	
Description	MCS5 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NMCS} > 5$
Initial value	0

MCS5_CH[x]_EIRQ	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS6_CH[x]_EIRQ	
Description	MCS6 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > 6$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS7_CH[x]_EIRQ	
Description	MCS7 channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > 7$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no error interrupt occurred 1 : error interrupt was raised by the corresponding submodule

MCS7_CH[x]_EIRQ	
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

24.5.13 ICM_IRQG_MCS[j]_CI

Description	ICM Interrupt group MCS[j] for Channel Interrupt information
Loop	$j = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	$j < \text{NMCS}$
Storage Type	REGISTER
Clock Active Cond	$\text{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_MCS[j]_CI
Address	$0x4 * j + 0x520$
C-Name	GTM.CLS[0].ICM.IRQG_MCS_CI[j]

Interface: MCS[i]

Name	ICM_IRQG_MCS[j]_CI
Address	$0x4 * j + 0x520$
C-Name	

MCS_CH[x]_IRQ	
Description	MCS-channel [x] interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

24.5.14 ICM_IRQG_MCS[j]_CEI

Description	ICM Interrupt group MCS[j] for Channel Error Interrupt information
Loop	$j = \{n : 0 <= n <= \text{NMCS} - 1\}$
Condition	$j < \text{NMCS}$
Storage Type	REGISTER
Clock Active Cond	$\text{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_MCS[j]_CEI
Address	$0x4 * j + 0x464$
C-Name	GTM.CLS[0].ICM.IRQG_MCS_CEI[j]

Interface: MCS[i]

Name	ICM_IRQG_MCS[j]_CEI
Address	$0x4 * j + 0x464$
C-Name	

MCS_CH[x]_EIRQ	
Description	MCS-channel [x] error interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

24.5.15 ICM_IRQG_SPE_CI

Description	ICM Interrupt group SPE for module Interrupt information
Loop	
Condition	$\text{NSPE} > 0$
Storage Type	REGISTER

Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_SPE_CI
Address	$0x570$
C-Name	GTM.CLS[0].ICM.IRQG_SPE_CI

Interface: MCS[i]

Name	ICM_IRQG_SPE_CI
Address	$0x570$
C-Name	

SPE[j]_IRQ	
Description	SPE[j] interrupt
Loop	$j = \{n : 0 <= n <= \mathrm{NSPE} - 1\}$
Bit Range	$[j : j]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

24.5.16 ICM_IRQG_SPE_CEI

Description	ICM Interrupt group SPE for module Error Interrupt information
Loop	
Condition	$\mathrm{NSPE} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_SPE_CEI
Address	$0x4B4$

C-Name	GTM.CLS[0].ICM.IRQG_SPE_CEI

Interface: MCS[i]

Name	ICM_IRQG_SPE_CEI
Address	$0x4B4$
C-Name	

SPE[j]_EIRQ	
Description	SPE[j] error interrupt
Loop	$j = \{n : 0 <= n <= \text{NSPE} - 1\}$
Bit Range	$[j : j]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule.

24.5.17 ICM_IRQG_PSM_0_CI

Description	ICM Interrupt group PSM 0 for Channel Interrupt information of FIFO0, FIFO1, FIFO2
Loop	
Condition	$\text{NPSM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\text{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_PSM_0_CI
Address	$0x560$
C-Name	GTM.CLS[0].ICM.IRQG_PSM_0_CI

Interface: MCS[i]

Name	ICM_IRQG_PSM_0_CI
Address	$0x560$
C-Name	

PSM_M0_CH[x]_IRQ	
Description	PSM0 channel [x] shared interrupt
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	0 = no interrupt occurred 1 = interrupt was raised by the corresponding submodule **Note:** This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the four interrupt sources *FIFO_EMPTY* , *FIFO_FULL* , *FIFO_LOWER_WM* or *FIFO_UPPER_WM* of FIFO0 channel [x].

PSM_M1_CH[x]_IRQ	
Description	PSMm channel [x] shared interrupt ([j]=4*0+1)
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	0 = no interrupt occurred 1 = interrupt was raised by the corresponding submodule **Note:** This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the four interrupt sources *FIFO_EMPTY* , *FIFO_FULL* , *FIFO_LOWER_WM* or *FIFO_UPPER_WM* of FIFO instance 1 channel [x].

PSM_M2_CH[x]_IRQ	
Description	PSMm channel [x] shared interrupt ([j]=4*0+2)
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NPSM} > 2$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
	0 = no interrupt occurred 1 = interrupt was raised by the corresponding submodule **Note:** This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the four interrupt sources *FIFO_EMPTY* , *FIFO_FULL* , *FIFO_LOWER_WM* or *FIFO_UPPER_WM* of FIFO instance 2 channel [x].

24.5.18 ICM_IRQG_PSM_0_CEI

Description	ICM Interrupt group PSM 0 for Channel Error Interrupt information of FIFO0, FIFO1, FIFO2
Loop	
Condition	$\mathrm{NPSM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_PSM_0_CEI
Address	$0x4A4$
C-Name	GTM.CLS[0].ICM.IRQG_PSM_0_CEI

Interface: MCS[i]

Name	ICM_IRQG_PSM_0_CEI
Address	$0x4A4$
C-Name	

PSM_M0_CH[x]_EIRQ	
Description	PSMm channel [x] error interrupt ([j]=4*0+0)
Loop	$x = \{n : 0 <= n <= 7\}$

PSM_M0_CH[x]_EIRQ	
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: 0 = no interrupt occurred 1 = error interrupt was raised by the corresponding submodule **Note:** This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

PSM_M1_CH[x]_EIRQ	
Description	PSM[j] channel [x] error interrupt ([j]=4*0+1)
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: 0 = no interrupt occurred 1 = error interrupt was raised by the corresponding submodule **Note:** This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

PSM_M2_CH[x]_EIRQ	
Description	PSM[j] channel [x] error interrupt ([j]=4*0+2)
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True

PSM_M2_CH[x]_EIRQ	
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > 2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	Note: 0 = no interrupt occurred 1 = error interrupt was raised by the corresponding submodule **Note:** This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

24.5.19 ICM_IRQG_TOM_[g]_CI

Description	ICM Interrupt group TOM [g] for Channel Interrupt information of TOMm ([j]=2*[g]+(0..1))
Loop	$g = \{n : 0 <= n <= \mathrm{ceil}(\mathrm{NTOM}/2) - 1\}$
Condition	$\mathrm{NTOM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_TOM_0_CI
Address	$0x5A0$
C-Name	GTM.CLS[0].ICM.IRQG_TOM_0_CI
Name	ICM_IRQG_TOM_1_CI
Address	$0x5A4$
C-Name	GTM.CLS[0].ICM.IRQG_TOM_1_CI
Name	ICM_IRQG_TOM_2_CI
Address	$0x5A8$
C-Name	GTM.CLS[0].ICM.IRQG_TOM_2_CI
Name	ICM_IRQG_TOM_3_CI
Address	$0x5AC$
C-Name	GTM.CLS[0].ICM.IRQG_TOM_3_CI
Name	ICM_IRQG_TOM_4_CI
Address	$0x5B0$
C-Name	GTM.CLS[0].ICM.IRQG_TOM_4_CI
Name	ICM_IRQG_TOM_5_CI
Address	$0x5B4$

C-Name	GTM.CLS[0].ICM.IRQG_TOM_5_CI

Interface: MCS[i]

Name	ICM_IRQG_TOM_0_CI
Address	$0x5A0$
C-Name	
Name	ICM_IRQG_TOM_1_CI
Address	$0x5A4$
C-Name	
Name	ICM_IRQG_TOM_2_CI
Address	$0x5A8$
C-Name	
Name	ICM_IRQG_TOM_3_CI
Address	$0x5AC$
C-Name	
Name	ICM_IRQG_TOM_4_CI
Address	$0x5B0$
C-Name	
Name	ICM_IRQG_TOM_5_CI
Address	$0x5B4$
C-Name	

TOM_M0_CH[x]_IRQ	
Description	TOM[j] channel [x] interrupt ([j]=2*[g]+0)
Loop	$x = \{n : 0 <= n <= 15\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule

TOM_M0_CH[x]_IRQ	
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the two interrupt sources *TOM_CCU0TC[x]_IRQ* or *TOM_CCU1TC[x]_IRQ* of TOM instance 0 channel [x].

TOM_M1_CH[x]_IRQ	
Description	TOM[j] channel [x] interrupt ([j]=2*[g]+1)
Loop	$x = \{n : 0 <= n <= 15\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTOM} > g * 2 + 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the two interrupt sources *TOM_CCU0TC[x]_IRQ* or *TOM_CCU1TC[x]_IRQ* of TOM instance 1 channel [x].

24.5.20 ICM_IRQG_ATOM_[g]_CI

Description	ICM Interrupt group ATOM [g] for Channel Interrupt information of ATOM[j] (m=4*[g]+(0..3))
Loop	$g = \{n : 0 <= n <= \mathrm{ceil}(\mathrm{NATOM}/4) - 1\}$
Condition	$\mathrm{NATOM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_ATOM_0_CI
Address	$0x590$
C-Name	GTM.CLS[0].ICM.IRQG_ATOM_0_CI
Name	ICM_IRQG_ATOM_1_CI

Address	$0x594$
C-Name	GTM.CLS[0].ICM.IRQG_ATOM_1_CI
Name	ICM_IRQG_ATOM_2_CI
Address	$0x598$
C-Name	GTM.CLS[0].ICM.IRQG_ATOM_2_CI

Interface: MCS[i]

Name	ICM_IRQG_ATOM_0_CI
Address	$0x590$
C-Name	
Name	ICM_IRQG_ATOM_1_CI
Address	$0x594$
C-Name	
Name	ICM_IRQG_ATOM_2_CI
Address	$0x598$
C-Name	

ATOM_M0_CH[x]_IRQ	
Description	ATOM[j] channel [x] interrupt ([j]=4*[g]+0)
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the two interrupt sources *CCU0TC[x]_IRQ* or *CCU1TC[x]_IRQ* of ATOM instance 0 channel [x].

ATOM_M1_CH[x]_IRQ	
Description	ATOM[j] channel [x] interrupt ([j]=4*[g]+1)
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$

ATOM_M1_CH[x]_IRQ	
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NATOM} > g * 4 + 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the two interrupt sources *CCU0TC[x]_IRQ* or *CCU1TC[x]_IRQ* of ATOM instance 1 channel [x].

ATOM_M2_CH[x]_IRQ	
Description	ATOMm channel [x] interrupt ([j]=4*[g]+2)
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NATOM} > g * 4 + 2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the two interrupt sources *CCU0TC[x]_IRQ* or *CCU1TC[x]_IRQ* of ATOM instance 2 channel [x].

ATOM_M3_CH[x]_IRQ	
Description	ATOM[j] channel [x] interrupt ([j]=4*[g]+3)
Loop	$x = \{n : 0 <= n <= 7\}$

ATOM_M3_CH[x]_IRQ	
Bit Range	$[x+24:x+24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NATOM} > g*4+3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : no interrupt occurred 1 : interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the interrupt is enabled in the interrupt enable register of the corresponding submodule. **Note:** Set this bit represents an OR function of the two interrupt sources *CCU0TC[x]_IRQ* or *CCU1TC[x]_IRQ* of ATOM instance 3 channel [x].

24.5.21 ICM_IRQG_CLS_[g]_MEI

Description	ICM Interrupt group for module Error Interrupt information for each TIM[j], MCS[j], SPE[j], FIFO[j] (j=4*[g]+k, k∈{0, ... ,3})
Loop	$g = \{n : 0 <= n <= \mathrm{ceil}(\max(\max(\max(\mathrm{NMCS}, \mathrm{NTIM}), \mathrm{NSPE}), \mathrm{NPSM})/4) - 1\}$
Condition	$\mathrm{NTIM} > 0 \vert\vert \mathrm{NMCS} > 0 \vert\vert \mathrm{NSPE} > 0 \vert\vert \mathrm{NPSM} > 0$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{ICM\_CLK\_ENABLE} == 1$

Interface: CPU

Name	ICM_IRQG_CLS_0_MEI
Address	$0x510$
C–Name	GTM.CLS[0].ICM.IRQG_CLS_0_MEI
Name	ICM_IRQG_CLS_1_MEI
Address	$0x514$
C–Name	GTM.CLS[0].ICM.IRQG_CLS_1_MEI
Name	ICM_IRQG_CLS_2_MEI
Address	$0x518$
C–Name	GTM.CLS[0].ICM.IRQG_CLS_2_MEI

Interface: MCS[i]

Name	ICM_IRQG_CLS_0_MEI
Address	$0x510$

C-Name	
Name	ICM_IRQG_CLS_1_MEI
Address	$0x514$
C-Name	
Name	ICM_IRQG_CLS_2_MEI
Address	$0x518$
C-Name	

TIM_M0_EIRQ	
Description	Error interrupt TIM[j]_EIRQ (j=4*[g])
Loop	-
Bit Range	$[0:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NTIM} > g * 4$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS_M0_EIRQ	
Description	Error interrupt MCS[j]_EIRQ (j=4*[g])
Loop	-
Bit Range	$[1:1]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{NMCS} > g * 4$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule

MCS_M0_EIRQ	
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

SPE_M0_EIRQ	
Description	Error interrupt SPE[j]_EIRQ ([j]=4*[g]+0)
Loop	–
Bit Range	$[2:2]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NSPE} > g*4$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

FIFO_M0_EIRQ	
Description	Error interrupt FIFO[j]_EIRQ ([j]=4*[g]+0)
Loop	–
Bit Range	$[3:3]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > g*4$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM_M1_EIRQ	
Description	Error interrupt TIM[j]_EIRQ ([j]=4*[g]+1)
Loop	–
Bit Range	$[8 : 8]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > g * 4 + 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS_M1_EIRQ	
Description	Error interrupt MCS[j]_EIRQ ([j]=4*[g]+1)
Loop	–
Bit Range	$[9 : 9]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > g * 4 + 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

SPE_M1_EIRQ	
Description	Error interrupt SPE[j]_EIRQ ([j]=4*[g]+1)
Loop	–
Bit Range	$[10 : 10]$
Access Type	R
Volatile	True

SPE_M1_EIRQ	
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NSPE} > g * 4 + 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

FIFO_M1_EIRQ	
Description	Error interrupt FIFO[j]_EIRQ ([j]=4*[g]+1)
Loop	–
Bit Range	$[11 : 11]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > g * 4 + 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM_M2_EIRQ	
Description	Error interrupt TIM[j]_EIRQ ([j]=4*[g]+2)
Loop	–
Bit Range	$[16 : 16]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > g * 4 + 2$
Initial value	0
Protect Enable Cond	–

TIM_M2_EIRQ	
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS_M2_EIRQ	
Description	Error interrupt MCS[j]_EIRQ ([j]=4*[g]+2)
Loop	–
Bit Range	$[17 : 17]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > g * 4 + 2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

SPE_M2_EIRQ	
Description	Error interrupt SPE[j]_EIRQ ([j]=4*[g]+2)
Loop	–
Bit Range	$[18 : 18]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NSPE} > g * 4 + 2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule

SPE_M2_EIRQ	
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

FIFO_M2_EIRQ	
Description	Error interrupt FIFO[j]_EIRQ ([j]=4*[g]+2)
Loop	–
Bit Range	$[19:19]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > g*4+2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

TIM_M3_EIRQ	
Description	Error interrupt TIM[j]_EIRQ ([j]=4*[g]+3)
Loop	–
Bit Range	$[24:24]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NTIM} > g*4+3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

MCS_M3_EIRQ	
Description	Error interrupt MCS[j]_EIRQ ([j]=4*[g]+3)
Loop	–
Bit Range	$[25 : 25]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NMCS} > g * 4 + 3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

SPE_M3_EIRQ	
Description	Error interrupt SPE[j]_EIRQ ([j]=4*[g]+3)
Loop	–
Bit Range	$[26 : 26]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NSPE} > g * 4 + 3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

FIFO_M3_EIRQ	
Description	Error interrupt FIFO[j]_EIRQ ([j]=4*[g]+2)
Loop	–
Bit Range	$[27 : 27]$
Access Type	R
Volatile	True

FIFO_M3_EIRQ	
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NPSM} > g * 4 + 3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : no interrupt occurred 1 : error interrupt was raised by the corresponding submodule
	Note: This bit is only set, when the error interrupt is enabled in the error interrupt enable register of the corresponding submodule.

24.6 ICM Signal Description

24.6.1 ICM TOM Interrupt Signals

Loop	$j = \{n : 0 <= n <= \mathrm{NTOM} - 1\}$
Condition	–
Format	–

TOM[j]_CH0_IRQ	
Description	TOM[j] interrupt signal channel 0
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH1_IRQ	
Description	TOM[j] interrupt signal channel 1
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH2_IRQ	
Description	TOM[j] interrupt signal channel 2
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH3_IRQ	
Description	TOM[j] interrupt signal channel 3
Loop	–
Condition	–

TOM[j]_CH3_IRQ	
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH4_IRQ	
Description	TOM[j] interrupt signal channel 4
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH5_IRQ	
Description	TOM[j] interrupt signal channel 5
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH6_IRQ	
Description	TOM[j] interrupt signal channel 6
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH7_IRQ	
Description	TOM[j] interrupt signal channel 7
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH8_IRQ	
Description	TOM[j] interrupt signal channel 8
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH9_IRQ	
Description	TOM[j] interrupt signal channel 9
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH10_IRQ	
Description	TOM[j] interrupt signal channel 10

TOM[j]_CH10_IRQ	
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH11_IRQ	
Description	TOM[j] interrupt signal channel 11
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH12_IRQ	
Description	TOM[j] interrupt signal channel 12
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH13_IRQ	
Description	TOM[j] interrupt signal channel 13
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH14_IRQ	
Description	TOM[j] interrupt signal channel 14
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

TOM[j]_CH15_IRQ	
Description	TOM[j] interrupt signal channel 15
Loop	–
Condition	–
Signal Type	ARRAY[NTOM]
Assignment	–

24.6.2 ICM ATOM Interrupt Signals

Loop	$j = \{n : 0 <= n <= \mathrm{NATOM} - 1\}$
Condition	–
Format	–

ATOM[j]_CH0_IRQ	
Description	ATOM[j] interrupt signal channel 0
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	–

ATOM[j]_CH1_IRQ	
Description	ATOM[j] interrupt signal channel 1
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	–

ATOM[j]_CH2_IRQ	
Description	ATOM[j] interrupt signal channel 2
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	–

ATOM[j]_CH3_IRQ	
Description	ATOM[j] interrupt signal channel 3
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	–

ATOM[j]_CH4_IRQ	
Description	ATOM[j] interrupt signal channel 4
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	–

ATOM[j]_CH5_IRQ	
Description	ATOM[j] interrupt signal channel 5
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	–

ATOM[j]_CH6_IRQ	
Description	ATOM[j] interrupt signal channel 6
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	–

ATOM[j]_CH7_IRQ	
Description	ATOM[j] interrupt signal channel 7
Loop	–
Condition	–
Signal Type	ARRAY[NATOM]
Assignment	–

24.6.3 Interrupt Signals for MCS

Loop	$j = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	–
Format	–

MCS_TIM[j]_IRQ	
Description	TIM[j] Shared interrupts
Loop	–
Condition	–
Signal Type	ARRAY[NMCS]
Assignment	–

MCS_TOM[j]_IRQ	
Description	TOM[j] Shared interrupts
Loop	–
Condition	–
Signal Type	ARRAY[NMCS]
Assignment	–

MCS_ATOM[j]_IRQ	
Description	ATOM[j] Shared interrupts
Loop	–
Condition	–
Signal Type	ARRAY[NMCS]
Assignment	–

MCS_TIMATOM[j]_IRQ	
Description	TOM[j+1], ATOM[j+1], TIM[j+1] Shared interrupts
Loop	–
Condition	–
Signal Type	ARRAY[NMCS]
Assignment	–

MCS_TIO[j]_G[g]_IRQ	
Description	TIO[j] Shared interrupts
Loop	–
Condition	–
Signal Type	ARRAY[NMCS][NTIO_CH8]
Assignment	–

MCS_NMCS[j]_IRQ	
Description	MCS[j+1] Interrupts
Loop	–
Condition	–
Signal Type	ARRAY[NMCS]
Assignment	–

MCS_MCS[0]_IRQ	
Description	MCS[0] Interrupt
Loop	–
Condition	–
Signal Type	ARRAY[1]
Assignment	–

25 Output Compare Unit (CMP)

25.1 Overview

The Output Compare Unit (CMP) is designed for the use in safety relevant applications. The main idea is to have the possibility to duplicate outputs in order to be compared in this unit. Because of the simple EXOR function used, it is necessary to ensure the total cycle accurate output behavior of the output modules to be compared. This is given when two neighboring DTM channels (e.g. *CDTM0_DTM4_CH0_OUT* and *CDTM0_DTM4_CH1_OUT* , for neighboring DTM channel pairs see *25.7.4 "CMP ABWC Comparator Input 1"*) generate identical signals with phase shift zero at their outputs. This can be reached if they start their output generation at the same time. This start of synchronization is possible by means of the trigger mechanisms provided by the TOM or ATOM as shown in the TOM chapter *14 "Timer Output Module (TOM)"* or ATOM chapter *15 "ARU-connected Timer Output Module (ATOM)"* .

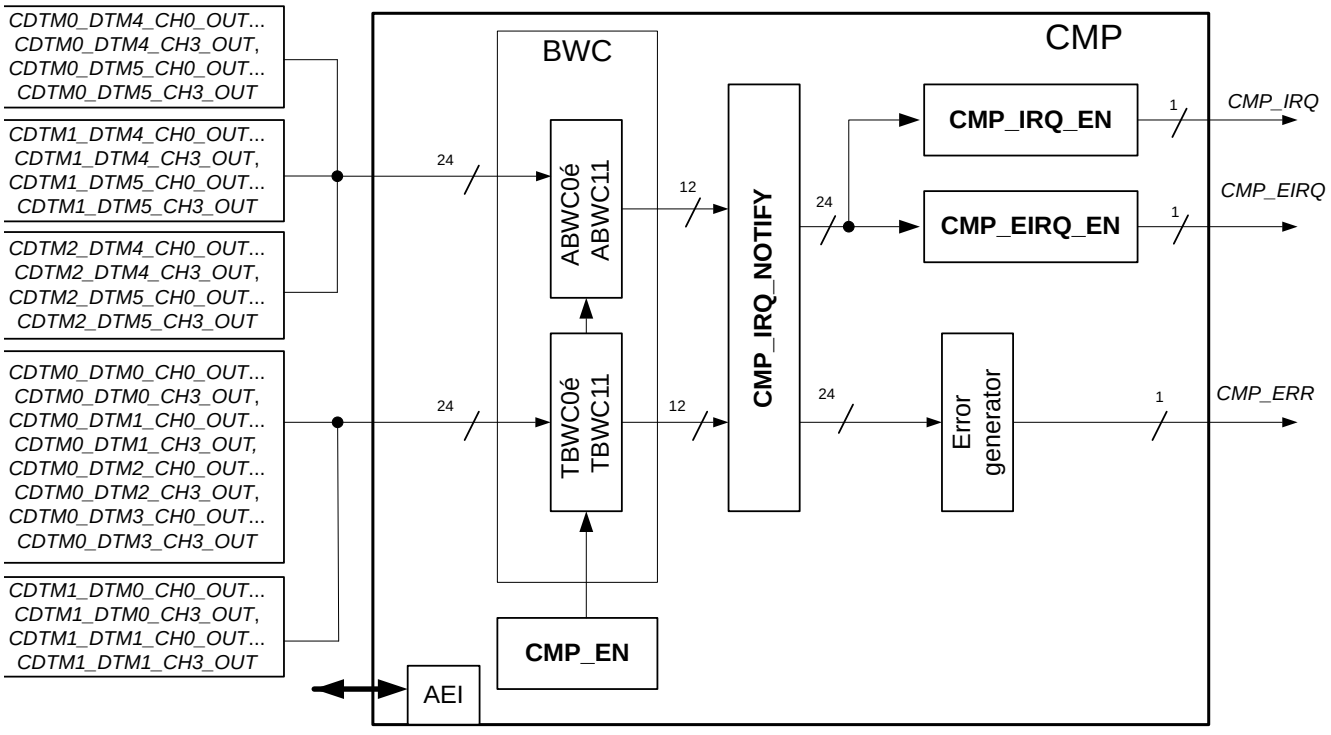
Indices and their range as used inside this chapter are:

- i:={0, 1, 2} (index of cluster or module instance)
- c:={0, 1,..., 11} (index of bitwise comparators)
- d:={0, 1,..., 5} (index of DTM unit after TOM/ ATOM)
- x:={0, 1, 2} (index of output channel in DTM)

The CMP enables the comparison of 2x24 channels of the CDTM[i]_DTM[d] instances and is restricted to neighboring channels. The first 24 CMP channels are the 24 DTM channels placed behind TOM[0] and TOM[1] instances and the second 24 CMP channels are the 24 DTM channels placed behind the ATOM[0], ATOM[1] and ATOM[2] instances.

Note: In case the cluster 1 is operating on half GTM system clock *CLK* and signals in cluster 0 or 2 are generated on a GTM system clock *CLK* resolution, a deviation of the compare signals by one *CLK* clock cycle might not be detected by the bitwise compare unit.

Figure 163 Architecture of the Compare Unit



25.2 Bitwise Compare Unit (BWC)

The Bitwise Compare Unit ABWC compares the combinations shown in following table in pairs.

Table 56 ABWC compare unit

ABWC Comparator	Comparator Input 1	Comparator Input 2
ABWC0	*CDTM0_DTM4_CH0_OUT*	*CDTM0_DTM4_CH1_OUT*
ABWC1	*CDTM0_DTM4_CH2_OUT*	*CDTM0_DTM4_CH3_OUT*
ABWC2	*CDTM0_DTM5_CH0_OUT*	*CDTM0_DTM5_CH1_OUT*

ABWC Comparator	Comparator Input 1	Comparator Input 2
ABWC3	*CDTM0_DTM5_CH2_OUT*	*CDTM0_DTM5_CH3_OUT*
ABWC4	*CDTM1_DTM4_CH0_OUT*	*CDTM1_DTM4_CH1_OUT*
ABWC5	*CDTM1_DTM4_CH2_OUT*	*CDTM1_DTM4_CH3_OUT*
ABWC6	*CDTM1_DTM5_CH0_OUT*	*CDTM1_DTM5_CH1_OUT*
ABWC7	*CDTM1_DTM5_CH2_OUT*	*CDTM1_DTM5_CH3_OUT*
ABWC8	*CDTM2_DTM4_CH0_OUT*	*CDTM2_DTM4_CH1_OUT*
ABWC9	*CDTM2_DTM4_CH2_OUT*	*CDTM2_DTM4_CH3_OUT*
ABWC10	*CDTM2_DTM5_CH0_OUT*	*CDTM2_DTM5_CH1_OUT*
ABWC11	*CDTM2_DTM5_CH2_OUT*	*CDTM2_DTM5_CH3_OUT*

The Bitwise Compare Unit TBWC compares the combinations shown in following table in pairs.

Table 57 TBWC compare unit

TBWC Comparator	Comparator Input 1	Comparator Input 2
TBWC0	*CDTM0_DTM0_CH0_OUT*	*CDTM0_DTM0_CH1_OUT*
TBWC1	*CDTM0_DTM0_CH2_OUT*	*CDTM0_DTM0_CH3_OUT*
TBWC2	*CDTM0_DTM1_CH0_OUT*	*CDTM0_DTM1_CH1_OUT*
TBWC3	*CDTM0_DTM1_CH2_OUT*	*CDTM0_DTM1_CH3_OUT*
TBWC4	*CDTM0_DTM2_CH0_OUT*	*CDTM0_DTM2_CH1_OUT*
TBWC5	*CDTM0_DTM2_CH2_OUT*	*CDTM0_DTM2_CH3_OUT*
TBWC6	*CDTM0_DTM3_CH0_OUT*	*CDTM0_DTM3_CH1_OUT*
TBWC7	*CDTM0_DTM3_CH2_OUT*	*CDTM0_DTM3_CH3_OUT*
TBWC8	*CDTM1_DTM0_CH0_OUT*	*CDTM1_DTM0_CH1_OUT*
TBWC9	*CDTM1_DTM0_CH2_OUT*	*CDTM1_DTM0_CH3_OUT*
TBWC10	*CDTM1_DTM1_CH0_OUT*	*CDTM1_DTM1_CH1_OUT*
TBWC11	*CDTM1_DTM1_CH2_OUT*	*CDTM1_DTM1_CH3_OUT*

25.3 Configuration of the Compare Unit

Because of the restrictions described in the section above, the Compare Unit consists of 24 antivalence (EXOR) elements, a select register **CMP_EN** which selects the corresponding comparisons and a status register **CMP_IRQ_NOTIFY** which shows and stores each mismatching result, when selected.

For each mismatching error enabled with **CMP_IRQ_EN** an interrupt signal on *CMP_IRQ* is generated.

For each mismatching error enabled with **CMP_EIRQ_EN** an interrupt signal on *CMP_EIRQ* is generated.

25.4 Error Generator

The error generator generates an error signal to be transmitted to the MON unit directly and independent of the *CMP_IRQ* and *CMP_EIRQ* . The error is set when in the **CMP_IRQ_NOTIFY** register at least one bit is set. The **CMP_IRQ_NOTIFY** bits are not maskable for this purpose.

Additionally, *CMP_ERR* is a primary output port for interrupt actions by CPU itself.

25.5 CMP Interrupt Signals

The CMP submodule has two interrupt signals, one normal interrupt and one error interrupt. The source of both interrupts can be determined by reading the **CMP_IRQ_NOTIFY** register under consideration of the registers **CMP_IRQ_EN** and **CMP_EIRQ_EN** . Each source can be forced separately for debug purposes using the interrupt force **CMP_IRQ_FORCINT** register. **CMP_IRQ_MODE** configures interrupt output characteristic. All interrupt modes are described in detail in section *3.12 "GTM-IP Interrupt Concept"* .

25.6 CMP Configuration Registers Description

25.6.1 CMP_EN

Description	CMP comparator enable register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMP\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMP_EN
Address	$0x206C0$
C-Name	GTM.CLS[1].CMP.EN

Interface: MCS[i]

Name	CMP_EN
Address	$0x6C0$
C-Name	

ABWC[c]_EN	
Description	Enable comparator channel [c] in ABWC
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c : c]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$c < \min(4 * \mathrm{NATOM}, 12)$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : ABWC Comparator [c] is disabled 1 : ABWC Comparator [c] is enabled
	(see chapter *25.2 "Bitwise Compare Unit (BWC)"*)

TBWC[c]_EN	
Description	Enable comparator channel [c] in TBWC
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c + 12 : c + 12]$
Access Type	RW
Volatile	False

TBWC[c]_EN	
Multithread	False
ModifiedWriteValue	-
Condition	$c < \min(8 * \mathrm{NTOM}, 12)$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : TBWC comparator [c] is disabled 1 : TBWC comparator [c] is enabled
	(see chapter *25.2 "Bitwise Compare Unit (BWC)"*)

25.6.2 CMP_IRQ_NOTIFY

Description	CMP event notification register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMP\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMP_IRQ_NOTIFY
Address	$0x206C4$
C-Name	GTM.CLS[1].CMP.IRQ_NOTIFY

Interface: MCS[i]

Name	CMP_IRQ_NOTIFY
Address	$0x6C4$
C-Name	

ABWC[c]	
Description	ATOM submodules output bitwise comparator [c] error indication
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c : c]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$c < \min(4 * \mathrm{NATOM}, 12)$
Initial value	0
Protect Enable Cond	-

ABWC[c]	
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

TBWC[c]	
Description	TOM submodules output bitwise comparator [c] error indication
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c + 12 : c + 12]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$c < \min(8 * \mathrm{NTOM}, 12)$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt
	Note: This bit will be cleared on a CPU write access of value '1'. A read access leaves the bit unchanged.

25.6.3 CMP_IRQ_EN

Description	CMP interrupt enable register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMP\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMP_IRQ_EN
Address	$0x206C8$
C-Name	GTM.CLS[1].CMP.IRQ_EN

Interface: MCS[i]

Name	CMP_IRQ_EN
Address	$0x6C8$
C-Name	

ABWC[c]_EN_IRQ	
Description	Enable CMP_IRQ_NOTIFY.ABWC[c] interrupt source for CMP_IRQ line
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c : c]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$c < \min(4 * \mathrm{NATOM}, 12)$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : interrupt source of ABWC comparator [c] is disabled 1 : interrupt source of ABWC comparator [c] is enabled

TBWC[c]_EN_IRQ	
Description	Enable CMP_IRQ_NOTIFY.TBWC[c] interrupt source for CMP_IRQ line
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c + 12 : c + 12]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$c < \min(8 * \mathrm{NTOM}, 12)$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : interrupt source of TBWC comparator [c] is disabled 1 : interrupt source of TBWC comparator [c] is enabled

25.6.4 CMP_IRQ_FORCINT

Description	CMP interrupt force register
Loop	
Condition	
Storage Type	REGISTER

Clock Active Cond	CMP_CLK_ENABLE == 1

Interface: CPU

Name	CMP_IRQ_FORCINT
Address	$0x206CC$
C-Name	GTM.CLS[1].CMP.IRQ_FORCINT

Interface: MCS[i]

Name	CMP_IRQ_FORCINT
Address	$0x6CC$
C-Name	

TRG_ABWC[c]	
Description	Trigger the bit CMP_IRQ_NOTIFY.ABWC[c] bit by software
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c : c]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	$c < \min(4 * \mathrm{NATOM}, 12)$
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[1]_AEI_ARB_WDATA, 23, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **CMP_IRQ_NOTIFY.ABWC[c]**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** .

TRG_TBWC[c]	
Description	Trigger CMP_IRQ_NOTIFY.TBWC[c] bit by software
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c + 12 : c + 12]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	$c < \min(8 * \mathrm{NTOM}, 12)$

TRG_TBWC[c]	
Initial value	0
Protect Enable Cond	(RF_PROT == 1) && (bitrange(CLS[1]_AEI_ARB_WDATA, 23, 0) != 0)
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **CMP_IRQ_NOTIFY.TBWC[c]**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** .

25.6.5 CMP_IRQ_MODE

Description	CMP interrupt mode configuration register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMP\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMP_IRQ_MODE
Address	$0x206D0$
C-Name	GTM.CLS[1].CMP.IRQ_MODE

Interface: MCS[i]

Name	CMP_IRQ_MODE
Address	$0x6D0$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	-
Bit Range	$[1:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	$\mathrm{IRQ\_MODE\_RST\_VAL}$
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async

IRQ_MODE	
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

25.6.6 CMP_EIRQ_EN

Description	CMP error interrupt enable register
Loop	
Condition	
Storage Type	REGISTER
Clock Active Cond	$\mathrm{CMP\_CLK\_ENABLE} == 1$

Interface: CPU

Name	CMP_EIRQ_EN
Address	$0x206D4$
C-Name	GTM.CLS[1].CMP.EIRQ_EN

Interface: MCS[i]

Name	CMP_EIRQ_EN
Address	$0x6D4$
C-Name	

ABWC[c]_EN_EIRQ	
Description	Enable ABWC comparator [c] interrupt source for CMP_EIRQ line
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c : c]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$c < \min(4 * \mathrm{NATOM}, 12)$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : interrupt source of ABWC comparator [c] is disabled 1 : interrupt source of ABWC comparator [c] is enabled

TBWC[c]_EN_EIRQ	
Description	Enable TBWC comparator [c] interrupt source for CMP_EIRQ line

TBWC[c]_EN_EIRQ	
Loop	$c = \{n : 0 <= n <= 11\}$
Bit Range	$[c + 12 : c + 12]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$c < \min(8 * \mathrm{NTOM}, 12)$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : interrupt source of TBWC comparator [c] is disabled 1 : interrupt source of TBWC comparator [c] is enabled

25.7 CMP Port Description

25.7.1 CMP signal interface

Loop	–
Condition	–
Format	–

TOM0	
Description	Output signals from TOM0
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	15 DOWNTO 0
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TOM1	
Description	Output signals from TOM1
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	15 DOWNTO 0
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET

TOM1	
Reset Value	–

ATOM0	
Description	Output signals from ATOM0
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

ATOM1	
Description	Output signals from ATOM1
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

ATOM2	
Description	Output signals from ATOM2
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

25.7.2 CMP interrupt interface

Loop	–
Condition	–
Format	–

CMP_IRQ	
Description	CMP IRQ line signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CMP_EIRQ	
Description	CMP error IRQ line signal
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

25.7.3 CMP ERR interface

Loop	–
Condition	–
Format	–

CMP_ERR	
Description	CMP error line to MON
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	OUT
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

25.7.4 CMP ABWC Comparator Input 1

Loop	–
Condition	–

Format	–

CDTM0_DTM4_CH0_OUT	
Description	ABWC0 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM4_CH2_OUT	
Description	ABWC1 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM5_CH0_OUT	
Description	ABWC2 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM5_CH2_OUT	
Description	ABWC3 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–

CDTM0_DTM5_CH2_OUT	
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM4_CH0_OUT	
Description	ABWC4 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM4_CH2_OUT	
Description	ABWC5 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM5_CH0_OUT	
Description	ABWC6 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM5_CH2_OUT	
Description	ABWC7 Comparator Input 1
Loop	–
Condition	–

CDTM1_DTM5_CH2_OUT	
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM2_DTM4_CH0_OUT	
Description	ABWC8 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM2_DTM4_CH2_OUT	
Description	ABWC9 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM2_DTM5_CH0_OUT	
Description	ABWC10 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM2_DTM5_CH2_OUT	
Description	ABWC11 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

25.7.5 CMP ABWC Comparator Input 2

Loop	–
Condition	–
Format	–

CDTM0_DTM4_CH1_OUT	
Description	ABWC0 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM4_CH3_OUT	
Description	ABWC1 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM5_CH1_OUT	
Description	ABWC2 Comparator Input 2
Loop	–
Condition	–

CDTM0_DTM5_CH1_OUT	
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM5_CH3_OUT	
Description	ABWC3 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM4_CH1_OUT	
Description	ABWC4 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM4_CH3_OUT	
Description	ABWC5 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM5_CH1_OUT	
Description	ABWC6 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM5_CH3_OUT	
Description	ABWC7 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM2_DTM4_CH1_OUT	
Description	ABWC8 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM2_DTM4_CH3_OUT	
Description	ABWC9 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–

CDTM2_DTM4_CH3_OUT	
Operational Reset	GTM_RESET
Reset Value	–

CDTM2_DTM5_CH1_OUT	
Description	ABWC10 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM2_DTM5_CH3_OUT	
Description	ABWC11 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

25.7.6 CMP TBWC Comparator Input 1

Loop	–
Condition	–
Format	–

CDTM0_DTM0_CH0_OUT	
Description	TBWC0 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM0_CH2_OUT	
Description	TBWC1 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM1_CH0_OUT	
Description	TBWC2 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM1_CH2_OUT	
Description	TBWC3 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM2_CH0_OUT	
Description	TBWC4 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–

CDTM0_DTM2_CH0_OUT	
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM2_CH2_OUT	
Description	TBWC5 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM3_CH0_OUT	
Description	TBWC6 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM3_CH2_OUT	
Description	TBWC7 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM0_CH0_OUT	
Description	TBWC8 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–

CDTM1_DTM0_CH0_OUT	
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM0_CH2_OUT	
Description	TBWC9 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM1_CH0_OUT	
Description	TBWC10 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM1_CH2_OUT	
Description	TBWC11 Comparator Input 1
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

25.7.7 CMP TBWC Comparator Input 2

Loop	–
Condition	–
Format	–

CDTM0_DTM0_CH1_OUT	
Description	TBWC0 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM0_CH3_OUT	
Description	TBWC1 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM1_CH1_OUT	
Description	TBWC2 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM1_CH3_OUT	
Description	TBWC3 Comparator Input 2
Loop	–
Condition	–
Logical Name	–

CDTM0_DTM1_CH3_OUT	
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM2_CH1_OUT	
Description	TBWC4 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM2_CH3_OUT	
Description	TBWC5 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM3_CH1_OUT	
Description	TBWC6 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM0_DTM3_CH3_OUT	
Description	TBWC7 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM0_CH1_OUT	
Description	TBWC8 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM0_CH3_OUT	
Description	TBWC9 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM1_CH1_OUT	
Description	TBWC10 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–

CDTM1_DTM1_CH1_OUT	
Operational Reset	GTM_RESET
Reset Value	–

CDTM1_DTM1_CH3_OUT	
Description	TBWC11 Comparator Input 2
Loop	–
Condition	–
Logical Name	–
Port Type	–
Port Direction	–
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

26 Monitor Unit (MON)

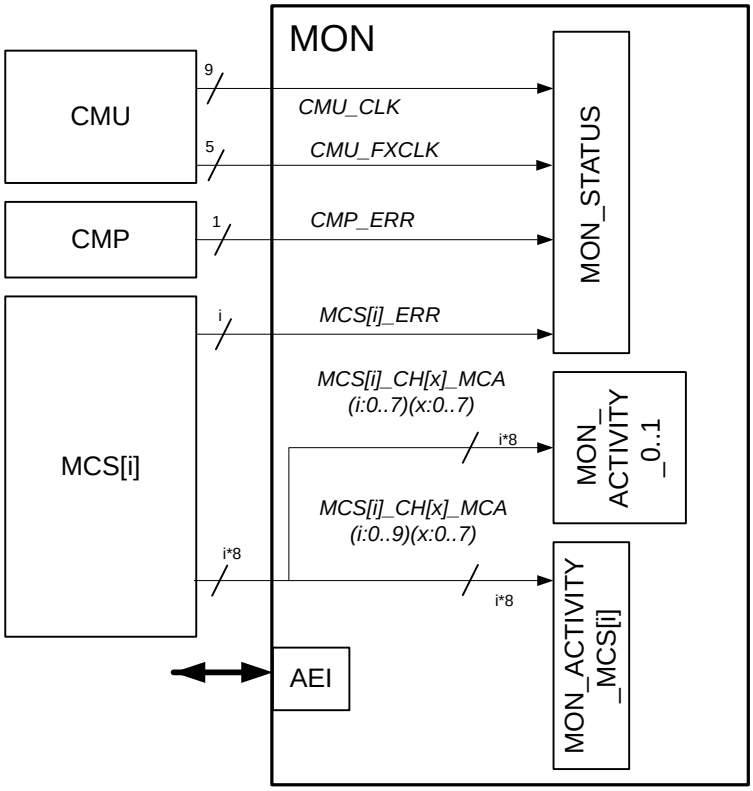
26.1 Overview

The Monitor Unit (MON) is designed for the use in safety relevant applications. The main idea is to have a possibility to supervise commonly used circuitry and resources. In this way the activity of the clocks is supervised. In addition, the characteristics of output signals can be checked in a MCS channel by a re-read-in via TIM and routing to the MCS. When the comparison fails an error signal is generated in MCS and sent to the monitor unit. One error signal per MCS summarizes the errors of all channels. By generation of an activity signal per channel for each such performed comparison, the activity of TIM, ARU and the used clocks is checked implicitly.
In addition, the ARU cycle time could also be compared in a MCS channel to given values.

Indices and their range as used inside this chapter are:

- x:={0, 1,..., 7} channel index of TOM, ATOM, MCS channels
- y:={0, 1,..., 4} index for fixed clock for TOM FXCLK(y)
- z:={0, 1} index for external clocks ECLK(z)
- j:={0, 1,..., NMCS-1} instance index of cluster / MCS module

Figure 164 MON Block Diagram



26.1.1 Realization without Activity Checker of the Clock Signals

An activity checker of the clock signals used is not needed because these signals are only enables to be used in combination with the system clock. Therefore, the clock enables are to be checked to have a high value.

26.2 Clock Monitoring

The monitor unit has a connection to each of the 9 clocks *CCM[1]_CLK_RES* , provided by the CMU. Some of these clocks can be used for special tasks (see chapter *10 "Clock Management Unit (CMU)"*).

In addition, the 5 clock inputs of the TOMs *CCM[1]_FXCLK_RES* are also connected to the MON unit.

The supervising of the clocks is done by scanning for activity of each clock.

A high value is defined as the state to be monitored.

When a high value of the clock enable is detected, the corresponding bit in the status register **MON_STATUS** is set.

The status register bits are reset by writing a one.

When the register is polled by the CPU and the time between two read accesses is higher than the period of the slowest clock, all bits of the corresponding clocks must have been set.

When polling in shorter time distances, not for all clocks an activity can be shown, although they are still working.

Because of the realization without a select register for the clock signals only the bits of the status register are to be considered for which the clock signal is enabled in the CMU.

26.3 CMP Error Monitoring

The signal **MON_STATUS.CMP_ERR** is to be received directly from module CMP and is set if an error occurred.

26.4 Checking the Characteristics of Signals by MCS

By use of the MCS some given properties of signals can be checked. Such signals can be generated output signals of TOM or ATOM channels including DTM function, which are reread in into a TIM and the time stamp information is routed via ARU to the MCS module.

The corresponding MCS signal performs the check according to given properties. In this way signal high or low time as well as signal periods can be checked, also taking into account tolerances. When the check fails a MCS internal error signal is generated and ORed with the error signals of the other channels of the MCS module to a summarized error signal *MON_MCS[j]_ERR* .

For each MCS a summarized error signal is transmitted to MON and monitored in the **MON_STATUS** register.

In order to check the execution of the comparison for each MCS channel an activity signal is generated. In the *MON_MCS[j]_CH_MCA* vector 8 bits for each channel of MCS instance j exist. The activity signals are stored in the **MON_ACTIVITY_MCS[j]** register. In addition, the first 8 bits of MCS[j] (j∈{0, 1,..., 3}) are stored in **MON_ACTIVITY_0** and the first 8 bits of MCS[j] (j∈{4, 5,..., 7}) are stored in **MON_ACTIVITY_1** . The bits are set by a one signal and reset by writing a one to it (preferably after polling the status of the register).

Because the activity signal shows the execution of a comparison, the involved units for providing the signals and execution of comparison (like TIM, ARU and MCS itself) are checked implicitly to work accordingly. Also, the involved clocks and time bases are checked in this way.

26.5 Checking ARU Cycle Time

The cycle time of the ARU can be checked, when this is essential for safety purposes. This check can be performed by an MCS channel.

Note: The MCS program for measuring the ARU round trip time must add a tolerance value.

The resulting error is reported to the MON unit using the summarized error signal *MON_MCS[j]_ERR* for each MCS module in addition to an interrupt, generated in MCS. The same signals and status bits are used as in the case of checking the signal characteristics.

The corresponding MCS is programmed to get a fixed data value at address 0x1FF. The data value is always zero and is not blocked. When getting the access the time stamp value *MON_TBU_TS0* is stored in a register. The next time getting the access the new *MON_TBU_TS0* value is stored and the difference between both values is compared with a given value. When the comparison fails, an error flag is set in the MCS internal status register, an interrupt is generated and the error signal *MON_MCS[j]_ERR* is provided.

When the check is performed, an activity signal *MON_MCS[j]_CH_MCA* [x:x] is provided for each channel x for each MCS[j] instance together with a summarized interrupt *MON_MCS[j]_ERR* for each MCS.

The activity signal sets a bit in the corresponding **MON_ACTIVITY_0** , **MON_ACTIVITY_1** , **MON_ACTIVITY_MCS[j]** registers.

The bits in the **MON_ACTIVITY_0** , **MON_ACTIVITY_1** , **MON_ACTIVITY_MCS[j]** registers are reset by writing a one.

When the check fails, an interrupt is generated and the error signal *MON_MCS[j]_ERR* is provided for the MON unit.

Figure *164 "MON Block Diagram"* shows the block diagram of the Monitor Unit.

26.6 MON Interrupt Signals

The MON submodule has no interrupt signals.

26.7 MON Configuration Registers Description

26.7.1 MON_STATUS

Description	MON status register
Loop	
Condition	$\mathrm{NMON} > 0$

Storage Type	REGISTER
Clock Active Cond	$\mathrm{MON\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MON_STATUS
Address	$0x20680$
C-Name	GTM.CLS[1].MON.STATUS

Interface: MCS[i]

Name	MON_STATUS
Address	$0x680$
C-Name	

ACT_CMU[x]	
Description	CCM[1]_CLK_RES activity
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No activity detected for *CCM[1]_CLK_RES* [x:x] 1 : Activity detected for *CCM[1]_CLK_RES* [x:x]
W-Coding	0 : No action 1 : Clear activity for *CCM[1]_CLK_RES* [x:x]
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

ACT_CMUFX[y]	
Description	CCM[1]_FXCLK_RES activity
Loop	$y = \{n : 0 <= n <= 4\}$
Bit Range	$[y + 8 : y + 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$\mathrm{NTOM} > 0$

ACT_CMUFX[y]	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No activity detected for *CCM[1]_FXCLK_RES* [y:y] 1 : Activity detected for *CCM[1]_FXCLK_RES* [y:y]
W-Coding	0 : No action 1 : Clear activity for *CCM[1]_FXCLK_RES* [y:y]
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

ACT_CMU8	
Description	CCM[1]_CLK_RES[8:8] activity
Loop	–
Bit Range	[14 : 14]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	1
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No activity detected for *CCM[1]_CLK_RES* [8:8] 1 : Activity detected for *CCM[1]_CLK_RES* [8:8]
W-Coding	0 : No action 1 : Clear activity for *CCM[1]_CLK_RES* [8:8]
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged. **Note:** Bit is set, when a rising edge is detected at the considered clock.

CMP_ERR	
Description	Error detected at CMP
Loop	–
Bit Range	[16 : 16]
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{NCMP} > 0$
Initial value	0

CMP_ERR	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No error detected 1 : Error detected
	Note: This bit will be readable only.

MCS[j]_ERR	
Description	Error detected at MCS[j]
Loop	$j = \{n : 0 <= n <= \text{NMCS} - 1\}$
Bit Range	$[j + 20 : j + 20]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	$\text{NMCS} > 0$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No error detected 1 : Error detected

Note:
Bits [12:0] are set, when a rising edge is detected at the considered clock.

Note:
Bits [16:16] and [29:20] are set, when the corresponding unit reports an error.

Note:
Example for MCS error: The MCS can be programmed to generate an error, when the comparison of signal values (duty time, cycle time) fails or also when the cycle time of the ARU (checking of the *MON_TBU_TS0* between two periodic accesses) is out of the expected range.

26.7.2 MON_ACTIVITY_0

Description	MON activity register 0
Loop	
Condition	$\text{NMCS} > 0$
Storage Type	REGISTER
Clock Active Cond	$\text{MON\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MON_ACTIVITY_0
Address	$0x20684$
C-Name	GTM.CLS[1].MON.ACTIVITY_0

Interface: MCS[i]

Name	MON_ACTIVITY_0
Address	$0x684$
C-Name	

MCA_0_[x]	
Description	Activity of check performed in module MCS0 at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No activity detected 1 : Activity detected
W-Coding	0 : No action 1 : Clear activity
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

MCA_1_[x]	
Description	Activity of check performed in module MCS1 at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[8 + x : 8 + x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$\mathrm{NMCS} > 1$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No activity detected 1 : Activity detected
W-Coding	0 : No action 1 : Clear activity

MCA_1_[x]	
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

MCA_2_[x]	
Description	Activity of check performed in module MCS2 at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[16 + x : 16 + x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$\mathrm{NMCS} > 2$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No activity detected 1 : Activity detected
W–Coding	0 : No action 1 : Clear activity
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

MCA_3_[x]	
Description	Activity of check performed in module MCS3 at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[24 + x : 24 + x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$\mathrm{NMCS} > 3$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No activity detected 1 : Activity detected
W–Coding	0 : No action 1 : Clear activity

MCA_3_[x]	
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

Note:
When not all MCS modules are implemented or the channels are not used for check purposes with supervising, the corresponding activity bits remain zero.

26.7.3 MON_ACTIVITY_1

Description	MON activity register 1
Loop	
Condition	$\mathrm{NMCS} > 4$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MON\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MON_ACTIVITY_1
Address	$0x20688$
C-Name	GTM.CLS[1].MON.ACTIVITY_1

Interface: MCS[i]

Name	MON_ACTIVITY_1
Address	$0x688$
C-Name	

MCA_4_[x]	
Description	Activity of check performed in module MCS4 at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No activity detected 1 : Activity detected
W-Coding	0 : No action 1 : Clear activity

MCA_4_[x]	
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

MCA_5_[x]	
Description	Activity of check performed in module MCS5 at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 8 : x + 8]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$\mathrm{NMCS} > 5$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No activity detected 1 : Activity detected
W–Coding	0 : No action 1 : Clear activity
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

MCA_6_[x]	
Description	Activity of check performed in module MCS6 at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 16 : x + 16]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$\mathrm{NMCS} > 6$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No activity detected 1 : Activity detected
W–Coding	0 : No action 1 : Clear activity

MCA_6_[x]	
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

MCA_7_[x]	
Description	Activity of check performed in module MCS7 at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x + 24 : x + 24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$\mathrm{NMCS} > 7$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No activity detected 1 : Activity detected
W-Coding	0 : No action 1 : Clear activity
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

Note:
When not all MCS modules are implemented or the channels are not used for check purposes with supervising, the corresponding activity bits remain zero.

26.7.4 MON_ACTIVITY_MCS[j]

Description	MON activity register for MCS [j]
Loop	$j = \{n : 0 <= n <= \mathrm{NMCS} - 1\}$
Condition	$j < \mathrm{NMCS}$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{MON\_CLK\_ENABLE} == 1$

Interface: CPU

Name	MON_ACTIVITY_MCS[j]
Address	$0x4 * j + 0x2068C$
C-Name	GTM.CLS[1].MON.ACTIVITY_MCS[j]

Interface: MCS[i]

Name	MON_ACTIVITY_MCS[j]

Address	$(((0x4 * j) + 0x2068C) - 0x20000)$
C-Name	

MCA_[x]	
Description	Activity of check performed in module MCS [j] at channel [x]
Loop	$x = \{n : 0 <= n <= 7\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No activity detected 1 : Activity detected
W-Coding	0 : No action 1 : Clear activity
	Note: This bit will be cleared on a CPU write access of value 1. A read access leaves the bit unchanged.

Note:
Unused MCA bits are reserved

26.8 MON Port Description

26.8.1 MON Ports

Loop	-
Condition	-
Format	-

MON_MCS[j]_CH_MCA	
Description	MON Activity signaling for MCS, see register STA.MCA in MCS
Loop	-
Condition	-
Logical Name	-
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[1]_CLK
Special Function	-
Operational Reset	GTM_RESET
Reset Value	-

MON_MCS[j]_ERR	
Description	Error line from MCS
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC
Port Direction	IN
Port Range	–
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

MON_TBU_TS0	
Description	Time stamp 0 from TBU, not in MON
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[1]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

27 Timer Input Output (TIO)

27.1 Features

The TIO_Basic supports general purpose input /output functionality for low-end applications with reduced design complexity and function requirements compared with the GTM TIM / ATOM / TOM modules.
It allows a flexible usage of the functional resources as input or output on a channel granularity configurable at runtime.

o Module supports 8, 16, 24 channels

o Trigger generation / selection

- Generate trigger on rising and / or falling edge of input signal
- Generate trigger on rising and / or falling edge of output signal
- Select trigger signal of previous channel

o Configurable update resolution

- Choose update resolution out of existing *CCM[i]_CLK_RES* sources
- Update definable by software write access
- Update definable by HW events (see trigger generation / selection)
- Update definable by trigger generation of previous channel

o Mechanism for synchronous operation of multiple channels

- An atomic read / write operation initiated by a programmable core is able to read / change the status of up to 24 channels in the same clock cycle
- Synchronous read of value of up to 24 inputs
- Synchronous sampling of state of up to 24 channels
- Set / change output value of up to 24 channels synchronously
- Start / stop operation of up to 24 channels synchronously

o Input capture

- Read current input value captured with *CLS[i]_CLK*
- Input sampling on defined resolution

o Output signal generation

- Asynchronous update: set / clear output bits at any time
- Synchronous update: set / clear output bits on selected resolution
- Define output polarity / invert output

o Simultaneous input / output functionality of one channel

- Reading the actual input value and set / clear of the output is possible

o Signal multiplex

- Select 1 out of 24 sources
- Distribute 1 source to 24 outputs

o User definable Boolean functions of input and output signals

- Logic function of 2 signals
- Logic function of 3 signals

o Interrupt generation

- Use triggers as interrupts

The TIO_Basic functionality does not provide any hardware resources to process input or output signals in terms of timed signal generation, timed signal measurement or time based capture. These kind of functions can be provided with the TIO_Basic under assistance of a software program executed on a programmable core.
A list of typical application examples are described in chapter *27.6.1 "TIO Applications"* .

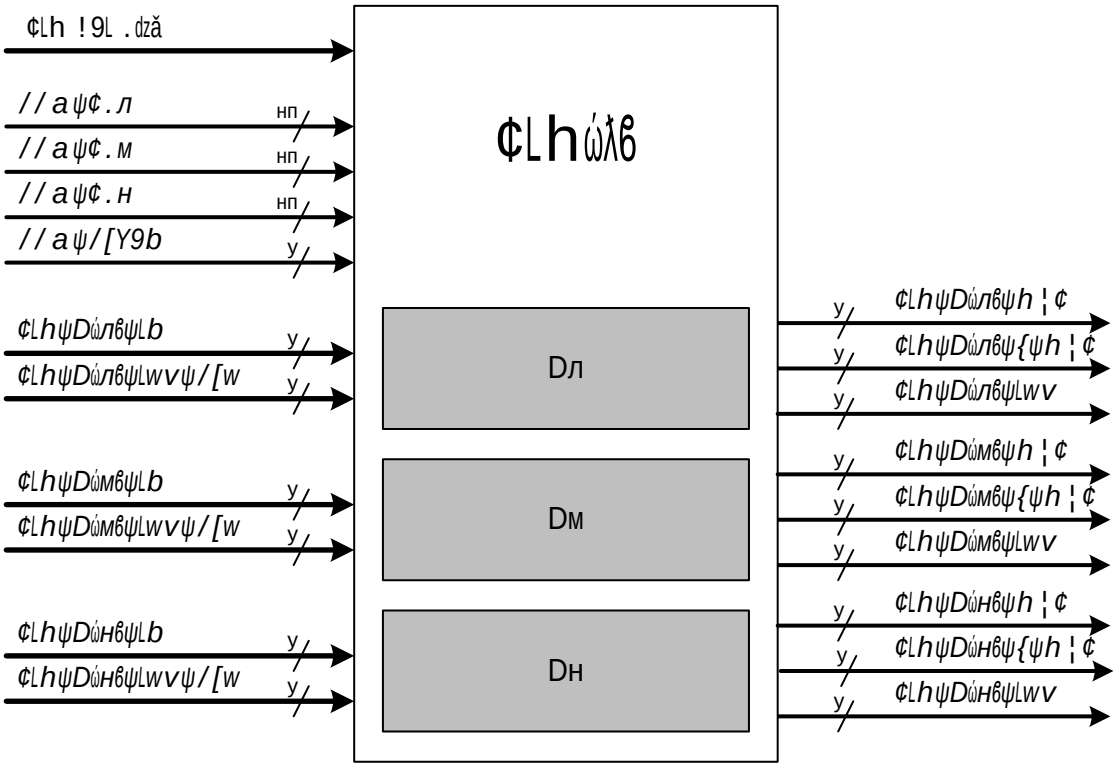
27.1.1 TIO_PL Extension

An additional TIO Plus extension exists to the TIO_Basic which adds hardware resources to support autonomous timed signal generation, signal measurement and time-based capture. Features and functions related to the TIO Plus will be covered in separate chapter *27.5 "TIO Plus Functional Description"* .

27.2 Block Diagram

The TIO is structured in a way that a scalability for 8, 16, 24 channels is supported.

Figure 165 TIO Module



Following scalability parameters control the existence of TIO modules in a GTM device:

- NCCM∈{0, 1,..., 11}: defines the number of clusters which could include a TIO module
- CTIO[i]:
- 0: no TIO available in cluster i
- 1: TIO available in cluster i

Scalability is controlled by the following parameters:

o NTIO_CH8:

- 1: TIO includes 8 channels
- 2: TIO includes 16 channels
- 3: TIO includes 24 channels

o TIO_PLUS:

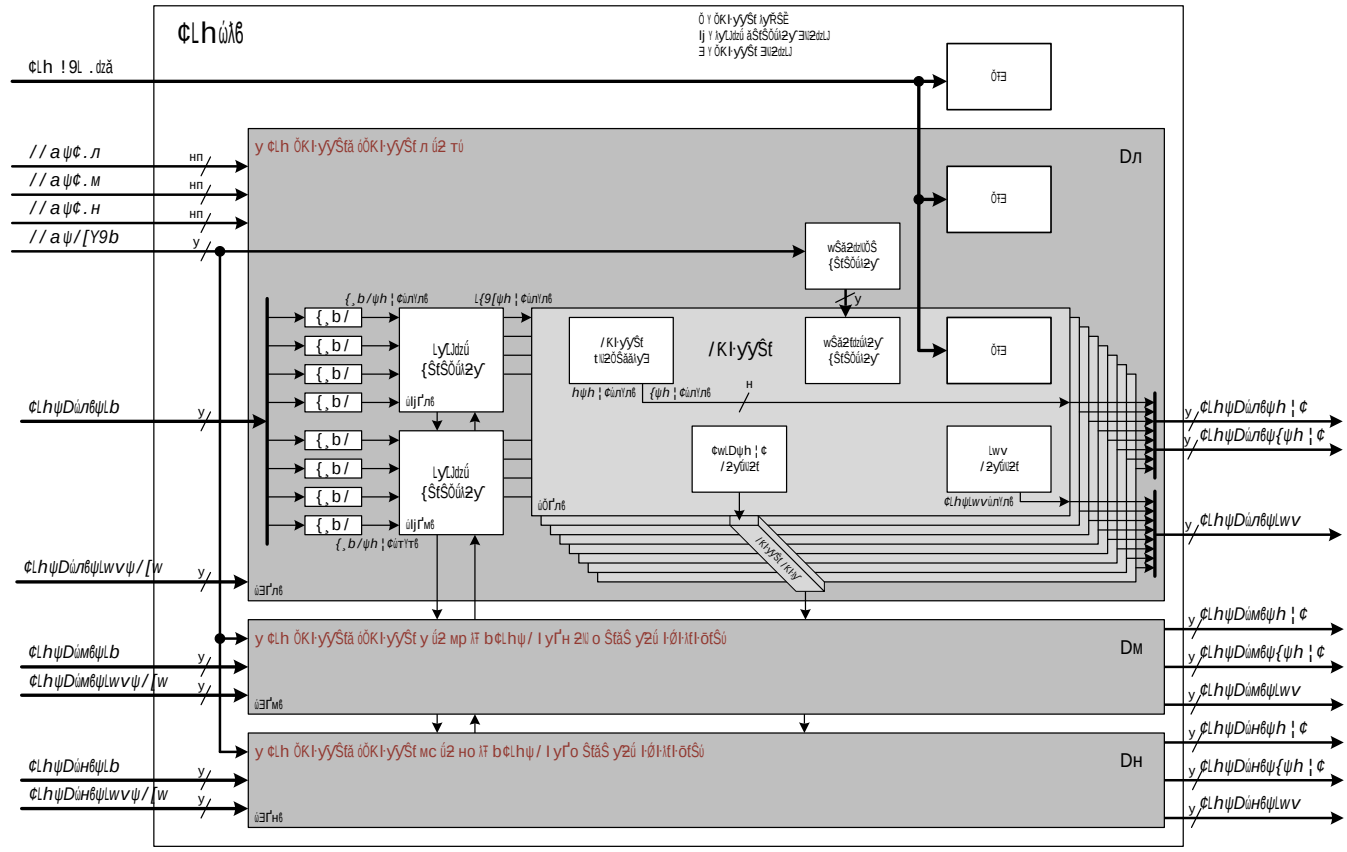
- 0: TIO does not support TIO_PL functionality
- 1: TIO supports TIO_PL functionality

An overview of the functional blocks is shown in figure *166 "TIO Block Diagram"* .

Following index definitions will be used:

- i∈{0, 1,..., 11} index of Custer/TIO instance
- g∈{0, 1,..., (NTIO_CH8−1)} channel index channel group (8 channels)
- c∈{0, 1,..., 7} index of channel in channel group
- q∈{0, 1} index of input selection group
- k∈{0, 1, 2, 3} index used in input selection group
- u: Context specific index (defined locally)
- x∈{0, 1,..., (NTIO_CH8 *8)−1} index of the TIO channel

Figure 166 TIO Block Diagram



The TIO module can support 8 / 16 / 24 channels. Depending on the configuration parameter NTIO_CH8 ∈ {1, 2, 3} channel groups (dark grey boxes) are available. Inside a channel group the following functional blocks exist:

- 8 SYNC: input synchronization block for each of the 8 input signals
- 2 Input Selection: configurable multiplex / duplicate of 4 inputs to 4 outputs
- 1 Resource Selection: configure usable update resolution sources for a channel group
- 8 Channel: configurable block for individual processing of 1 input and/or 1 output

In a channel resource (light grey box) following functions exist:

- 1 Resolution Selection: choose an update resolution for operation of this channel
- 1 Channel Processing: configure mode of operation of this channel
- 1 IRQ Control: configure which events of the channel shall trigger an IRQ to the CPU / GTM−MCS
- 1 Trigger Output Control / trigger generation: configure which events of the channel shall be used to trigger channel internal actions or trigger following channels via the channel chain

With the configuration interface TIO_AEI read / write operations from the CPU / GTM−MCS to the TIO module are possible. All blocks named "cfg" in the block diagram indicate, where configurability via configuration registers exists. In all block diagrams configuration signals will be indicated with the prefix "cfg" (e.g.: cfg.LUT3 instead of **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3**).

27.3 Hardware Interface

The TIO inputs / TIO outputs can be used for connections "EXTERNAL" to the SOC where the GTM is integrated.
Following signals can be used as external interfaces (SOC IO). NTIO_CH8 ∈ {1, 2, 3} can be selected.

Table 58 Pin List

Pin	Name	I/O	Description
TIO_G[0]_IN [7:0]	TIO input signals	I	Available with NTIO_CH8 >= 1
TIO_G[0]_OUT [7:0]	TIO output signals	O	Available with NTIO_CH8 >= 1
TIO_G[1]_IN [7:0]	TIO input signals	I	Available with NTIO_CH8 >= 2
TIO_G[1]_OUT [7:0]	TIO output signals	O	Available with NTIO_CH8 >= 2
TIO_G[2]_IN [7:0]	TIO input signals	I	Available with NTIO_CH8 >= 3
TIO_G[2]_OUT [7:0]	TIO output signals	O	Available with NTIO_CH8 >= 3

In addition, the TIO inputs / TIO outputs can be used as read / control signals inside the SOC. This functionality has to be defined by the GTM integrator.
It is even possible to use TIO inputs / TIO outputs to connect to bidirectional SOC IO–Ports. The output enable and the direction control is under the responsibility of the integrator of the GTM in the SOC.

27.4 TIO Functional Description

Following chapters describe the functional blocks of the TIO.

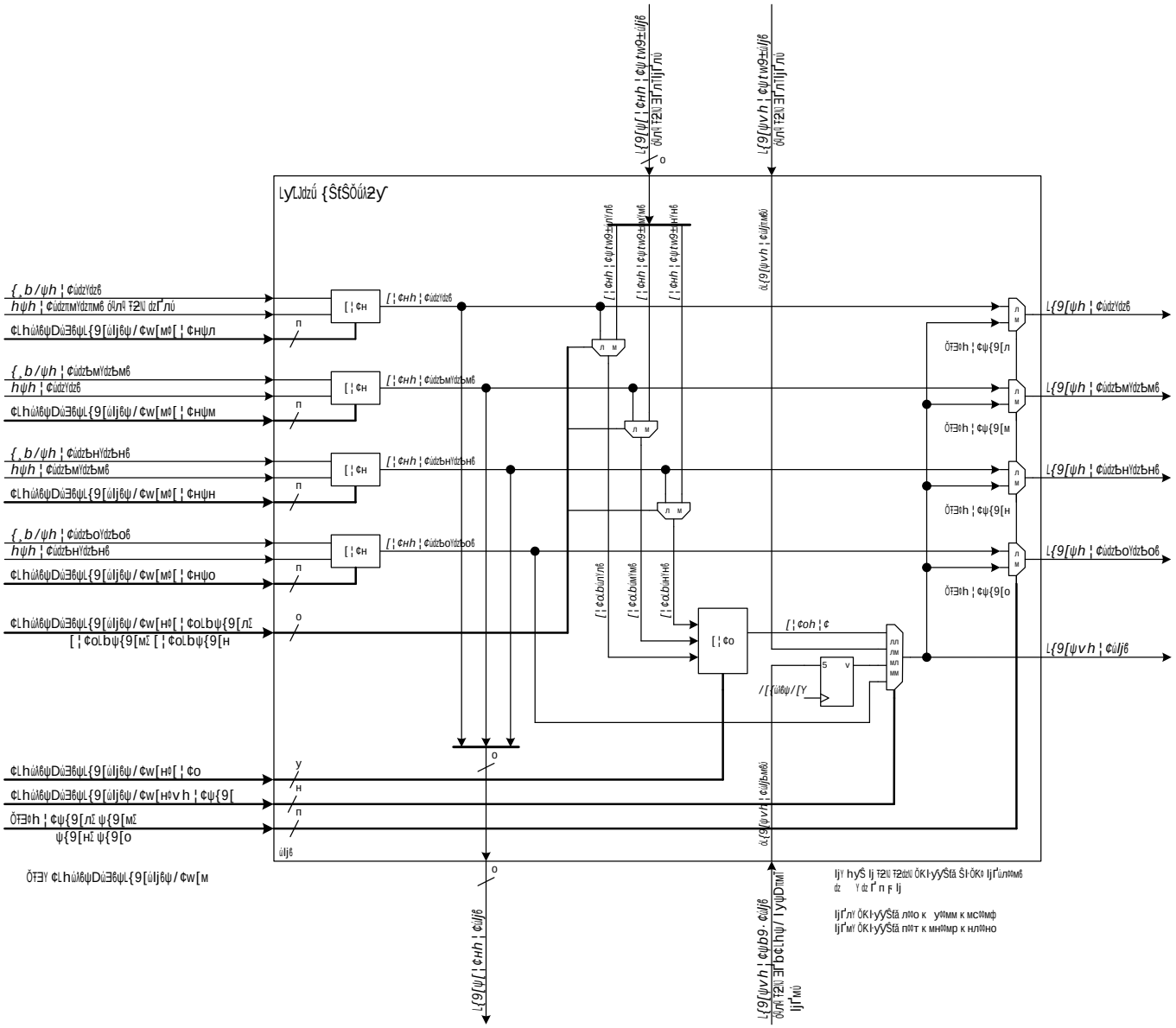
27.4.1 TIO Input Synchronizing

Every input *TIO_G[g]_IN* [c:c] will be synchronized with the TIO system clock by a 2–stage register. The synchronized signal is named as *SYNC_OUT* [c:c].

27.4.2 TIO Input Selection

Every channel group g has two input selection blocks (see Figure REF{TIO_900}). The first input selection block is able to generate 4 signals *ISEL_OUT* [0:0], *ISEL_OUT* [1:1], *ISEL_OUT* [2:2], *ISEL_OUT* [3:3] providing input data for the TIO channel 0 to 3. The second input selection block is able to generate 4 signals *ISEL_OUT* [4:4], *ISEL_OUT* [5:5], *ISEL_OUT* [6:6], *ISEL_OUT* [7:7] providing input data for the TIO channel 4 to 7. A block diagram of an input selection block is depicted in Figure *167 "Input Selection"* .
The 4 signals *ISEL_QOUT[q]* , *ISEL_LUT2OUT[q]* [2:2], *ISEL_LUT2OUT[q]* [1:1], *ISEL_LUT2OUT[q]* [0:0] are used to forward data to the neighboring input selection blocks.

Figure 167 Input Selection



The input selection block covers 4 neighbored input channels. It can be configured to support TIO input *SYNC_OUT* [c:c] and output of previous channels *O_OUT* [c−1:c−1] signal manipulation, such as:

- Signal multiplexing (TIO input *SYNC_OUT* [c:c] or output of previous channel *O_OUT* [c−1:c−1])
- Signal duplication
- Signal composition by usage of user definable logic functions
- Signal propagation to neighbored channels and neighbored input selection blocks

Signal composition for one internal channel k of channel group q

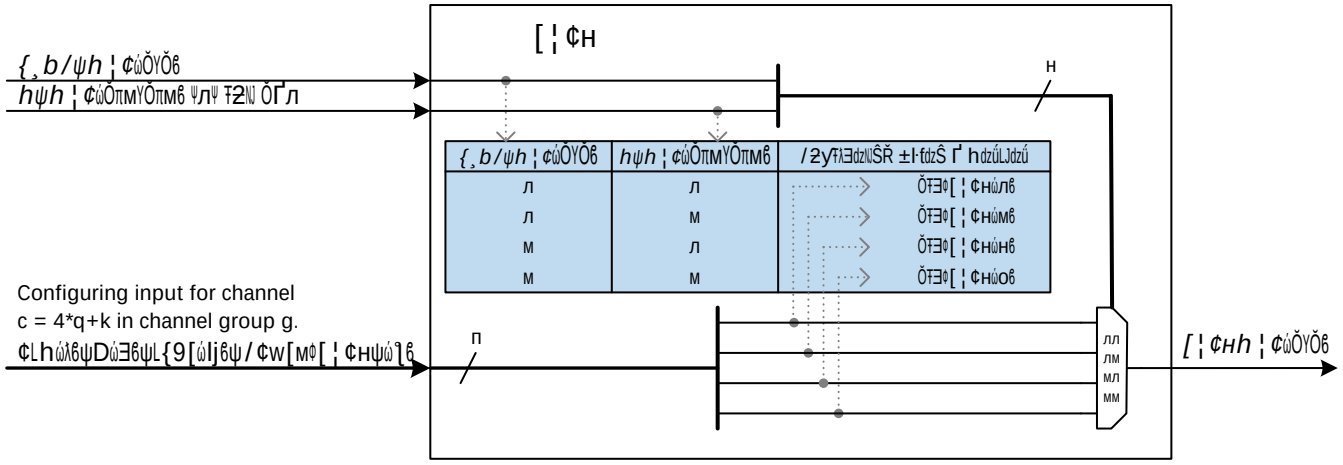
If control bit **TIO[i]_G[g]_ISEL[q]_CTRL1.OUT_SEL[k]** = 0, the LUT2 function is in use:
ISEL_OUT [q*4+k:q*4+k] = *LUT2OUT* [q*4+k:q*4+k]

LUT2OUT [q*4+k:q*4+k] signal generation with LUT2 function:

For each individual channel c = q*4+k in channel group g, it can be configured to apply any logic function of the 2 inputs *SYNC_OUT* [c:c] and *O_OUT* [c−1:c−1].

The signal *LUT2OUT* [c:c] is defined with the bit field **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]**

Figure 168 Lut2 Function



Definition: Lut2 (Look up table 2 inputs); the lookup table operates as a multiplexer, where the value of the 2 input signals (*SYNC_OUT* [c:c], *O_OUT* [c−1:c−1]) define an index which will be used to select the corresponding value stored in the array **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** .

Example:

- In case of *SYNC_OUT* [c:c]=0 and *O_OUT* [c−1:c−1]=0 the index = 0b00 is generated by concatenation of the 2 values. The index=0 will select the lookup table array element cfg.LUT2[index:index] = cfg.LUT2[0:0]. This value will be assigned to the signal *LUT2OUT* [c:c].
- In case of *SYNC_OUT* [c:c]=1 and *O_OUT* [c−1:c−1]=0 the index = 0b10 is generated by concatenation of the 2 values. The index=2 will select the lookup table array element cfg.LUT2[index:index] = cfg.LUT2[2:2]. This value will be assigned to the signal *LUT2OUT* [c:c].

In the lookup table 4 values are stored, each represents the output value associated with one input value combination of the 2 input signals.

This allows applications to use the local input signal *SYNC_OUT* [c:c] or the output signal of the previous stage *O_OUT* [c−1:c−1] or any Boolean function of both.

Examples:

- Lut2 function: *LUT2OUT* [c:c] = *SYNC_OUT* [c:c]; **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** = 0b1100
- Lut2 function: *LUT2OUT* [c:c] = *O_OUT* [c−1:c−1]; **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** = 0b1010
- Lut2 function: *LUT2OUT* [c:c] = *SYNC_OUT* [c:c] and *O_OUT* [c−1:c−1]; **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** = 0b1000
- Lut2 function: *LUT2OUT* [c:c] = *SYNC_OUT* [c:c] or *O_OUT* [c−1:c−1]; **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** = 0b1110

If control bit **TIO[i]_G[g]_ISEL[q]_CTRL1.OUT_SEL[k]** = 1; (signal cfg.OUT_SEL[k] in figure *167 "Input Selection"*):
ISEL_OUT [q*4+k:q*4+k] = *ISEL_QOUT[q]*

Selection of signal *ISEL_QOUT[q]* :

With the control bit **TIO[i]_G[g]_ISEL[q]_CTRL2.QOUT_SEL** the source for the signal *ISEL_QOUT[q]* can be selected; (signal cfg.QOUT_SEL in figure *167 "Input Selection"*):

o **TIO[i]_G[g]_ISEL[q]_CTRL2.QOUT_SEL** = 0b00: *ISEL_QOUT[q]* is driven by *LUT3OUT* in input selection q in group g.

o **TIO[i]_G[g]_ISEL[q]_CTRL2.QOUT_SEL** = 0b01: *ISEL_QOUT[q]* is driven by *ISEL_QOUT_PREV[q]* output signal coming from the previous input selection block:

- q = 0 and g=0: 0
- q = 0 and g>0: *ISEL_QOUT[1]* of group g−1
- q = 1: *ISEL_QOUT[0]* of group g

o **TIO[i]_G[g]_ISEL[q]_CTRL2.QOUT_SEL** = 0b10: *ISEL_QOUT[q]* is driven by *ISEL_QOUT_NEXT[q]* output signal coming from next input selection block:

- q = 0: *ISEL_QOUT[1]* of group g
- q = 1: *ISEL_QOUT[0]* of group g+1

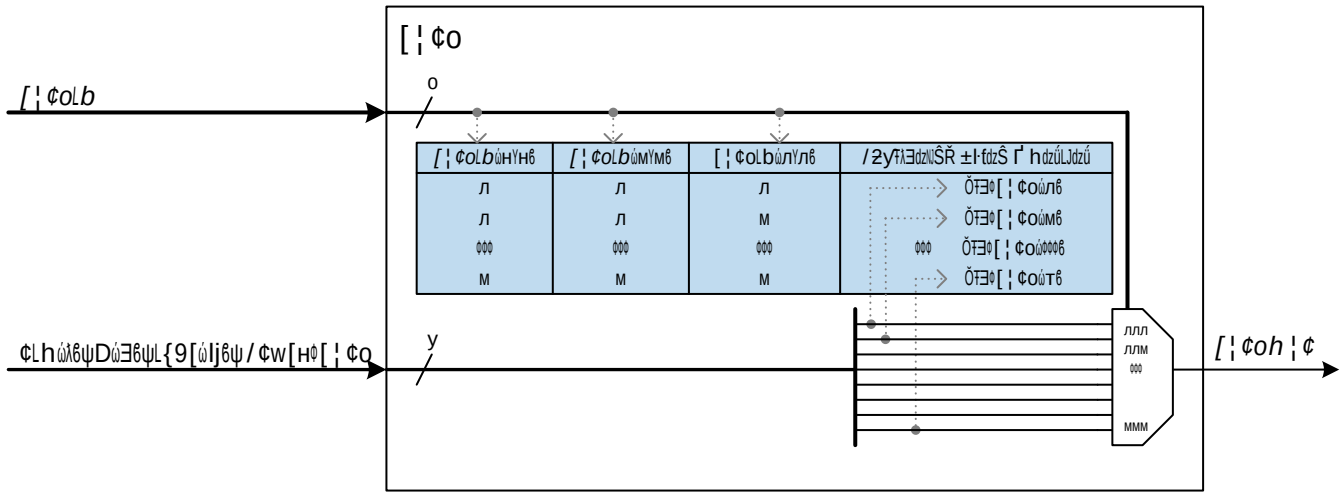
o **TIO[i]_G[g]_ISEL[q]_CTRL2.QOUT_SEL** = 0b11: *ISEL_QOUT[q]* is driven by *LUT2OUT* [u+3:u+3]

LUT3OUT signal generation with **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3** function:

The signal *LUT3OUT* in input selection instance q can be configured to apply any logic function of the 3 inputs *LUT3IN* [2:2], *LUT3IN* [1:1] and *LUT3IN* [0:0].

The signal *LUT3OUT* in the input selection instance q is defined with the bit field **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3** ; (signal cfg.LUT3 in figure *167 "Input Selection"*).

Figure 169 LUT3 Function



Definition: **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3** (lookup table 3 inputs); the lookup table operates as a multiplexer, where the value of the 3 input signals (*LUT3IN* [2:2], *LUT3IN* [1:1], *LUT3IN* [0:0]) define an index which will be used to select the corresponding value stored in the array cfg.LUT3. (see REF{TIO_946}).

Examples:

- LUT3 function: *LUT3OUT* = *LUT3IN* [2:2]; **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3** = 0b11110000
- LUT3 function: *LUT3OUT* = *LUT3IN* [0:0]; **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3** = 0b10101010
- LUT3 function: *LUT3OUT* = *LUT3IN* [2:2] and *LUT3IN* [1:1] and *LUT3IN* [0:0]; **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3** = 0b10000000
- LUT3 function: *LUT3OUT* = *LUT3IN* [2:2] or *LUT3IN* [1:1] or *LUT3IN* [0:0]; **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3** = 0b11111110

Selection of signal *LUT3IN* [k:k] k∈{0,1,2} in input selection group q

With the control bit **TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3IN_SEL[k]** 2 alternate sources can be selected for *LUT3IN* [k:k].

TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3IN_SEL[k] = 0: *LUT3IN* [k:k] = *LUT2OUT* [4*q+k:4*q+k];
TIO[i]_G[g]_ISEL[q]_CTRL2.LUT3IN_SEL[k] = 1: *LUT3IN* [k:k] = 0 for q = 0 and g=0; *LUT3IN* [k:k] = *LUT2OUT_PREV* [4*q+k:4*q+k] for q > 0 or g > 0;

Summary

Depending on the value of **TIO[i]_G[g]_ISEL[q]_CTRL2.QOUT_SEL** and **TIO[i]_G[g]_ISEL[q]_CTRL1.OUT_SEL[k]** = 1, one input signal *LUT2OUT* [c:c] of any input selection block can be selectively duplicated up to all *ISEL_OUT* [c:c] of each channel group g.
Many other multiplex / duplicate / logic function combinations of *SYNC_OUT* [c:c] and *O_OUT* [c:c] are possible.

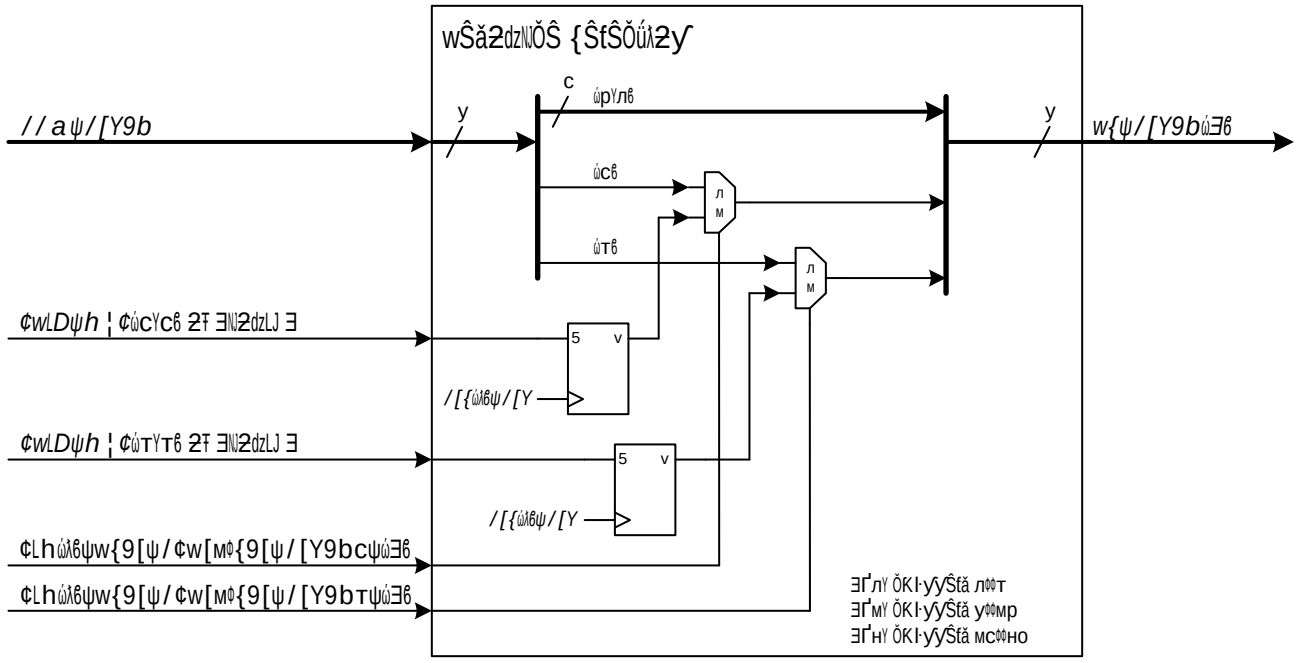
Atomic operations multi core/thread capable to configure the input selection block

The register **TIO[i]_G[g]_ISEL[q]_CTRL1** hosts the bit fields **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** , **TIO[i]_G[g]_ISEL[q]_CTRL1.OUT_SEL[k]** for all 4 channels of the corresponding input selection block. Additionally the bit fields **TIO[i]_G[g]_ISEL[q]_CTRL1.WRITE_EN[k]** exist, which implement a write protection for the bit fields **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** , **TIO[i]_G[g]_ISEL[q]_CTRL1.OUT_SEL[k]** .
A write operation to this register will only alter the state of the corresponding bit fields **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** , **TIO[i]_G[g]_ISEL[q]_CTRL1.OUT_SEL[k]** if the **TIO[i]_G[g]_ISEL[q]_CTRL1.WRITE_EN[k]** bit is 1. Otherwise, the value of bit fields **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** , **TIO[i]_G[g]_ISEL[q]_CTRL1.OUT_SEL[k]** will not change.
This allows atomic reconfiguration of a single channel without influencing others. Furthermore, atomic synchronous reconfiguration of multiple channels in one input selection block is possible.

27.4.3 TIO Resource Selection

This functional block allows to select the source for *RS_CLKEN[g]* .

Figure 170 Resource Selection



The resource selection is available for every group of 8 TIO channels:

- Group g = 0: TIO channel x (x∈{0, 1,..., 7})
- Group g = 1: TIO channel x (x∈{8, 9,..., 15})
- Group g = 2: TIO channel x (x∈{16, 17,..., 23})

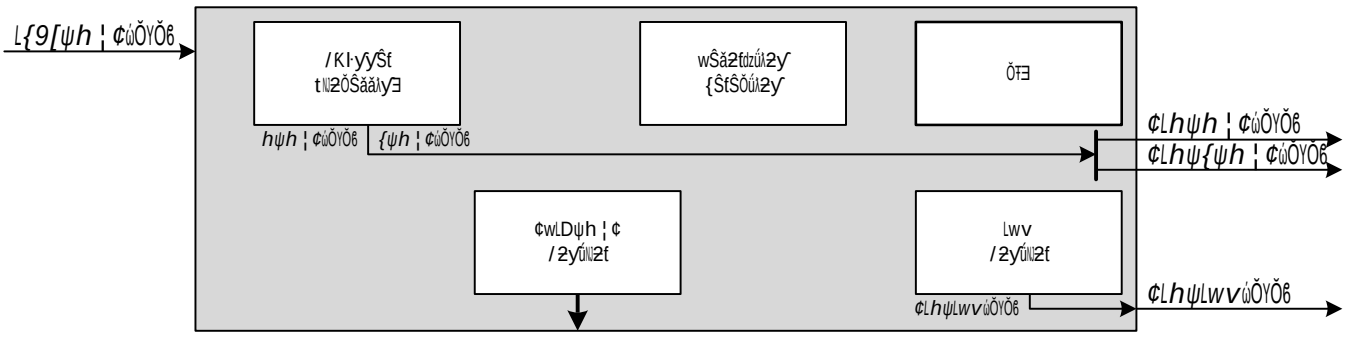
By default (register **TIO[i]_RSEL_CTRL1.SEL_CLKEN7_[g]** = 0; **TIO[i]_RSEL_CTRL1.SEL_CLKEN6_[g]** = 0) the cluster global *CCM[i]_CLK_RES* signals are in use internally by the signals *RS_CLKEN[g]* .
Usage of **TIO[i]_RSEL_CTRL1.SEL_CLKEN6_[g]** = 1 enables: *RS_CLKEN[g]* [6:6] = *TRIG_OUT* [6:6] of channel group g
Usage of **TIO[i]_RSEL_CTRL1.SEL_CLKEN7_[g]** = 1 enables: *RS_CLKEN[g]* [7:7] = *TRIG_OUT* [7:7] of channel group g

This allows to use the locally generated trigger events as local update resolution inside a group g of 8 channels.

27.4.4 TIO Channel

Each TIO channel is equipped with the functional blocks shown in figure *171 "Channel Block Diagram"* . They are explained in detail in the following chapters.

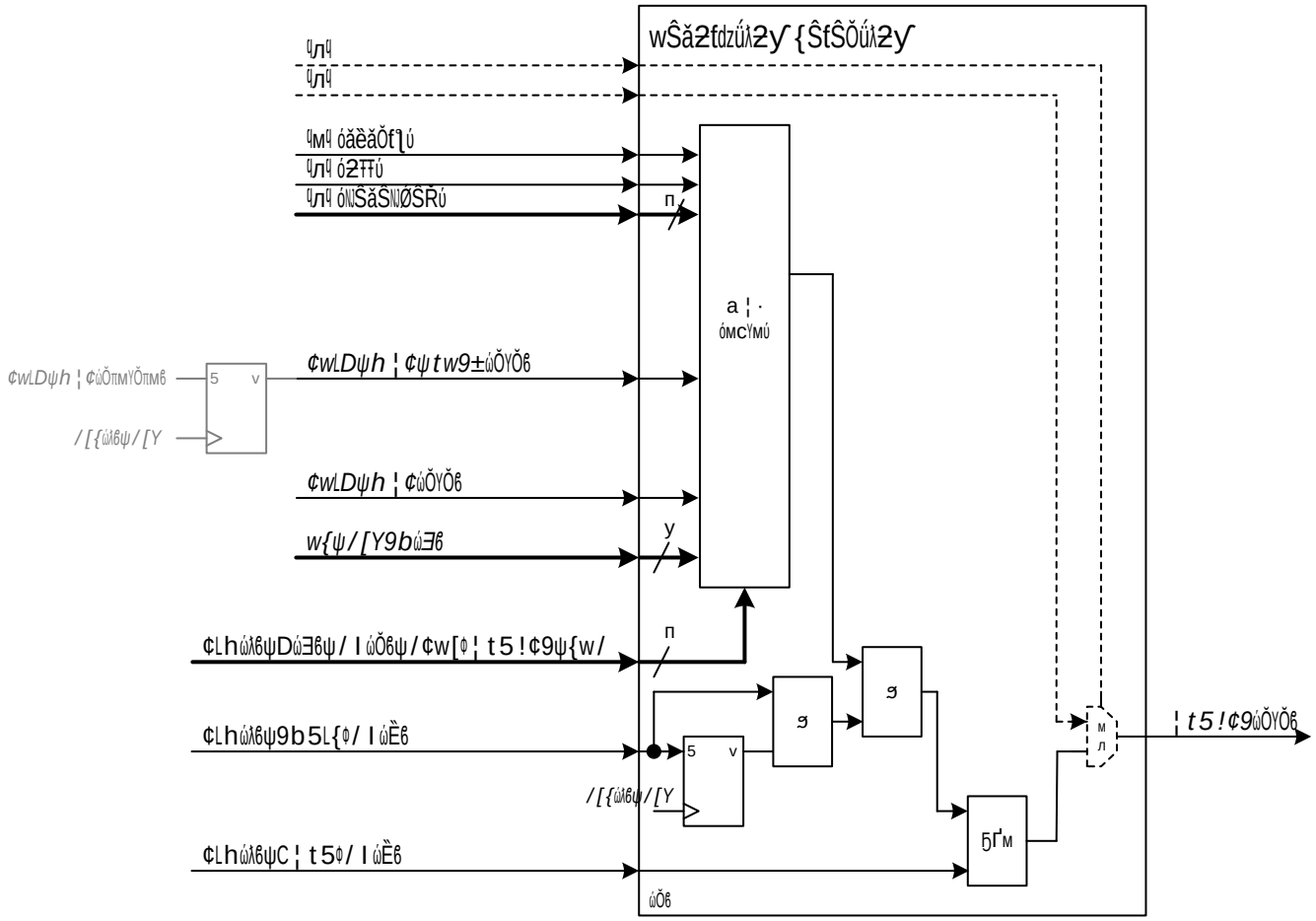
Figure 171 Channel Block Diagram



27.4.5 TIO Resolution Selection

This functional block allows to select the source for the update resolution used in channel c of group g. x = g*8 + c;

Figure 172 Resolution Selection



Note:

Dotted drawn parts in figure *172 "Resolution Selection"* are not available in the TIO basic implementation, they are in use with a TIO Plus implementation (chapter *27.5.7 "TIO_PL Resolution Selection"* ; figure *183 "TIO_PL Resolution Selection"*).

The signal *UPDATE* [c:c] = 1 defines the condition for an update of the output register **TIO[i]_O.CH[x]** or signal sampling register **TIO[i]_S.CH[x]** . There are multiple sources which can be selected with **TIO[i]_G[g]_CH[c]_CTRL.UPDATE_SRC** . This allows to select different sampling resolutions according to the needs of the application.

Typically, these signals are single TIO system clock cycle events. In the default configuration of the resource selection block the signals *RS_CLKEN[g]* [7:0] are identical to *CCM[i]_CLK_RES* [7:0].

Note:

Depending on the configuration of the trigger output control block (see chapter *27.4.7 "TIO Trigger Output Control"*), the signals *TRIG_OUT* [c:c] and *TRIG_OUT_PREV* [c:c] could generate high pulses which are longer than one TIO system clock cycle.

The control bit **TIO[i]_ENDIS.CH[x]** allows to enable / disable the *UPDATE* [c:c] event generation.

Enabling the *UPDATE* [c:c] event generation by setting of **TIO[i]_ENDIS.CH[x]** from 0b0 to 0b1 is delayed by one system clock cycle.
Disabling the *UPDATE* [c:c] event generation by setting of **TIO[i]_ENDIS.CH[x]** from 0b1 to 0b0 has no delay.

For the purpose of a software-based generation of an *UPDATE* [c:c] event the **TIO[i]_FUPD.CH[x]** register can be used. Each write of a 1 by the configuration interface to this bit field will issue a single system clock high pulse on *UPDATE* [c:c].

Note:

Keep in mind that a forced update introduced by writing 1 to **TIO[i]_FUPD.CH[x]** together with a single cycle high pulse event selected by the 16:1 MUX can result in a high pulses which is longer than one TIO system clock cycle.

In case of channel g = 0; c = 0 the signal *TRIG_OUT_PREV* [c:c] will be set to 0.

Be aware that the signal *TRIG_OUT_PREV* [c:c] which is used inside a channel c, is the signal *TRIG_OUT* [c-1:c-1] delayed by one TIO system clock.
Cascading of n channels by using *TRIG_OUT* [c:c] = *TRIG_OUT_PREV* [c-1:c-1] will result in n system clock cycles delay between *TRIG_OUT* [j:j] of source channel j to channel j+n *UPDATE* [j+n:j+n].

Example:

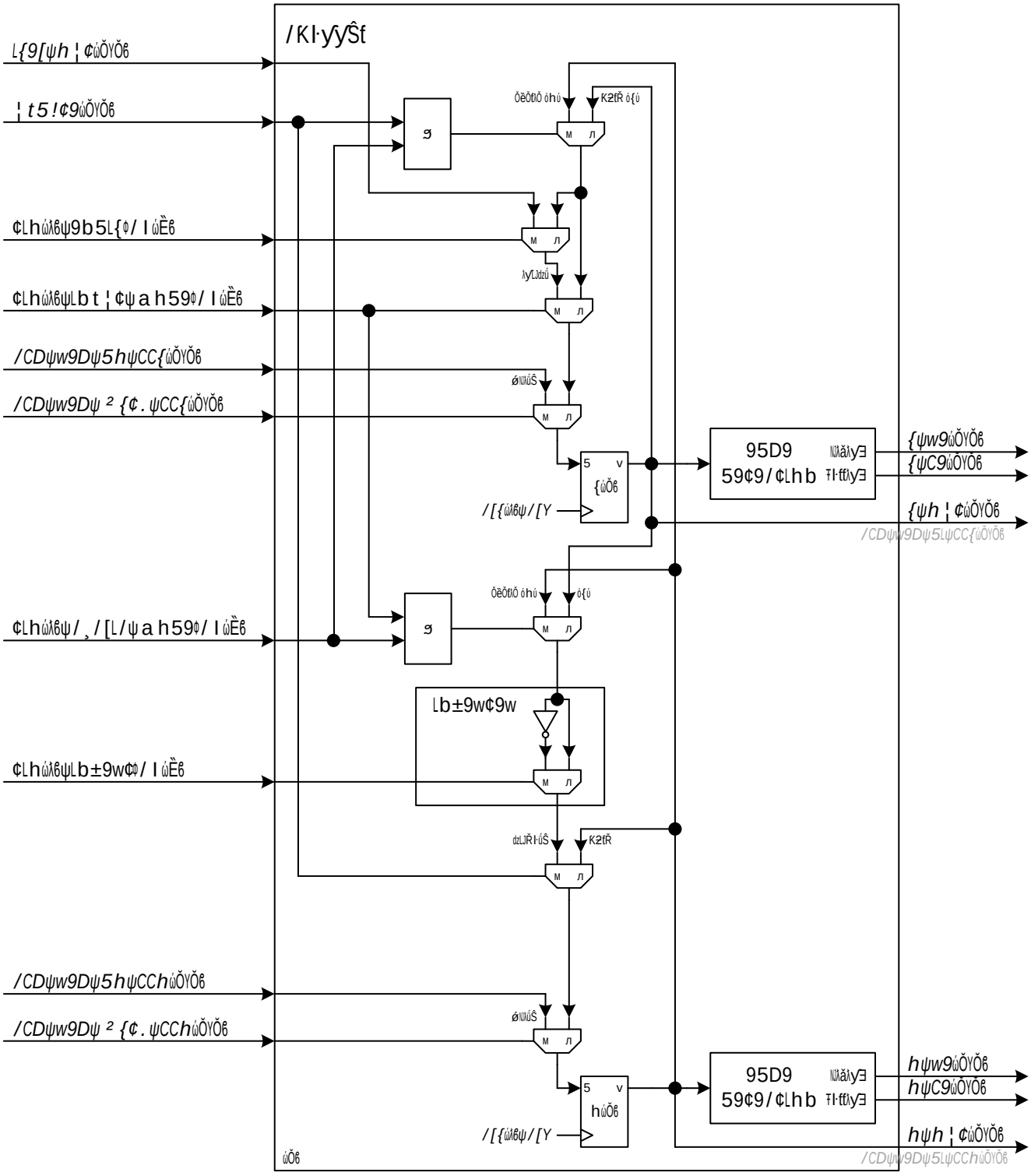
- In channel 7 the signal *TRIG_OUT* [0:0] shall be used as *UPDATE* [c:c].

- Configure **TIO[i]_G[g]_CH[i]_CTRL.UPDATE_SRC** = 0b1001.
- Signal from *TRIG_OUT* [0:0] will occur at *UPDATE* [c:c] with 7 TIO clock cycles delay.

27.4.6 TIO Channel Processing

This functional block covers the basic processing of a channel. Next is shown the block diagram.

Figure 173 Signal Sampling and Output



27.4.6.1 Signal Sampling Register TIO[i]_S

The configuration bit in register **TIO[i]_INPUT_MODE.CH[x]** = 1 selects *ISEL_OUT* [c:c] as source for the signal sampling register **TIO[i]_S.CH[x]** . The register **TIO[i]_S.CH[x]** operates on the TIO system clock.

The signal *ISEL_OUT* [c:c] from the input selection block is gated from processing by the signal sampling register **TIO[i]_S.CH[x]** by the enable/disable control signal **TIO[i]_ENDIS.CH[x]** (shown in figure *173 "Signal Sampling and Output"* as cfg.ENDIS).

The configuration bit in register **TIO[i]_INPUT_MODE.CH[x]** = 0 and **TIO[i]_CYCLIC_MODE.CH[x]** = 1 selects **TIO[i]_O.CH[x]** as source for the signal sampling register **TIO[i]_S.CH[x]** . The register **TIO[i]_S.CH[x]** operates on the chosen update resolution *UPDATE* [c:c].

Next table shows the functionality of **TIO[i]_S.CH[x]** :

Table 59 Sample Register TIO[i]_S.CH[x] Functionality

Cycle t: TIO[i]_INPUT_MODE.CH[x]	Cycle t: TIO[i]_ENDIS.CH[x]	Cycle t: TIO[i]_CYCLIC_MODE.CH[x]	Cycle t: *UPDATE* [c:c]	Cycle t+1: TIO[i]_S.CH[x]
0	–	–	0	**TIO[i]_S.CH[x]** at cycle t
0	–	0	–	**TIO[i]_S.CH[x]** at cycle t
0	–	1	1	**TIO[i]_O.CH[x]** at cycle t
1	0	–	0	**TIO[i]_S.CH[x]** at cycle t
1	0	0	–	**TIO[i]_S.CH[x]** at cycle t
1	0	1	1	**TIO[i]_O.CH[x]** at cycle t
1	1	–	–	*ISEL_OUT* [c:c] at cycle t

Independent of the bit fields **TIO[i]_INPUT_MODE.CH[x]** / **TIO[i]_CYCLIC_MODE.CH[x]** / **TIO[i]_ENDIS.CH[x]** the register **TIO[i]_S.CH[x]** can be updated with *CFG_REG_DO_FFS* [c:c] via the configuration interface (AEI) at any time (*CFG_REG_WSTB_FFS* [c:c] = 1). This write will be executed with highest priority.
On every TIO system clock an evaluation of a **TIO[i]_S.CH[x]** state change will take place. Any state change will be indicated by one pulse with duration of one TIO system clock period on the corresponding edge detection signal (*S_RE* [c:c], *S_FE* [c:c]).

At any point in time the state of the register **TIO[i]_S** can be read via the configuration interface (AEI).

Note:

There is a delay between the primary input signal *TIO_G[g]_IN* [c:c] and the corresponding *SYNC_OUT* [c:c] signal of two clock cycles and further from *SYNC_OUT* [c:c] to S[c] of at least 1 clock cycle (direct path through input selection block and resolution selection block configured to for example, cls_clk).

27.4.6.2 Output Register TIO[i]_O

The output register **TIO[i]_O.CH[x]** drives directly the output signal *O_OUT* [c:c] of the TIO channel.

The register **TIO[i]_O.CH[x]** operates on the chosen update resolution *UPDATE* [c:c].

The default behavior (**TIO[i]_INVERT.CH[x]** = 0; **TIO[i]_CYCLIC_MODE.CH[x]** = 0) is to register the value of **TIO[i]_S.CH[x]** into **TIO[i]_O.CH[x]** every time *UPDATE* [c:c] = 1.

The control bit **TIO[i]_INVERT.CH[x]** = 1 allows to invert the signal **TIO[i]_S.CH[x]** / **TIO[i]_O.CH[x]** before capturing in **TIO[i]_O.CH[x]** .

The source to be captured can be controlled by **TIO[i]_CYCLIC_MODE.CH[x]** , **TIO[i]_INPUT_MODE.CH[x]** .

Next table shows the functionality for **TIO[i]_O.CH[x]** .

Table 60 Output Register TIO[i]_O.CH[x] Functionality

Cycle t: TIO[i]_INPUT_MODE.CH[x]	Cycle t: TIO[i]_CYCLIC_MODE.CH[x]	Cycle t: TIO[i]_INVERT.CH[x]	Cycle t: *UPDATE* [c:c]	Tcycle t+1: TIO[i]_O.CH[x]
–	–	–	0	**TIO[i]_O.CH[x]** at cycle t
–	0	0	1	**TIO[i]_S.CH[x]** at cycle t
–	0	1	1	**TIO[i]_S.CH[x]** not at cycle t
0	–	0	1	**TIO[i]_S.CH[x]** at cycle t
0	–	1	1	**TIO[i]_S.CH[x]** not at cycle t
1	1	0	1	**TIO[i]_O.CH[x]** at cycle t
1	1	1	1	**TIO[i]_O.CH[x]** not at cycle t

Independent of the control bits **TIO[i]_INPUT_MODE.CH[x]** / **TIO[i]_INVERT.CH[x]** / **TIO[i]_CYCLIC_MODE.CH[x]** the register **TIO[i]_O.-CH[x]** can be updated with *CFG_REG_DO_FFO* [c:c] via the configuration interface (AEI) at any time (*CFG_REG_WSTB_FFO* [c:c] = 1). The write access over the configuration interface has the highest priority.

On every TIO system clock an evaluation of a **TIO[i]_O.CH[x]** state change will take place. Any state change will be indicated by one pulse with duration of one TIO system clock period on the corresponding edge detection signal (*O_RE* [c:c], *O_FE* [c:c]).

At any point in time the state of the register **TIO[i]_O** can be read via the configuration interface (AEI).

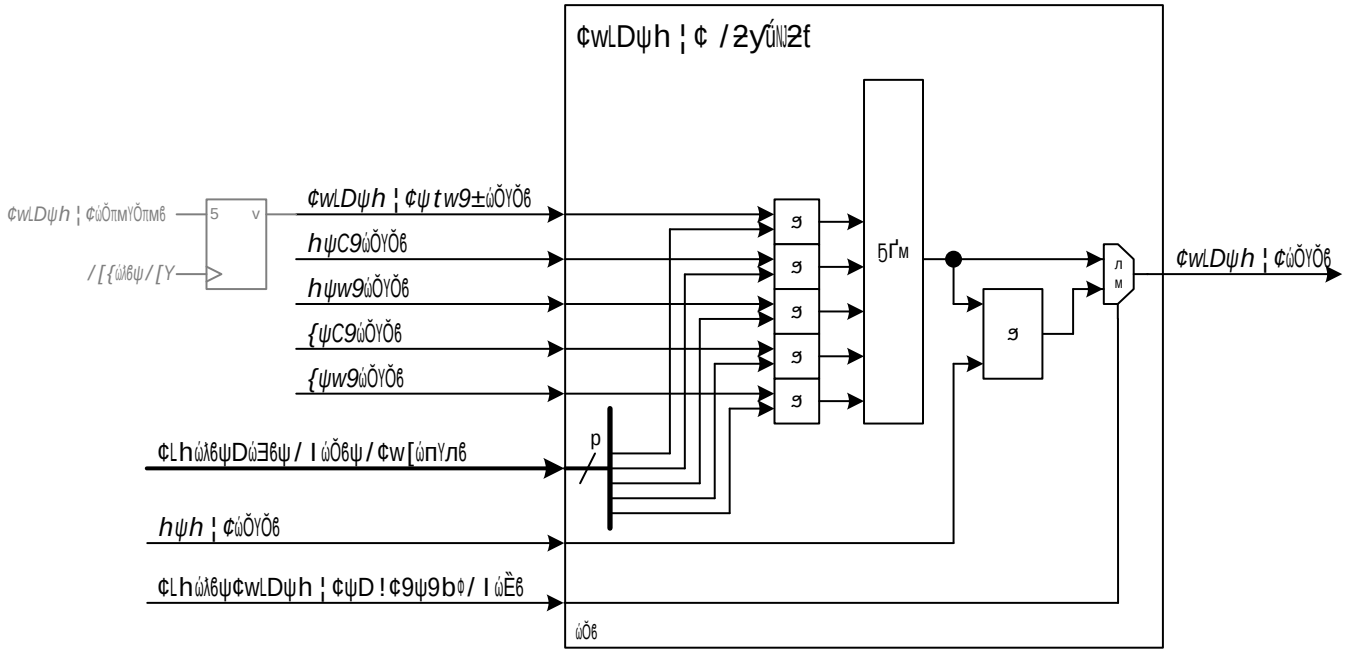
Note:

There is a delay between the primary input signal *TIO_G[g]_IN* [c:c] and the corresponding *SYNC_OUT* [c:c] signal of two clock cycles. Further from *SYNC_OUT* [c:c] to register S[c] of at least 1 clock cycle (direct path through input selection block and resolution selection block configured to for example, cls_clk) and between register S[c] and register O[c] of at least 1 clock cycle.

27.4.7 TIO Trigger Output Control

This functional block controls the trigger generation / selection of trigger output events.

Figure 174 TRIG_OUT Control



The following 5 signal sources for trigger generation on *TRIG_OUT* [c:c] exist:

- *S_RE* [c:c]
- *S_FE* [c:c]
- *O_RE* [c:c]
- *O_FE* [c:c]
- *TRIG_OUT_PREV* [c:c]

Each source can be enabled individually with the following configuration bits in register **TIO[i]_G[g]_CH[c]_CTRL** :

- **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_S_RE**
- **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_S_FE**
- **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_O_RE**
- **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_O_FE**
- **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_PREV_TRIG**

Due to the OR function of all enabled trigger event signals, it is possible to propagate none, individual or multiple of the 5 signal sources to *TRIG_OUT* [c:c].

Additionally the trigger generation can be gated by the state of *O_OUT* [c:c]when the control bit **TIO[i]_TRIG_OUT_GATE_EN.CH[x]** = 1 is in use.

Keep in mind that if multiple sources are enabled by **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_S_RE** , **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_S_FE** , **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_O_RE** , **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_O_FE** , **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_PREV_TRIG** , **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_PL_EVT** , **TIO[i]_G[g]_CH[c]_CTRL.TRIG_OUT_EN_PREV_PL_TRIG** , multiple different trigger events can overlay to a single event with length of >= 2 system clock cycles.

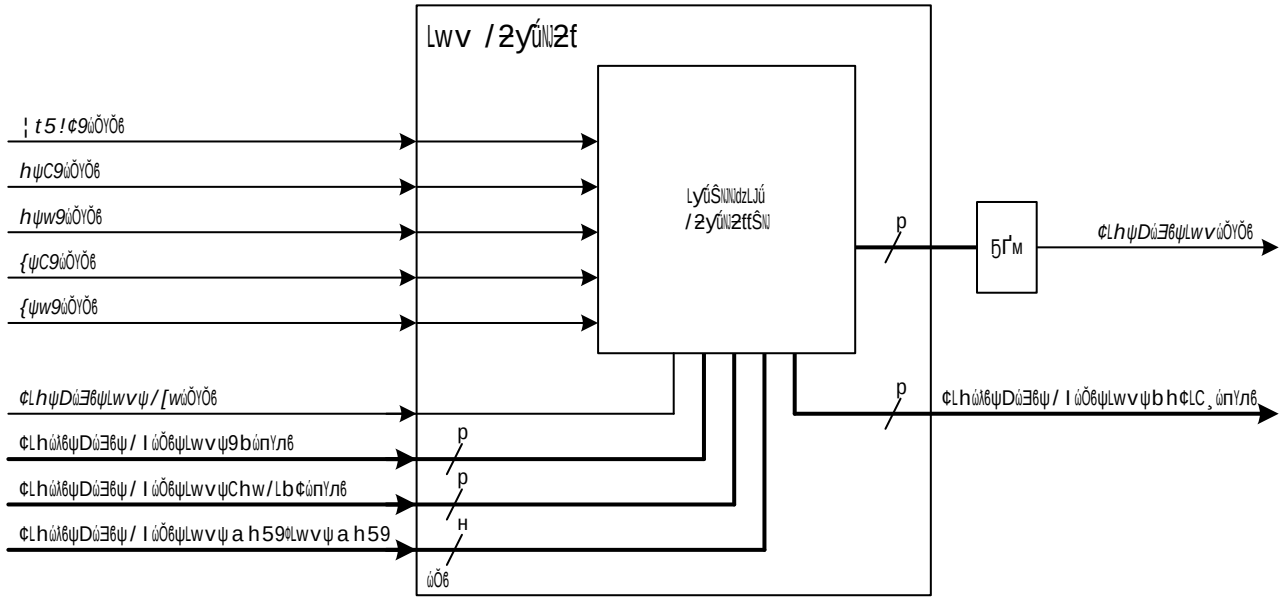
Note:

Any programmed application has to ensure that *TRIG_OUT* [c:c] pulses with length >= 2 system clock cycles, do not lead to unexpected application behavior.

27.4.8 TIO IRQ Control

This functional block controls the interrupt signaling to the outside of the TIO.

Figure 175 IRQ Control



The 5 TIO channel internal sources TIO_IRQ_SRC = { *UPDATE* [c:c], *O_RE* [c:c], *O_FE* [c:c], *S_RE* [c:c], *S_FE* [c:c]} are available to be enabled for interrupt signaling to CPU / GTM-MCS.

The GTM-IP interrupt handling scheme is described in the chapter Architecture *3.12 "GTM-IP Interrupt Concept"* . GTM-IP interrupt concept is in use for each TIO channel interrupt source. The functionality of the following registers is described there:

- **TIO[i]_G[g]_CH[c]_IRQ_NOTIFY**
- **TIO[i]_G[g]_CH[c]_IRQ_EN**
- **TIO[i]_G[g]_CH[c]_IRQ_FORCINT**
- **TIO[i]_G[g]_CH[c]_IRQ_MODE**

Each TIO channel [c] will have a distinct interrupt signal *TIO_IRQ* [c:c] to trigger threads on a CPU or GTM MCS.

Note:

The TIO IRQ control block does not support the signal IRQ_occurred described in the chapter *3.12 "GTM-IP Interrupt Concept"* .

27.4.9 TIO Atomic Operations On Multiple Channels

The address / register layout of the TIO module ensures that atomic operations on multiple channels inside one TIO module are possible. Furthermore, it is guaranteed that even in a multi core environment (CPU0..c; GTM MCS) with multiple threads (CPU threads, MCS-channels) atomic operations without any interference of cores / threads are possible.

Each channel x is assigned a unique bit position x in all configuration and operation registers:

- **TIO[i]_S**
- **TIO[i]_O**
- **TIO[i]_ENDIS**
- **TIO[i]_INVERT**
- **TIO[i]_INPUT_MODE**

- **TIO[i]_CYCLIC_MODE**
- **TIO[i]_TRIG_OUT_GATE_EN**

Writing via the configuration interface a value to these registers will change the behavior for all channels x (x ∈ {0, 1,..., 23}) simultaneously (atomic).

A configuration interface read operation will return the actual state of the register.

It is needed to alter the state of individual channels only without influencing the other channels to support the multi core / multi thread systems. This is possible by introducing 3 alternative register addresses for each of the above defined registers.

The alternative address is to set only the register bits.
This means writing a 1 to a dedicated bit will set the bit, writing a 0 will not change the bit. The register will be marked by the prefix S:

- **TIO[i]_SS**
- **TIO[i]_SO**
- **TIO[i]_SENDIS**
- **TIO[i]_SINVERT**
- **TIO[i]_SINPUT_MODE**
- **TIO[i]_SCYCLIC_MODE**
- **TIO[i]_STRIG_OUT_GATE_EN**

A configuration interface read operation will return the actual state of the register.

Another alternative address is to clear only the register bits.
This means writing a 1 to a dedicated bit will clear the bit, writing a 0 will not change the bit. The register will be marked by the prefix C:

- **TIO[i]_CS**
- **TIO[i]_CO**
- **TIO[i]_CENDIS**
- **TIO[i]_CINVERT**
- **TIO[i]_CINPUT_MODE**
- **TIO[i]_CCYCLIC_MODE**
- **TIO[i]_CTRIG_OUT_GATE_EN**

A configuration interface read operation will return the actual state of the register.

For some registers which are relevant for atomic operations where some bits have to set and others have to be cleared, a fourth alternative address is supported.
Writing a 1 to a dedicated bit will toggle the bit, writing a 0 will not change the bit. The register will be marked by the prefix I:

- **TIO[i]_IS**
- **TIO[i]_IO**
- **TIO[i]_IENDIS**
- **TIO[i]_IINVERT**
- **TIO[i]_IINPUT_MODE**
- **TIO[i]_ICYCLIC_MODE**

A configuration interface read operation will return the actual state of the register.

27.5 TIO Plus Functional Description

Naming convention: All functional blocks, signals and bit fields which are in use by the TIO Plus extension (TIO_PL) will have the prefix "PL_" or suffix "_PL".

The TIO Plus functionality enhances the TIO Basic functionality (figure *166 "TIO Block Diagram"*) in two ways:

o New functional blocks are added (colored blue in figure *176 "TIO Plus Block Diagram"*)

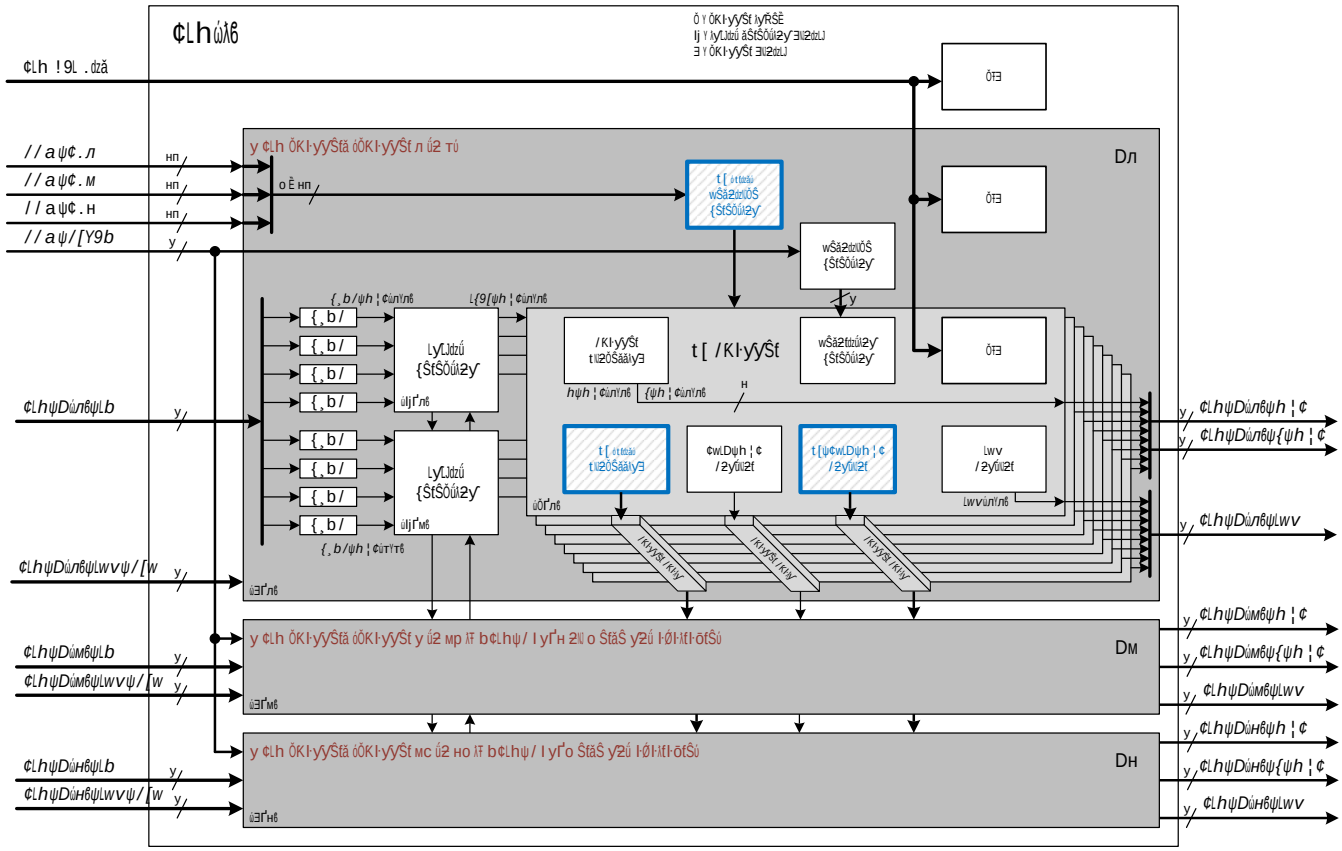
- TIO_PL Resource Selection (chapter *27.5.2 "TIO_PL Resource Selection"*)
- PL_TRIG_OUT Control (chapter *27.5.6 "TIO_PL PL Trigger Output Control"*)

- PL Processing (chapter *27.5.10 "TIO_PL Processing"*)

o TIO_Basic functional blocks are extended to be used together with the new TIO Plus functional blocks

- TIO_PL Channel (chapter *27.5.3 "TIO_PL Channel"*)
- TIO_PL Channel Processing (chapter *27.5.4 "TIO_PL Channel Processing"*)
- TIO_PL Resolution Selection (chapter *27.5.7 "TIO_PL Resolution Selection"*)
- TIO_PL Trigger Output Control (chapter *27.5.5 "TIO_PL Trigger Output Control"*)
- TIO_PL IRQ Control (chapter *27.5.8 "TIO_PL IRQ Control"*)

Figure 176 TIO Plus Block Diagram



27.5.1 TIO Plus Features

A summary of the features per channel, which are added to the TIO module based on the TIO Plus extension.

o Trigger generation / selection

- Select trigger signal of PL processing block
- Select trigger signal of PL processing block of previous channel

o Second Trigger generation /selection

- Generate trigger on rising and / or falling edge of input signal
- Generate trigger on rising and / or falling edge of output signal
- Select trigger signal of previous channel
- Select trigger signal of PL processing block
- Select trigger signal of PL processing block of previous channel

o Configurable update resolution

- The update resolution can be defined by PL processing block

- The update resolution can be defined by second trigger generation
- The update resolution can be defined by second trigger generation of previous channel

o Capture

- Controllable with 8-bit command
- 1 out of 3 24-bit time bases, together with input / output signal level on HW events (from trigger generation)
- Local 24-bit counter, together with input / output signal level on HW events (from trigger generation)
- Set / clear / toggle output signal based on occurrence of capture event

o Counter

- Controllable with 8-bit command
- Count HW events (from trigger generation)
- Count on signal level (high / low)
- Start counting on HW events (from trigger generation)
- Stop counting on HW events (from trigger generation)

o Compare

- Controllable with 8-bit command / 24-bit operand
- Set / clear / toggle output signal based on occurrence of compare event
- 24-bit time base comparison with target value using equal / greater than or equal comparison
- Single compare (one shot)
- Multiple compare (endless)
- Serve first / serve both dual compare (means two compares are active in parallel)

o Serial shift output generation

- Controllable with 8-bit command / 24-bit operand
- Shift out based on update resolution
- Single pattern length <= 24 bit or (<= 48 bit by using S resource as instruction buffer)

o Serial shift input sampling

- Controllable with 8-bit command
- Input sampling of up to 24-bits based on update resolution
- Support of pattern length > 24 bit and <=48 bit by using S resource as capture buffer in next channel

o Support for applications using multiple channels

- Synchronous start / stop of multiple channels
- Multiple channels using common time base

Channel resources usable for data storage

- Configurable to be used as data buffer for capture, compare or shift operation
- Configurable to be used as instruction buffer for capture, compare, count or shift operation (8-bit command / 24-bit data)

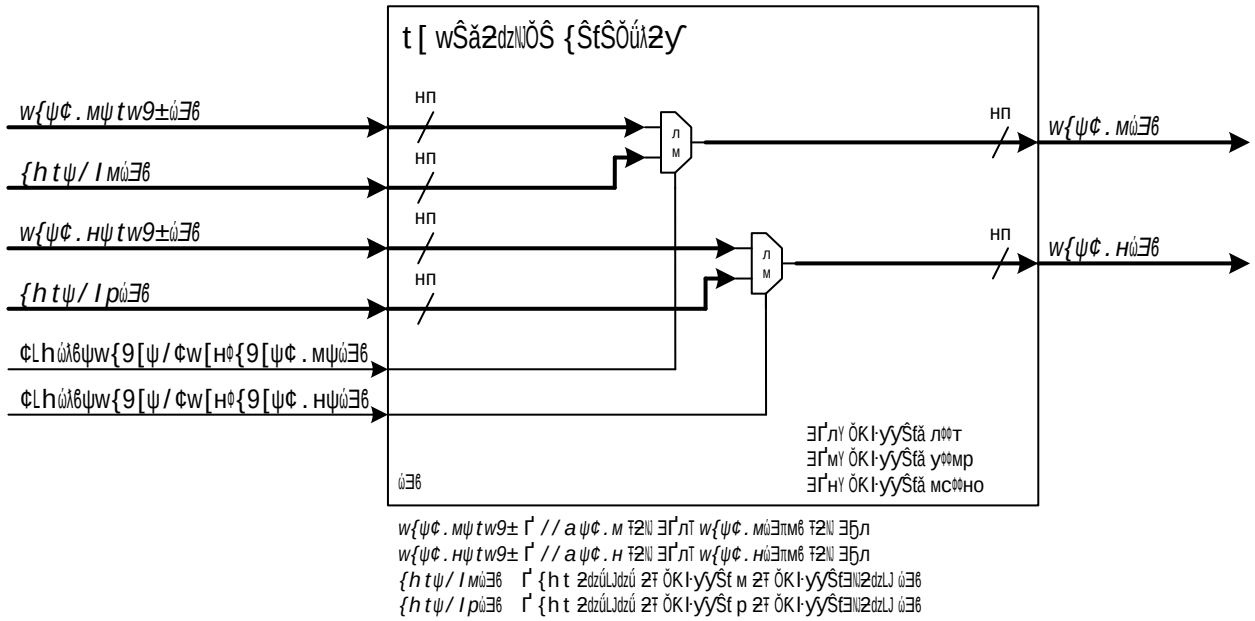
A list of typical application examples are described in chapter *27.6.2 "TIO_PL Applications"* .

27.5.2 TIO_PL Resource Selection

The PL Resource Selection block is added on the TIO top level.
The compare and capture features operate on a 24-bit time base. This functional block allows to select the source for the time base *RS_TB1[g]* and *RS_TB2[g]* .

The third available time base is the cluster global time base *CCM[i]_TBU_TS0* provided by the TBU / CCM submodule of the GTM. It is provided to all channel groups with the input signal *CCM[i]_TBU_TS0* of the TIO (see figure *178 "TIO Plus Connectivity of NTIO_CH8=3 Resource Selection Instances"*). *RS_TB0[g]* = *CCM[i]_TBU_TS0*

Figure 177 TIO Plus Resource Selection



The resource selection is available for every group of 8 TIO_PL channels:

- Group g = 0: TIO_PL channel x (x ∈ {0, 1,..., 7})
- Group g = 1: TIO_PL channel x (x ∈ {8, 9,..., 15})
- Group g = 2: TIO_PL channel x (x ∈ {16, 17,..., 23})

Table 61 TIO_PL Resource Selection – Table 1/2

Group g	TIO[i]_RSEL_CTRL2.SEL_TB1_[g]	Selected time base *RS_TB1[g]*
0	0	*CCM[i]_TBU_TS1*
0	1	**TIO[i]_G[0]_CH[1]_SINST.OP**
1	0	*RS_TB1[0]*
1	1	**TIO[i]_G[1]_CH[1]_SINST.OP**
2	0	*RS_TB1[1]*
2	1	**TIO[i]_G[2]_CH[1]_SINST.OP**

Table 62 TIO_PL Resource Selection – Table 2/2

Group g	TIO[i]_RSEL_CTRL2.SEL_TB2_[g]	Selected time base *RS_TB2[g]*
0	0	*CCM[i]_TBU_TS2*
0	1	**TIO[i]_G[0]_CH[5]_SINST.OP**
1	0	*RS_TB2[0]*
1	1	**TIO[i]_G[1]_CH[5]_SINST.OP**
2	0	*RS_TB2[1]*
2	1	**TIO[i]_G[2]_CH[5]_SINST.OP**

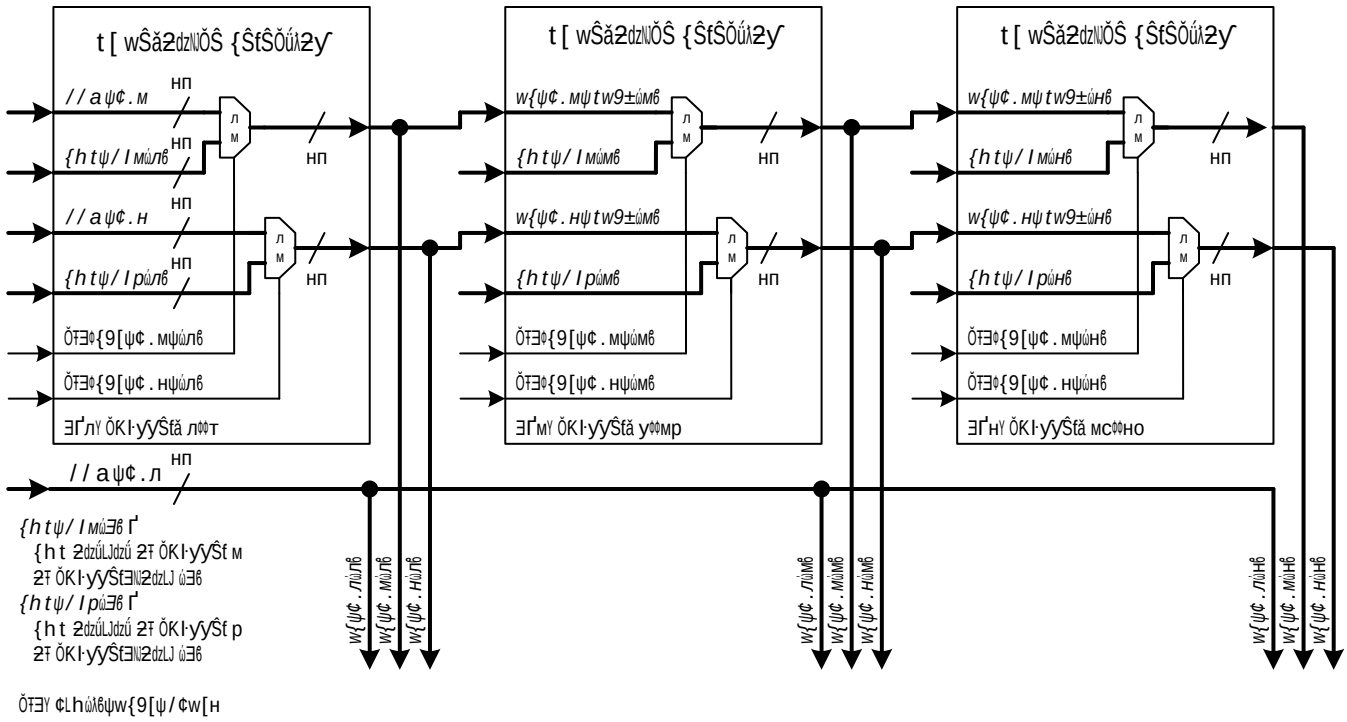
By default (**TIO[i]_RSEL_CTRL2.SEL_TB2_[g]** = 0; **TIO[i]_RSEL_CTRL2.SEL_TB1_[g]** = 0; g ∈ {0, 1, 2}) the cluster global *CCM[i]_TBU_TS1* ; *CCM[i]_TBU_TS2* time bases are in use:

- *RS_TB1[g]* = *CCM[i]_TBU_TS1*
- *RS_TB2[g]* = *CCM[i]_TBU_TS2*

Alternatively the local operand **TIO[i]_G[g]_CH[1]_SOP.OP** as source of *RS_TB1[g]* and **TIO[i]_G[g]_CH[5]_SOP.OP** as source of *RS_TB2[g]* can be used. This is useful, if SOP is configured to implement a counter which can wrap around at a definable value (Period counter for a PWM). This counter value (**TIO[i]_G[g]_CH[c]_SINST.OP**) can be used similar to a time base.

Next figure shows the resource selection connectivity of NTIO_CH8 = 3 instances, supporting 24 channels.

Figure 178 TIO Plus Connectivity of NTIO_CH8=3 Resource Selection Instances



27.5.3 TIO_PL Channel

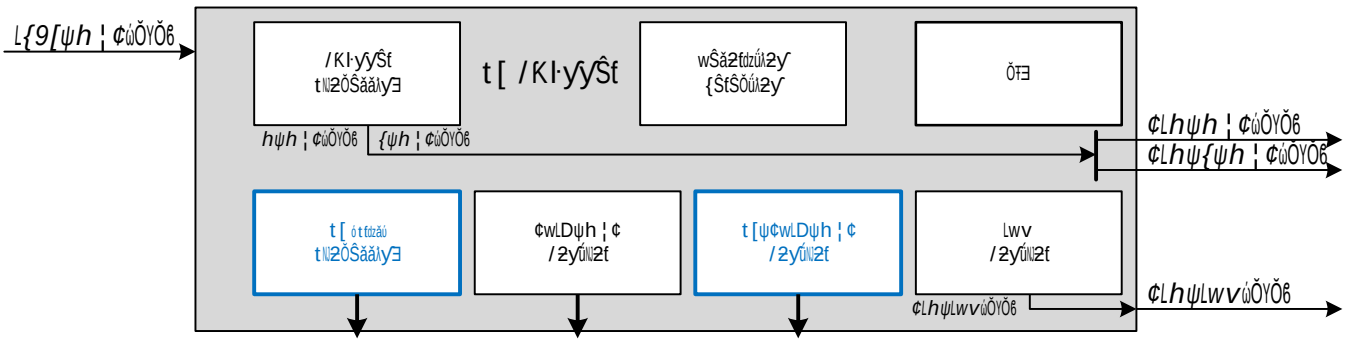
Next Figure *179 "TIO Plus Channel Block Diagram"* , shows the TIO_PL channel architecture. The following chapters will introduce the TIO_PL extensions to the TIO_Basic functional blocks. All extensions are shown in blue color.

The read only configuration bit **TIO[i]_HW_CONF.TIO_PLUS** , indicates the availability of the TIO_PL functionality.
0: TIO_Basic function available only
1: TIO_Basic is enhanced with TIO_PL functionality

The TIO_PL functionality is structured in channels and operates on the resources of a TIO_Basic channel processing block (e.g.: **TIO[i]_S.CH[x]** ; **TIO[i]_O.CH[x]** ; *TRIG_OUT* [c:c]; *PL_UPDATE* [c:c]).

Additional functional blocks are:
1 PL Processing: resources to count, measure, capture and generate signals
1 PL Trigger Output Control: second trigger channel, similar to Trigger Output Control

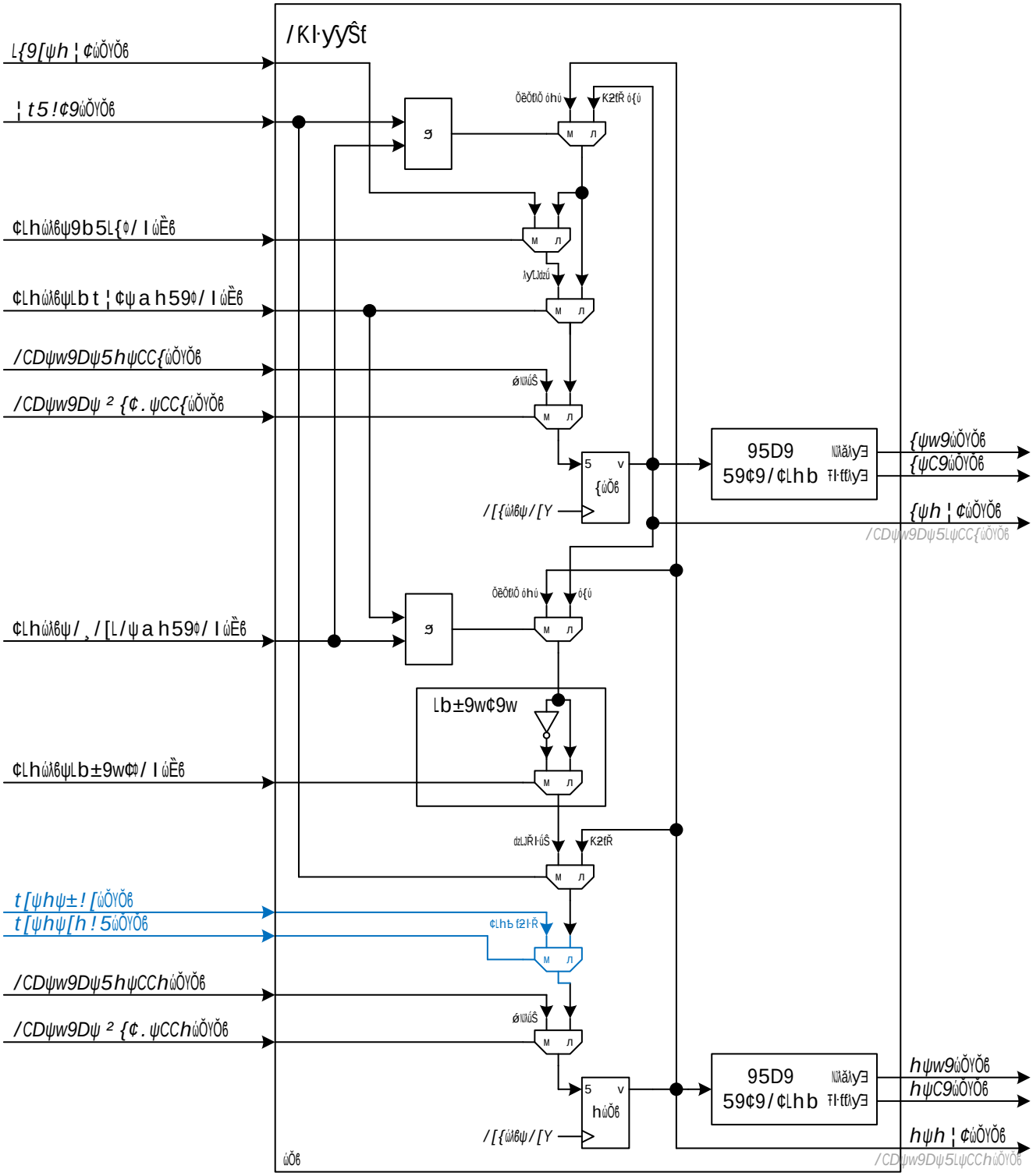
Figure 179 TIO Plus Channel Block Diagram



27.5.4 TIO_PL Channel Processing

In the TIO_PL implementation an additional possibility to alter the state of the output register **TIO[i]_O.CH[x]** register is available. This interface is used by the TIO_PL Processing block to influence the output **TIO[i]_O.CH[x]** . Independent of the control bits **TIO[i]_INPUT_-MODE.CH[x]** / **TIO[i]_INVERT.CH[x]** / **TIO[i]_CYCLIC_MODE.CH[x]** , the register **TIO[i]_O.CH[x]** can be updated with *PL_O_VAL* [c:c] via *PL_O_LOAD* [c:c] = 1.

Figure 180 TIO Plus Signal Sampling and Output

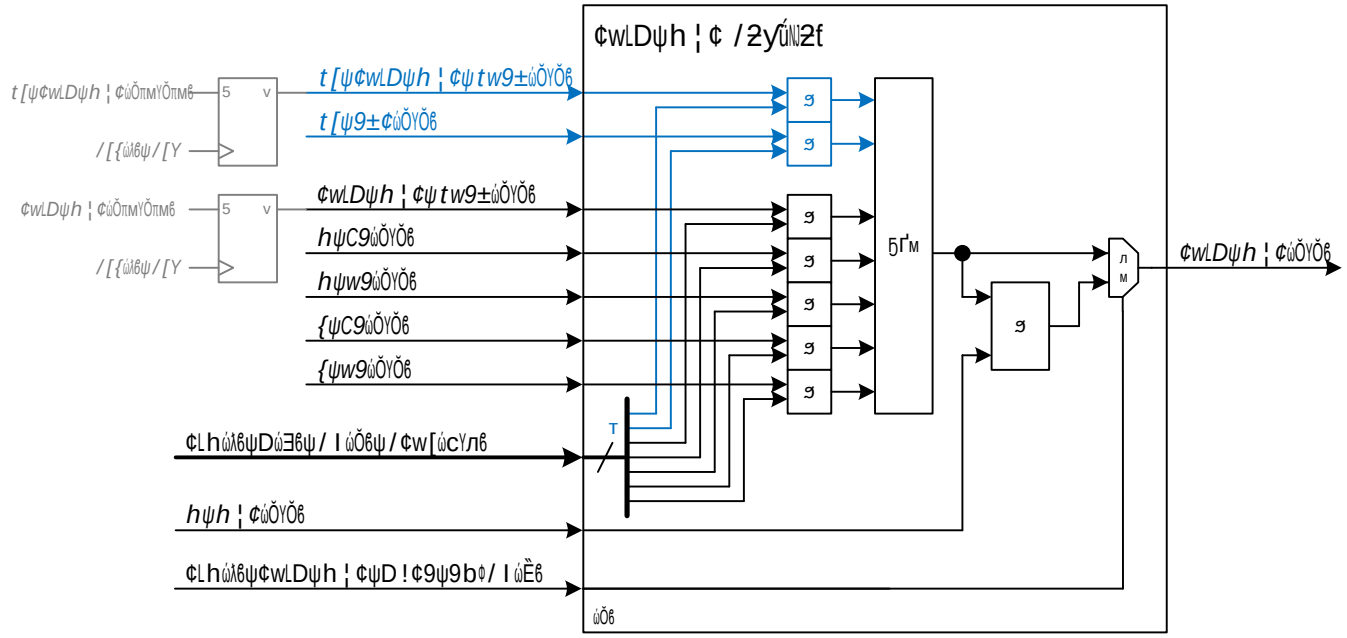


As can be seen in Figure *180 "TIO Plus Signal Sampling and Output"* , the write access to register **TIO[i]_O.CH[x]** (*CFG_REG_DO_FFO* [c:c] and *CFG_REG_WSTB_FFO* [c:c]) has precedence over other paths.

27.5.5 TIO_PL Trigger Output Control

The trigger output event generation of TIO_Basic *27.4.7 "TIO Trigger Output Control"* , is extended to support the additional *PL_EVT* [c:c] and *PL_TRIG_OUT_PREV* [c:c] events as inputs for the trigger selection.

Figure 181 TIO_PL Trigger Output Control



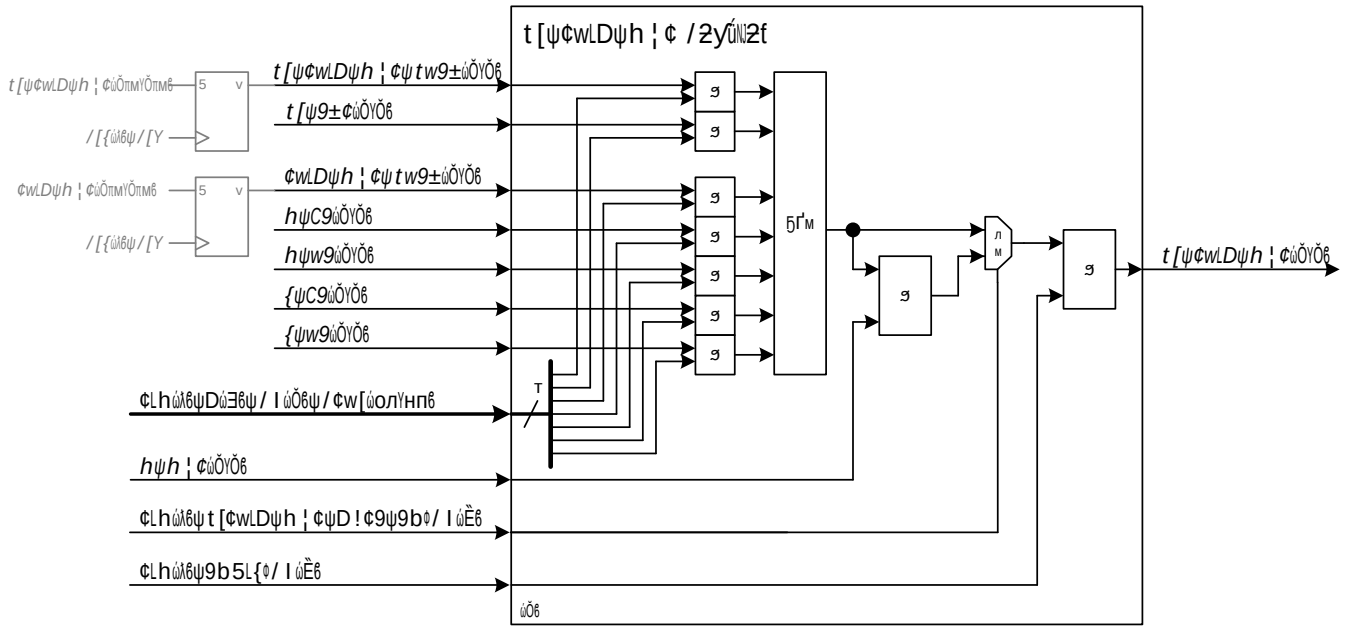
27.5.6 TIO_PL PL Trigger Output Control

In addition to the trigger output event generation *27.5.5 "TIO_PL Trigger Output Control"* , the TIO_PL adds a second similar independent *PL_TRIG_OUT* [c:c] event generation mechanism.

These two trigger control modules mainly control the behavior of the S and O resource functionality. Details will be discussed in the following chapters of the functional description of the individual modes.

The *PL_TRIG_OUT* [c:c] signal is enabled/disabled by **TIO[i]_ENDIS.CH[x]** . This prevents that TIO plus functionality of the corresponding channel [c] is triggered by a *PL_TRIG_OUT* [c:c] trigger pulse while the channel is disabled.

Figure 182 TIO_PL PL_TRIG_OUT Control



27.5.7 TIO_PL Resolution Selection

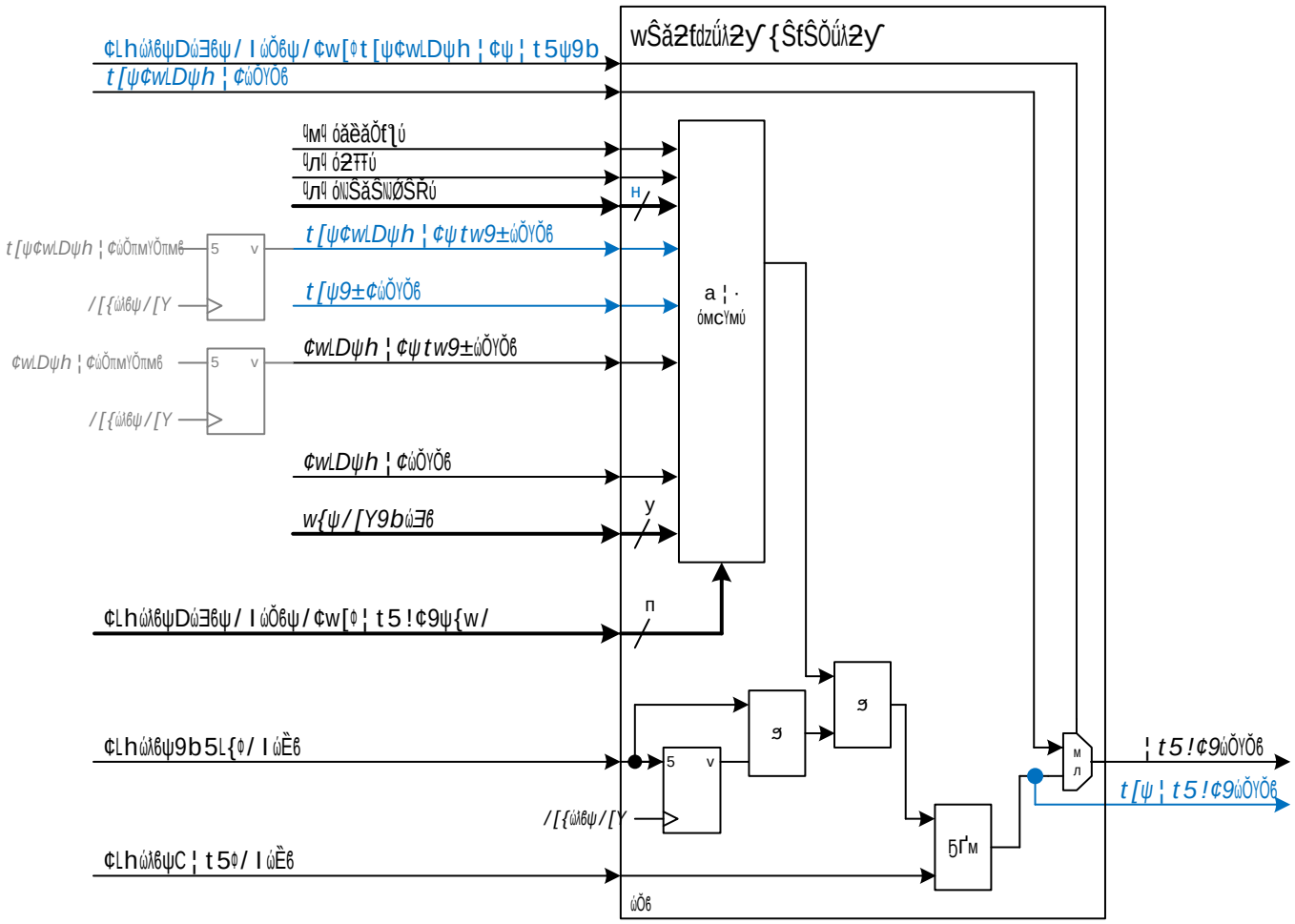
In the TIO_PL implementation, the resolution selection is enhanced. In addition, the events *PL_EVT* [c:c] or *PL_TRIG_OUT_PREV* [c:c] can be selected.

A new output signal *PL_UPDATE* [c:c] is added, this will be used only in the PL Processing block.

With the bit field **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_UPD_EN** the source for *UPDATE* [c:c] is selected:
0: *UPDATE* [c:c] = *PL_UPDATE* [c:c]
1: *UPDATE* [c:c] = *PL_TRIG_OUT* [c:c]; this configuration allows to use in the PL Processing block another update resolution as in the Channel Processing block.

Enabling the *UPDATE* [c:c] event generation by setting of **TIO[i]_ENDIS.CH[x]** from 0b0 to 0b1 is delayed by one system clock cycle when **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_UPD_EN** = 0.
Disabling the *UPDATE* [c:c] event generation by setting of **TIO[i]_ENDIS.CH[x]** from 0b1 to 0b0 has no delay.

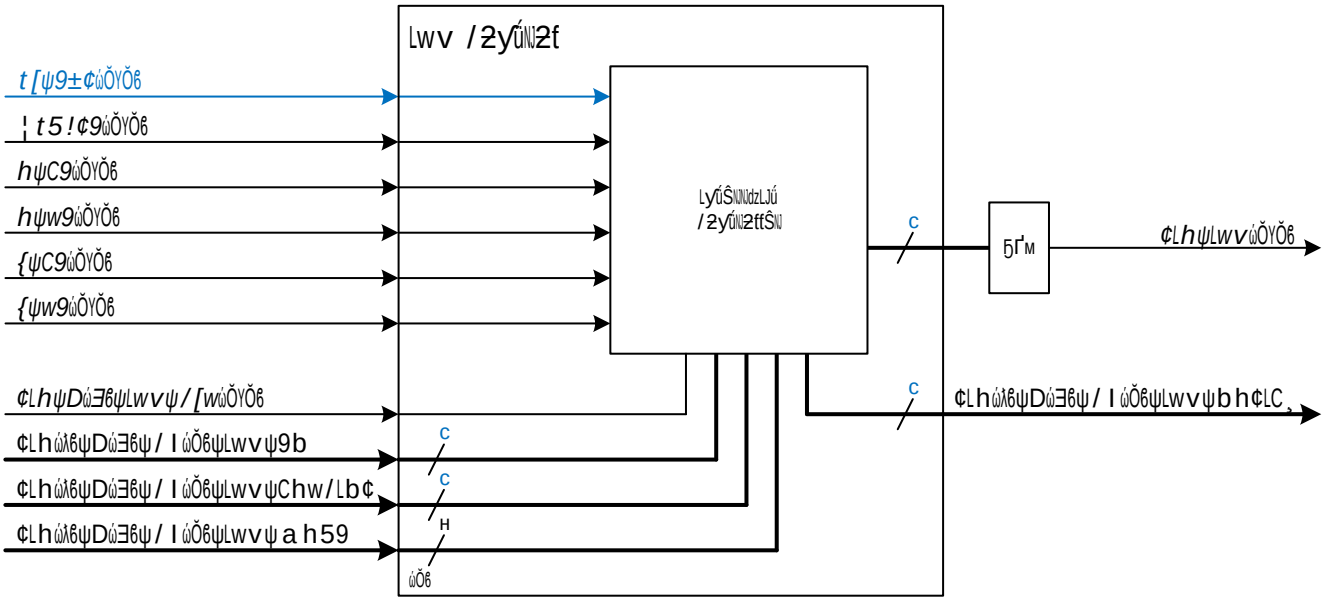
Figure 183 TIO_PL Resolution Selection



27.5.8 TIO_PL IRQ Control

The IRQ control generation is extended to support the additional *PL_EVT* [c:c] event as a usable interrupt trigger source.

Figure 184 TIO Plus IRQ Control



27.5.9 TIO_PL Atomic Operations on Multiple Channels

Following address / registers have been extended to support the TIO_PL extensions. They operate similar as described in chapter *27.4.9 "TIO Atomic Operations On Multiple Channels"* .

Each channel x is assigned a unique bit position x in all configuration and operation registers:
TIO[i]_PLTRIG_OUT_GATE_EN

Writing via the configuration interface a value to these registers will change the behavior for all channels x (x ∈ {0, 1,..., 23}) simultaneously (atomic).

A configuration interface read operation will return the actual state of the register.

One alternative address to set only register bits, this means writing a 1 to a dedicated bit will set the bit, writing a 0 will not change the bit. The register will be marked by the prefix S: **TIO[i]_SPLTRIG_OUT_GATE_EN**

A configuration interface read operation will return the actual state of the register.

Another alternative address to clear only register bits, this means writing a 1 to a dedicated bit will clear the bit, writing a 0 will not change the bit. The register will be marked by the prefix C: **TIO[i]_CPLTRIG_OUT_GATE_EN**

A configuration interface read operation will return the actual state of the register.

27.5.10 TIO_PL Processing

27.5.10.1 Overview

Each PL Processing block is equipped with:

o S resource (yellow / top section in figure *185 "TIO Plus – Plus Processing Block Diagram"*); operating on signal sampling register S[c] of TIO channel processing

- 8-bit Command register **TIO[i]_G[g]_CH[c]_SINST.CMD**
- 24-bit Operand / working registers **TIO[i]_G[g]_CH[c]_SINST.OP**
- Increment logic
- Decrement logic
- PL_S control logic
- **TIO[i]_G[g]_CH[c]_SINST.CMD** / **TIO[i]_G[g]_CH[c]_SINST.OP** source selection (Multiplexer)

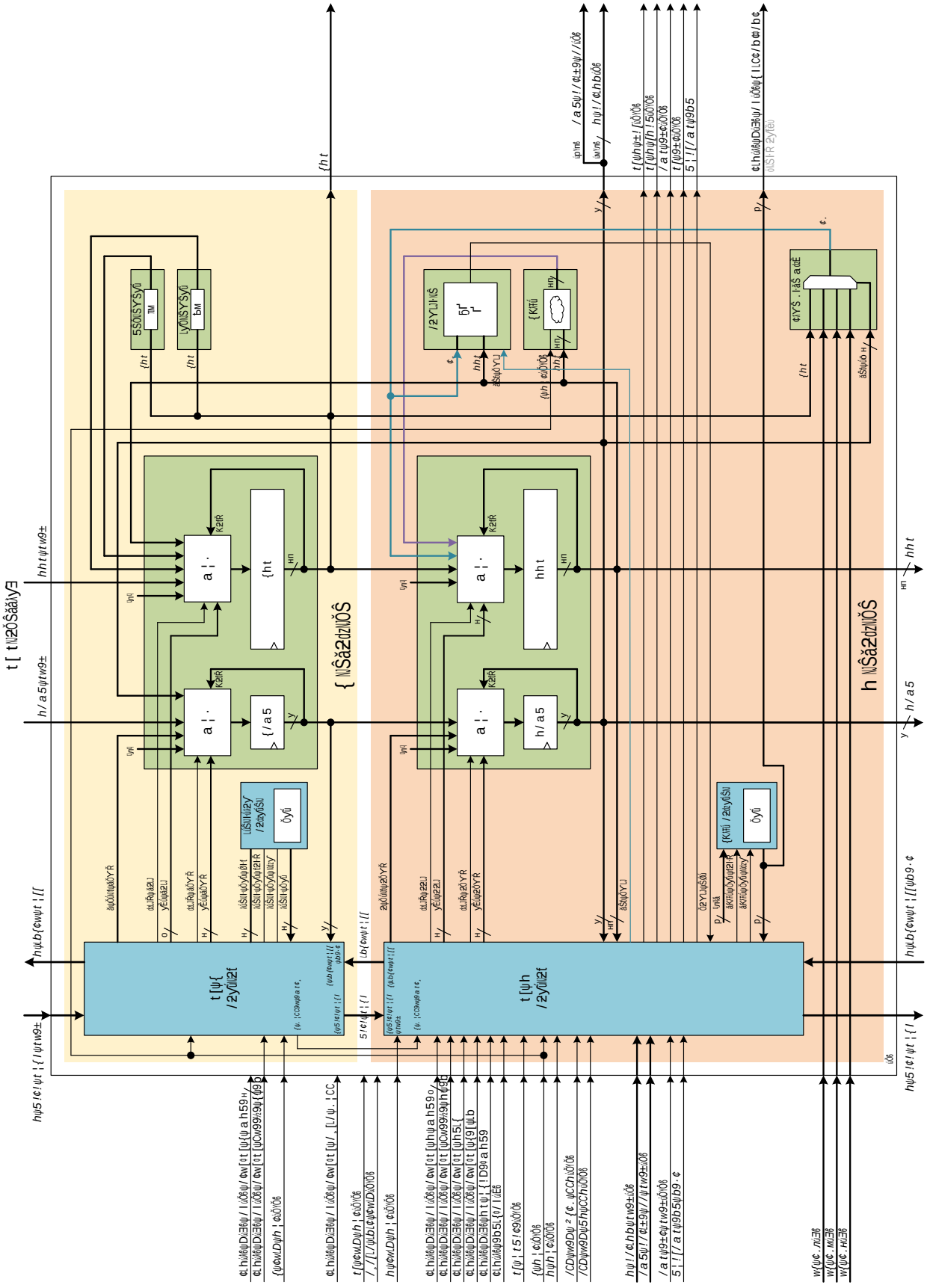
o O resource (red /bottom section in figure *185 "TIO Plus – Plus Processing Block Diagram"*); controlling signal Output register O[c] of TIO channel processing

- 8-bit Command register **TIO[i]_G[g]_CH[c]_OINST.CMD**

- 24-bit Operand / working registers **TIO[i]_G[g]_CH[c]_OINST.OP**
- Time base multiplexer
- Comparator (equal / greater than or equal)
- Shift logic / shift counter
- PL_O control logic
- **TIO[i]_G[g]_CH[c]_OINST.CMD** / **TIO[i]_G[g]_CH[c]_OINST.OP** source selection (Multiplexer)

A detailed block diagram which shows the PL processing block is shown next.

Figure 185 TIO Plus – Plus Processing Block Diagram



The S resource can be configured with the bit fields **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** to provide distinct functions. Following functions are available:

Table 63 S Resource Functions

TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE	S resource function	Description
0b00	Start buffer	S resource is operating as start buffer (chapter *27.5.10.9 "S Resource Start Buffer"*)
0b01	Continue buffer	S resource is operating as continue buffer (chapter *27.5.10.10 "S Resource Continue Buffer"*)
0b10	Counter; enable stop on *PL_TRIG_OUT* [c:c]	S resource is operating as counter (chapter *27.5.10.8 "S Resource Counter Instructions"*); enable stop (chapter *27.5.10.8.2 "Control Iterations of Count Instruction"*)
0b11	Counter; disable stop on *PL_TRIG_OUT* [c:c]	S resource is operating as counter (chapter *27.5.10.8 "S Resource Counter Instructions"*); disable stop (chapter *27.5.10.8.3 "Count Instruction Stopping Disabled"*)

The O resource can be configured with the bit fields **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** to provide distinct functions. Following functions are available:

Table 64 O Resource Functions

TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE	O resource function	Description
0b000	Start buffer	O resource is operating as start buffer (chapter *27.5.10.11 "O Resource Start Buffer"*)
0b001	Continue buffer	O resource is operating as continue buffer (chapter *27.5.10.12 "O Resource Continue Buffer"*)
0b010	Shift left	O resource is operating shift left (chapter *27.5.10.5 "O Resource Shift Instructions"*)
0b011	Shift right	O resource is operating shift right(chapter *27.5.10.5 "O Resource Shift Instructions"*)
0b100	Compare "greater than or equal"; disable signal capture	O resource is operating as compare (chapter *27.5.10.7 "O Resource Compare Instructions"*); "greater than or equal" (chapter *27.5.10.7.3 "Select "greater than or equal" Compare Instruction"*)
0b101	Compare "greater than or equal"; enable signal capture	O resource is operating as compare (chapter *27.5.10.7 "O Resource Compare Instructions"*); "greater than or equal" (chapter *27.5.10.7.3 "Select "greater than or equal" Compare Instruction"*) enable capture (chapter *27.5.10.7.13 "Compare Instruction With Capture"*)
0b110	Compare "equal"; disable signal capture	O resource is operating as compare (chapter *27.5.10.7 "O Resource Compare Instructions"*); "equal" (chapter *27.5.10.7.2 "Select Equal Compare Instruction"*)
0b111	Capture	O resource is operating as capture (chapter *27.5.10.6 "O Resource Capture Instructions"*)

With these functions multiple applications can be supported e.g.:

- PWM output generation: count and compare function used
- Input signal measurement: count and capture function used
- Serial in / output support: shift function used
- Signal generation on counter value: compare function used

Each S / O resource function (count, compare, shift, capture) is configurable with instructions. Each instruction uses a 8-bit command and a 24-bit operand. The 8 bit command allows to configure different behaviors of an instruction. They are programmable with the following registers or multiview registers:

Note:

Multiview register introduces for one implemented register with a unique address multiple views, where bit fields can be defined differently depending on the configured instruction.

o S resource: register **TIO[i]_G[g]_CH[c]_SINST**

- Multiview register **TIO[i]_G[g]_CH[c]_SINST_COUNT**
- Multiview register **TIO[i]_G[g]_CH[c]_SINST_COUNT12**

o O resource: register **TIO[i]_G[g]_CH[c]_OINST**

- Multiview register **TIO[i]_G[g]_CH[c]_OINST_COMP**
- Multiview register **TIO[i]_G[g]_CH[c]_OINST_COMP12**
- Multiview register **TIO[i]_G[g]_CH[c]_OINST_CAP**
- Multiview register **TIO[i]_G[g]_CH[c]_OINST_SHIFT**

Note: In figure *185 "TIO Plus – Plus Processing Block Diagram"* and in the following chapters following synonyms are in use:

- SCMD : bit field **TIO[i]_G[g]_CH[c]_SINST.CMD**
- SOP : bit field **TIO[i]_G[g]_CH[c]_SINST.OP**
- OCMD : bit field **TIO[i]_G[g]_CH[c]_OINST.CMD**
- OOP : bit field **TIO[i]_G[g]_CH[c]_OINST.OP**

The behavior of a function depends on the configured value in the command register **TIO[i]_G[g]_CH[c]_SINST.CMD** / **TIO[i]_G[g]_CH[c]_-OINST.CMD** .

2 bit fields are identical in name and function and are available for all instructions, they control together with the start buffer / continue buffer function, how data will be moved. The data can be pulled from previous channel in case of instruction buffer or pushed to the next channel in case of capture buffer.

- **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** : controls the usage of an instruction buffer
- **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** : controls the usage of a capture buffer

The remaining 6 bits of the command register are function dependent.

Figure 186 CMD Register Layout – Common Bit Fields

	lb{¢wψt¦[[ψ9b	5!¢!ψt¦{lψ9b						
lb{¢	ом	ол	нф	ну	нт	нс	нр	нп
/a5	т	с	р	п	о	н	м	л

With this approach of configuring a channel resource to operate as a distinct function and configuring of an instruction for the function, the usage of the TIO_PL resources can be tailored specific to the application needs. Furthermore, the usage of an instruction buffer allows to specify instruction sequences which can be executed similar to a "linear program flow" of a CPU.

An instruction can be started once or operate in an endless manner. In case of an instruction which is executed once, it is possible to define an instruction buffer holding an instruction, which has to be executed after termination of the instruction being executed currently.

Each S/O resource which is configured as a start buffer is able to hold one instruction. By cascading a start buffer with multiple (n-1) adjacent continue buffer resources an instruction buffer of length n can be built. It is necessary that the user initializes all the resources in the instruction buffer to values other than 0x0. The usage of an instruction buffer with 0x0 written to one (or more) resources is deemed unintended and results in undefined behavior.

An instruction buffer of length 1 is very usual. This allows to setup the next instruction, while the current instruction is in execution. When the execution of the current instruction terminates, the next instruction will be loaded automatically from the instruction buffer.

The other usage of the continue buffer resource is to build up a capture buffer. Instructions of the functions operating on channel [c] (count, capture, compare, shift) can be used to measure / timestamp internal or external signals / events. This data is captured in the

TIO[i]_G[g]_CH[c]_SINST.OP or **TIO[i]_G[g]_CH[c]_OINST.CMD** / **TIO[i]_G[g]_CH[c]_OINST.OP** register. By operating these instructions multiple times, the previously captured data is overwritten. This can be prevented by setting up a capture buffer of certain length n by cascading n adjacent continue buffer resources behind the function operating in channel [c].

Following terms are in use in the next chapters:

Instruction without reload: an instruction which is in operation once and after termination no further instruction will be loaded.

Instruction freeze: an instruction which is in operation endlessly, after termination the same instruction will remain active.

Instruction with reload: an instruction which is in operation once and after termination a new instruction will be loaded.

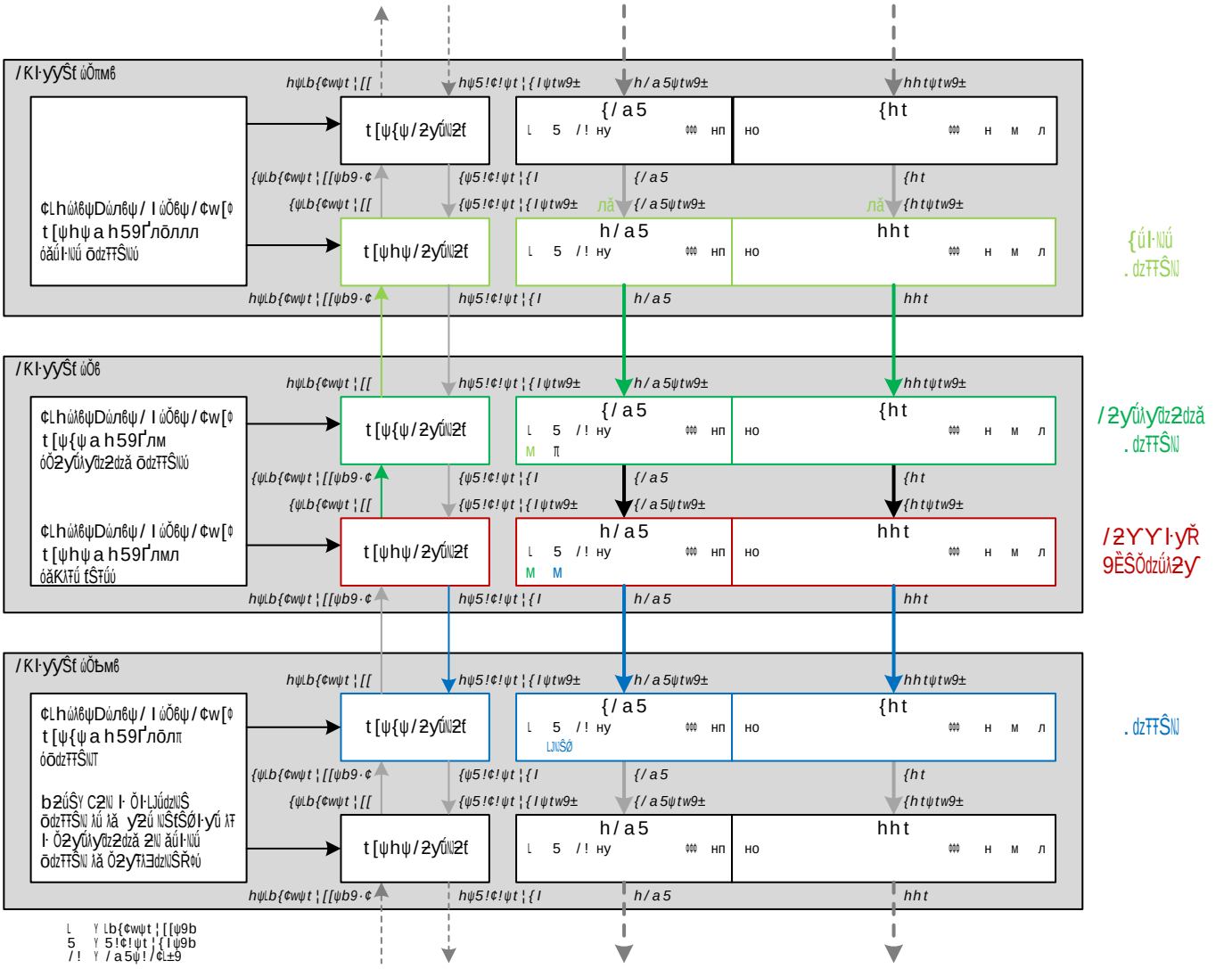
Cyclic buffer usage: the S/O resource in one channel is in use in a cyclic manner. Instructions will be loaded on termination out of the alternate resource of the cyclic buffer.

Figure *187 "Example of Instruction and Capture Buffer Usage"* gives an overview about how the buffer resources can be used. The example shows a shift instruction operating in the O resource of channel [c]. An instruction buffer of length 2 is setup with the S resource in channel [c] and the O resource in channel [c−1]. On each termination of a shift instruction the received data will be captured in the capture buffer build up in the S resource of channel [c+1].

The signals *S_INSTR_PULL* / *O_INSTR_PULL* are the request signals for fetching new data out of the instruction buffer.

The signals *S_DATA_PUSH* / *O_DATA_PUSH* are the request signals for storing data in the capture buffer.

Figure 187 Example of Instruction and Capture Buffer Usage



Instruction termination:
If executed instructions are terminating, it is very usual that based on this condition a programmable core has to react in a certain way depending on the application requirements. Examples are:

- Event on a matching compare; application read signal level or sets up a new compare instruction

- Event on a terminated shift in operation; application reads and processes the received data
- Event on occurrence of the empty state of an instruction buffer resource; application refills the instruction buffer with the next instruction

To support this, the signal *PL_EVT* [c:c] can be used. This signal can be selected by the Trigger Output Control block, *PL_TRIG_OUT* [c:c] Control block, PL Resolution Selection block or the IRQ Control block independently. This allows to setup triggers in the TIO or generate interrupts to a programmable core.

Depending on the configured values of **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** and **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** the source of *PL_EVT* can vary.

In the next table, the *PL_EVT* [c:c] selection is shown depending on the configured modes of the channel resources.

Note:

O_BUFFER_EMPTY , *S_BUFFER_EMPTY* and *INSTR_END* are internal conditions which are described later.

Table 65 PL_EVT[c:c] Selection Dependent On TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE / TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE

TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE	TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE	S resource	O resource	*PL_EVT* [c:c] condition
0b00 | 0b01	0b000 | 0b001	Start buffer | Continue buffer	Start buffer | Continue buffer	*O_BUFFER_EMPTY* or *S_BUFFER_EMPTY*
0b1–	0b000 | 0b001	Counter	Start buffer | Continue buffer	*O_BUFFER_EMPTY*
0b00 | 0b01	0b01–	Start buffer | Continue buffer	Shift	*INSTR_END* of O resource
0b1–	0b01–	Counter	Shift	*INSTR_END* of O resource
0b00 | 0b01	0b10– | 0b110	Start buffer | Continue buffer	Compare	*INSTR_END* of O resource
0b1–	0b10– | 0b110	Counter	Compare	*INSTR_END* of O resource
0b00 | 0b01	0b111	Start buffer | Continue buffer	Capture	*INSTR_END* of O resource
0b1–	0b111	Counter	Capture	*INSTR_END* of O resource

Note:

In case of the O resource is configured as shift, compare or capture, the S resource does not generate any *PL_EVT* .

In the following chapters, the details of all available functions and their instructions are described. Each instruction can be controlled by trigger events. The next chapter describes how trigger events can be enabled.

27.5.10.2 Trigger Event Generation for S Resource and O Resource

The signal *TRIG_OUT* [c:c] can be used to provide triggers to the instructions which are in operation in the S / O resource. The following 3 trigger signals can be enabled to use *TRIG_OUT* [c:c].

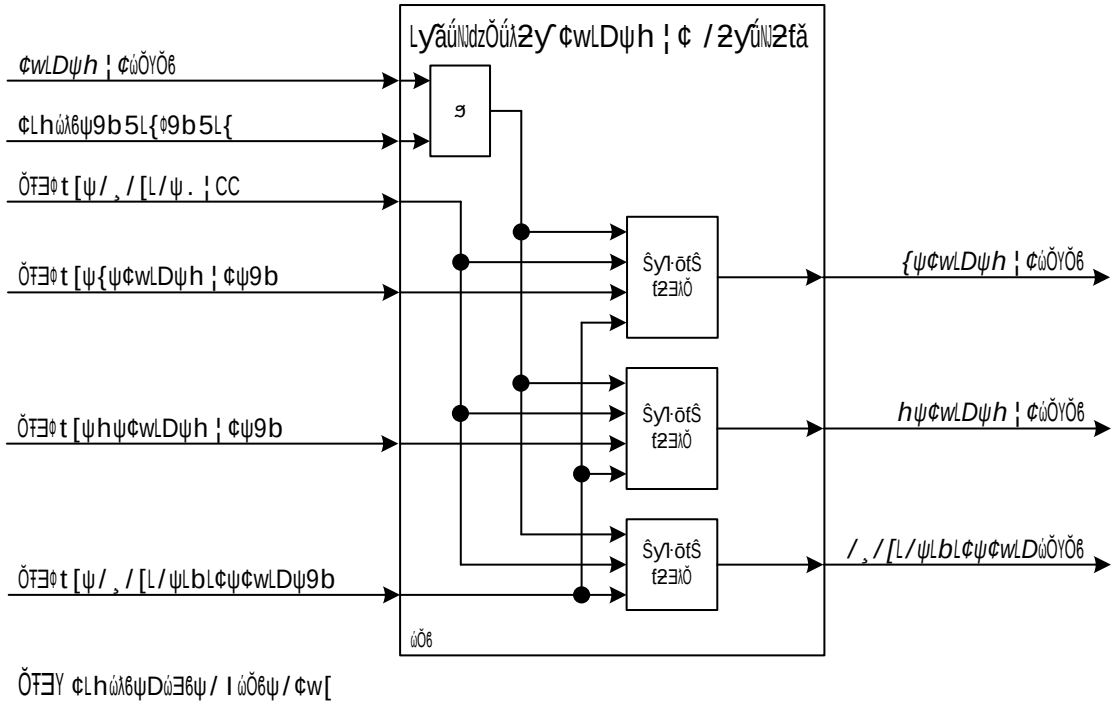
Following trigger signals are available:

- *S_TRIG_OUT* [c:c] : trigger event used in S resource
- *O_TRIG_OUT* [c:c] : trigger event used in O resource
- *CYCLIC_INIT_TRIG* [c:c]: init trigger event used in cyclic buffer operation

Trigger generation for the above signals is possible with the following configuration bits:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_S_TRIG_OUT_EN**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_O_TRIG_OUT_EN**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN**

Figure 188 Instruction Trigger Generation



ŎŦ∃Y ¢Lhώλ ß ψDώ∃ß ψ/ I ώŎß ψ/ ¢w[

The *TRIG_OUT* [c:c] input signal is gated by **TIO[i]_ENDIS.CH[x]** . This prevents that TIO plus functionality of the corresponding channel [c] is triggered by a trigger pulse (*S_TRIG_OUT* [c:c], *O_TRIG_OUT* [c:c] or *CYCLIC_INIT_TRIG* [c:c]) while the channel is disabled.

Table 66 S_TRIG_OUT[c:c] Enable Function

TRIG_OUT [c:c] AND TIO[i]_ENDIS.CH[x]	TIO[i]_G[g]_CH[c]_CTRL.PL_S_TRIG_OUT_EN	TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF	TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN	output *S_TRIG_OUT* [c:c]
0b0	0b-	0b-	0b-	0b0
0b1	0b0	0b-	0b-	0b0
0b1	0b-	0b1	0b1	0b0
0b1	0b1	0b0	0b-	0b1
0b1	0b1	0b1	0b0	0b1

Table 67 O_TRIG_OUT[c:c] Enable Function

TRIG_OUT [c:c] AND TIO[i]_ENDIS.CH[x]	TIO[i]_G[g]_CH[c]_CTRL.PL_O_TRIG_OUT_EN	TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF	TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN	output *O_TRIG_OUT* [c:c]
0b0	0b-	0b-	0b-	0b0
0b1	0b0	0b-	0b-	0b0
0b1	0b-	0b1	0b1	0b0
0b1	0b1	0b0	0b-	0b1
0b1	0b1	0b1	0b0	0b1

Table 68 CYCLIC_INIT_TRIG[c:c] Enable Function

TRIG_OUT [c:c] AND TIO[i]_ENDIS.CH[x]	TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF	TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN	output *CYCLIC_INIT_TRIG* [c:c]
0b0	0b-	0b-	0b0
0b1	0b0	0b-	0b0
0b1	0b1	0b0	0b0
0b1	0b1	0b1	0b1

BOSCH

27.5.10.3 Channel Chain

For many applications, more than one channel is needed to realize the required function.
The design is organized in channels and channel groups and some signals are routed from channel to channel as shown in figures *166 "TIO Block Diagram"* and *187 "Example of Instruction and Capture Buffer Usage"* .

These signals are:

- *OCMD*
- *OOP*
- *O_BUFF_FILLED*
- *O_INSTR_PULL*
- *O_DATA_PUSH*

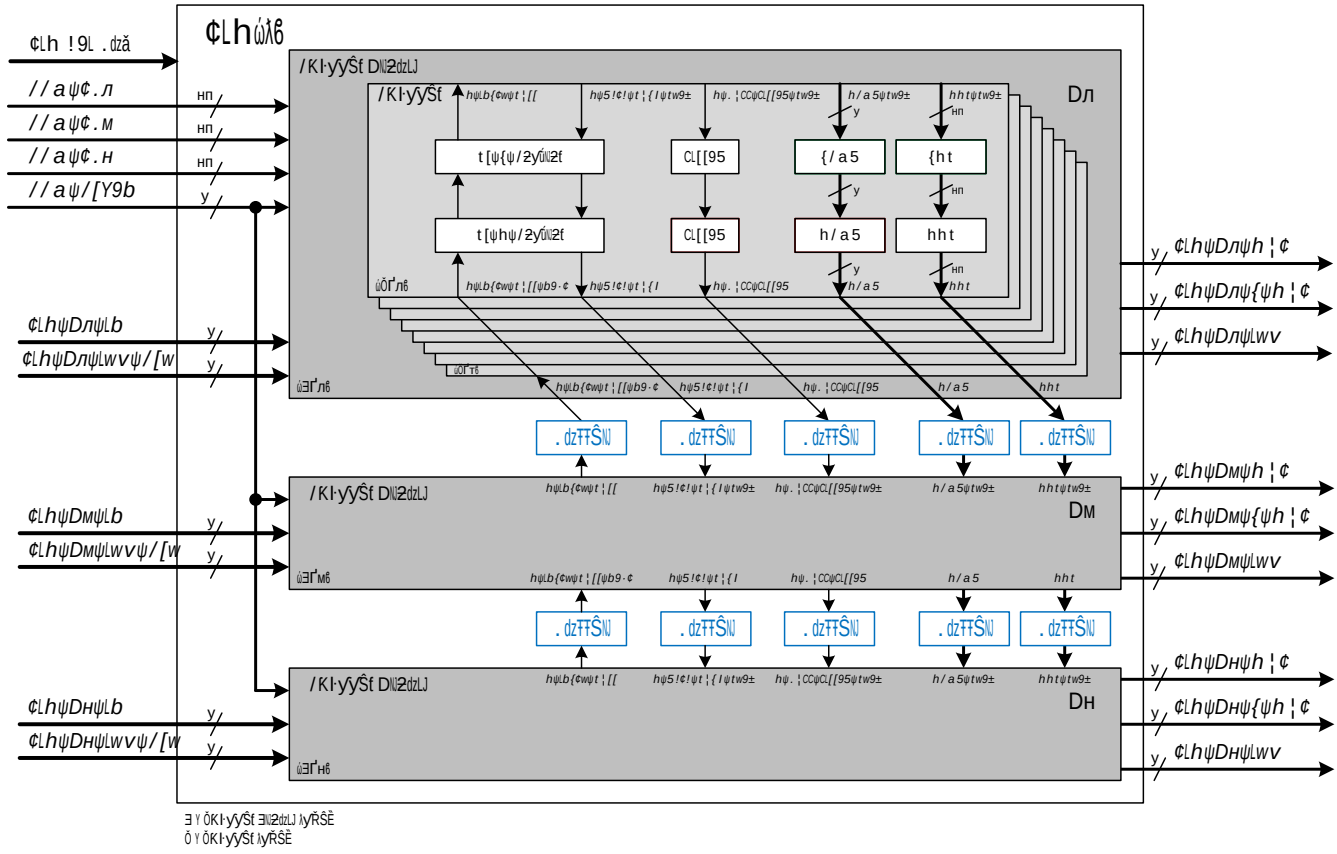
Some of these signals are combinational paths without any storage element. To prevent problems during chip design the above five signals are buffered between the channel groups as shown in figure *189 "Channel Chain with Buffers"* .

If functions use more than one channel and the used channels are located in different channel groups, there is an additional delay of one cluster clock cycle which has to be considered by the application.

Note:

If instruction reload feature is used beyond a channel group limit, then the above-mentioned signals are buffered and only *PL_UPDATE* [c:c] <= 1/3 cluster clock frequency is possible.
If instruction reload feature is used beyond both channel group limits only *PL_UPDATE* [c:c] <= 1/5 cluster clock frequency is possible.

Figure 189 Channel Chain with Buffers



27.5.10.4 TIO Plus Terminology and States of an Instruction

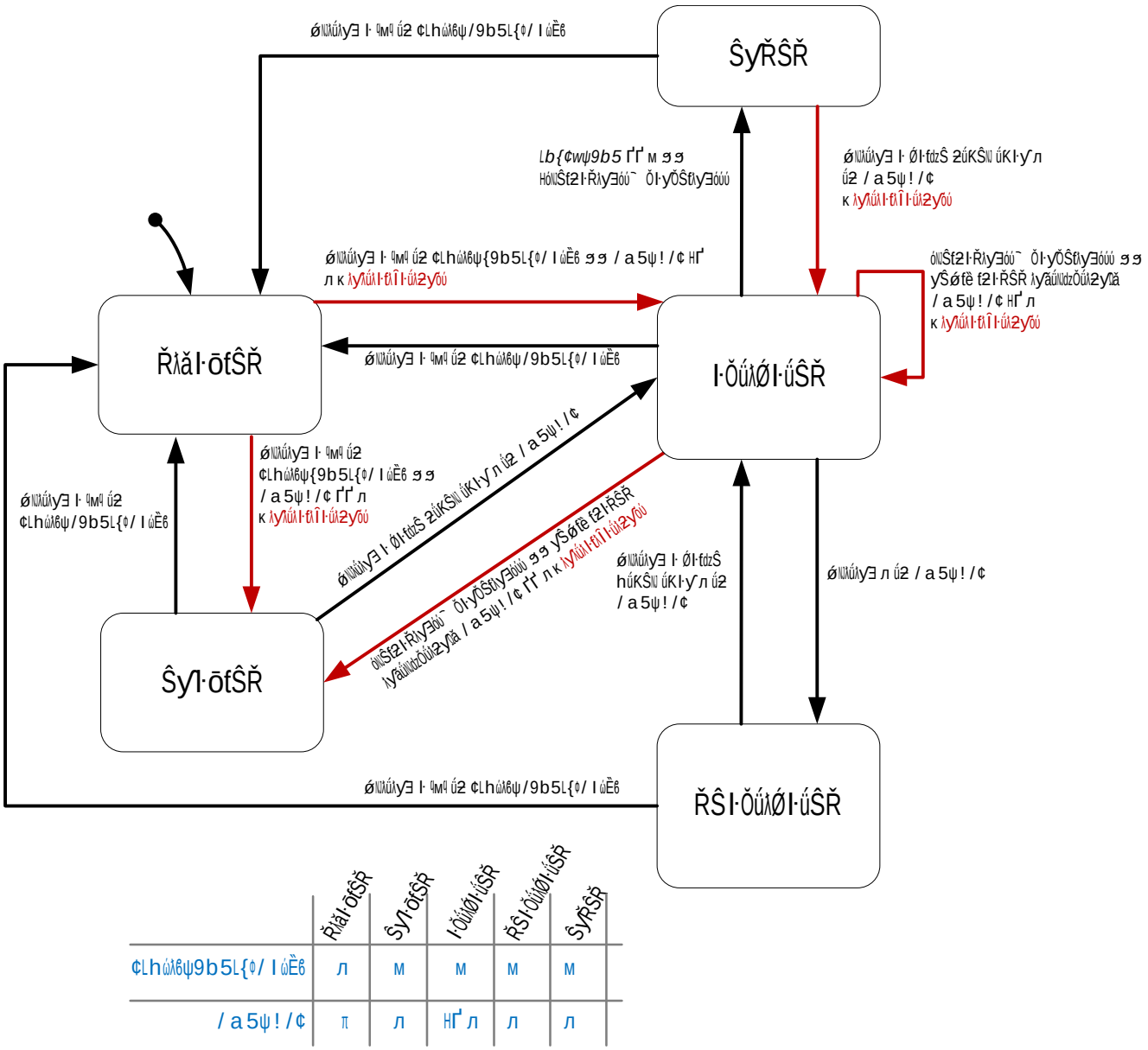
For the following discussion, let CMD_ACT ∈ { **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** , **TIO[i]_G[g]_CH[c]_SINST_COUNT12.CMD_ACTIVE** , **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CMD_ACTIVE** , **TIO[i]_G[g]_CH[c]_OINST_SHIFT.CMD_ACTIVE** , **TIO[i]_G[g]_CH[c]_OINST_CAP.CMD_ACTIVE_CC** , **TIO[i]_G[g]_CH[c]_OINST_COMP.CMD_ACTIVE_CC** , **TIO[i]_G[g]_CH[c]_OINST_COMP12.CMD_ACTIVE_CC** , **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** , **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** , **TIO[i]_G[g]_CH[c]_OCMD_CAP.CMD_ACTIVE_CC** , **TIO[i]_G[g]_CH[c]_OCAPTURE.CMD_ACTIVE_CC** , **TIO[i]_G[g]_CH[c]_OCAPTURE12.CMD_ACTIVE_CC** }. The following

table introduces some terms that are used throughout the chapter. The table also summarizes the states of a TIO Plus instruction (see *190 "States of a TIO Plus Instruction"*).

Table 69 Terms and States of an TIO Plus Instruction

Term / state	Description
Instruction reloading	Instruction reloading occurs when the instruction that has just terminated (termination is marked by the assertion of *INSTR_END*) is frozen, or when the instruction terminates while the channel that hosts the instruction has the cyclic buffer functionality enabled (**TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = '1'), or when the instruction that has just terminated is configured to pull an instruction from the preceding resource.
Instruction canceling	Instruction cancelling occurs when the channel that hosts the instruction has the cyclic buffer functionality enabled (**TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = '1'), and an init trigger or an exchange trigger is asserted. Upon instruction cancelling, the two instructions in the channel might be exchanged (*INSTR_END* is not asserted in contrast to instruction reloading).
Disabled state	An instruction transits to the 'disabled' state when the channel that hosts the instruction is disabled (by e.g. writing a one to **TIO[i]_CENDIS.CH** [x:x]). Transition from any state to 'disabled' state is allowed.
Enabled state	An instruction transits from the 'disabled' state to the 'enabled' state when the channel that hosts the instruction is enabled (by e.g. writing a one to **TIO[i]_SENDIS.CH** [x:x]) on an instruction that has its CMD_ACT at zero. An instruction transits from 'activated' state to the 'enabled' state when there is instruction reloading or instruction cancelling, and the newly loaded instruction has its CMD_ACT at zero.
Activated state	This is the state in which the execution of the instruction takes place. An instruction transits from 'enabled'/'ended'/'deactivated' state to the 'activated' state when the instruction is made active (by writing a value other than zero to CMD_ACT). An instruction transits from 'disabled' state to 'activated' state when the channel is enabled (by e.g. writing a one to **TIO[i]_SENDIS.CH** [x:x]) on an instruction that has its CMD_ACT at a value other than zero. An instruction remains in the 'activated' state when there is instruction reloading or instruction cancelling, and the newly loaded instruction has its CMD_ACT at a value other than zero.
Ended state	An instruction transits to the 'ended' state when the current state is 'activated' and an instruction terminates (*INSTR_END* is asserted) and there is neither instruction reloading nor instruction canceling.
Deactivated state	An instruction transits from 'activated' to 'deactivated' when a zero is written to CMD_ACT. In this state the instruction execution is suspended.
Instruction initialization	Any transition to 'enabled' state, transiting from 'disabled'/'ended' state to 'activated' state, and transiting from 'activated' state to the same state (due to instruction reloading or instruction cancelling) is referred to as the initialization of an instruction. Upon the initialization of an instruction, internal states are set to initial values that may differ from an instruction to another. It is important to emphasize here that not every transition to the 'activated' state shall result in the initialization of an instruction.

Figure 190 States of a TIO Plus Instruction



27.5.10.5 O Resource Shift Instructions

This instruction allows to set up a bit shift with variable length, this can be used to sample input signal values or generate a serial output based on a defined shift resolution.

Following inputs are taken into account:

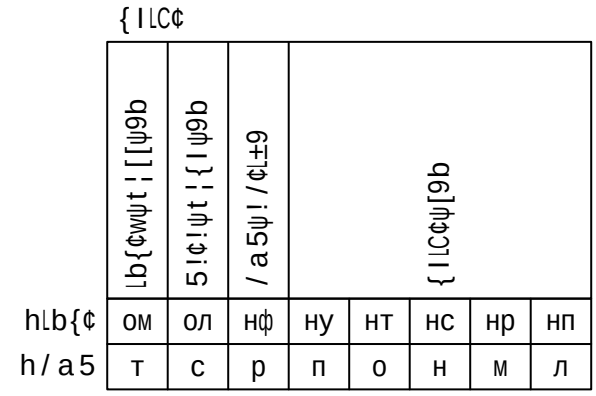
- *S_OUT* [c:c] : value of **TIO[i]_S.CH[x]** x = 8*g+c
- *PL_UPDATE* [c:c] : resolution for operation
- *O_TRIG_OUT* [c:c] : trigger event, activates an instruction end / reload; see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*

This mode is selected with **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b010 | 0b011.

In the O resource shift mode the following bit fields in the command register exist:

- **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.SHIFT_LEN** : number of bits to shift [1 ,...,31]; A shift length of 0 is not allowed.
- **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** : command active (shift in progress / shift stopped)
- **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.INSTR_PULL_EN** : enable instruction pull from instruction buffer
- **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.DATA_PUSH_EN** : enable data push to capture buffer

Figure 191 CMD Register Layout – Shift Instruction



They can be accessed via the alias register names:

- **TIO[i]_G[g]_CH[c]_OCMD_SHIFT**
- **TIO[i]_G[g]_CH[c]_OINST_SHIFT**

The bit field **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** in the command register controls the shift instruction:

- 0 : shift stopped
- 1 : shift active

The execution of shift instructions can be influenced with the following configuration bits, the corresponding functions will be explained in the referred sections:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0 : see chapter *27.5.10.5.2 "Shift INOUT Instruction"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1 : see chapter *27.5.10.5.5 "Instruction Termination With Instruction Freeze"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 : see chapter *27.5.10.5.7 "Instruction Termination With Cyclic Buffer Usage"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN** = 1 : see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_O_TRIG_OUT_EN** = 1 : see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 1 instruction buffer usage : see chapter *27.5.10.5.6 "Instruction Termination With Instruction Reload"*
- **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 1 capture buffer usage : see chapter *27.5.10.5.8 "Capture Buffer Usage On Instruction Termination"*

For the functional description in the next chapters (*27.5.10.5.1 "Shift IN Instruction"*) is assumed:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1
- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0
- **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 0
- **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 0

Note:

There is a delay of 4 clock cycles between the primary input signal *TIO_G[g]_IN* [c:c] and the **TIO[i]_G[g]_CH[c]_OINST.OP** register which stores the received bits during a shift instruction: 2 system clock cycles from *TIO_G[g]_IN* [c:c] to *SYNC_OUT* [c:c] by the SYNC block; 1 clock cycle from *SYNC_OUT* [c:c] to register S[c] and 1 clock cycle from register S[c] to **TIO[i]_G[g]_CH[c]_OINST.OP** depending on the configuration of the resolution selection block (*UPDATE* [c:c]=1 constantly).

Note:

There is a delay between the **TIO[i]_G[g]_CH[c]_OINST.OP** register bit which is shifted out and the *TIO_G[g]_OUT* [c:c] of 1 clock cycle.

Note:

TIO[i]_G[g]_CH[c]_SHIFTCNT is reset to 0x0 if a **TIO[i]_G[g]_CH[c]_OINST.CMD** is loaded (e.g. by *CYCLIC_INIT_TRIG* [c:c], *PL_TRIG_OUT* [c:c], *O_INSTR_PULL_NEXT* , *O_TRIG_OUT* [c:c] or a write access to **TIO[i]_G[g]_CH[c]_OINST.CMD**).

Attention:

It is mandatory for a shift instruction which uses any reload option (incl. freeze) that *PL_UPDATE* [c:c] <= 1/2 cluster clock frequency. Otherwise, every time the **TIO[i]_G[g]_CH[c]_OINST.OP** register is reloaded (e.g. shift of more than 24bit), the shifted out value is doubled and the shifted in value is not stored.
Further *PL_UPDATE* [c:c] must be single pulses of cluster clock frequency.

Note:
At the end of a shift INOUT instruction, the last bit which is shifted out is set on the output signal. Thus, the instruction termination is the start of the bit time, whereas the next *PL_UPDATE* [c:c] event marks the end of the bit time of the last shifted out bit. Be aware that during this bit time, the bit can be disturbed. For example, if the software writes to the output register bit **TIO[i]_O** .

27.5.10.5.1 Shift IN Instruction

The Shift IN instruction is selected by setting **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** to 1. It implements a serial shift, the number of bits to be shifted is defined with bit field **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.SHIFT_LEN** in the command register.

The shift function gets activated as soon as bit field **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** = 1 is set.

The local register **TIO[i]_G[g]_CH[c]_SHIFTCNT** represents the number of currently shifted bits.

This counter value **TIO[i]_G[g]_CH[c]_SHIFTCNT** will increment by one on the *PL_UPDATE* [c:c] = 1 resolution.

If **TIO[i]_G[g]_CH[c]_SHIFTCNT** >= **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.SHIFT_LEN** is true, then the shift command terminates, *INSTR_END* =1 is set.

Another possibility to terminate a shift instruction is if signal *O_TRIG_OUT* [c:c]=1, then the shift command terminates, *INSTR_END* =1 is set.

Until the *INSTR_END* = 1 occurs, on each clock cycle when *PL_UPDATE* [c:c] = 1 is fulfilled the value of signal *S_OUT* [c:c] gets stored.

A shift command can be canceled if **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 and (*CYCLIC_INIT_TRIG* [c:c]=1 or *PL_TRIG_OUT* [c:c]=1 or *O_INSTR_PULL_NEXT* =1).

If *CYCLIC_INIT_TRIG* [c:c] = 1 and **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 is fulfilled, then the shift command is canceled (no *INSTR_END* =1 is set). The command register reload actions explained in chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"* and *27.5.10.13.2 "Init Trigger Behavior"* are executed.

If *PL_TRIG_OUT* [c:c] = '1' OR *O_INSTR_PULL_NEXT* = '1' and *CYCLIC_INIT_TRIG* [c:c]=0 and **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 is fulfilled, then the shift command is canceled (no *INSTR_END* =1 is set). The command register reload actions explained in chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"* and *27.5.10.13.1 "Exchange Trigger Behavior"* are executed.

If the shift command terminates (*INSTR_END* =1) and *CYCLIC_INIT_TRIG* [c:c]=0 and *PL_TRIG_OUT* [c:c]=0 and *O_INSTR_PULL_NEXT* =0 and **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 is fulfilled, then *INSTR_END* =1 is set. Due to **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 the command register reload actions explained in chapter *27.5.10.5.7 "Instruction Termination With Cyclic Buffer Usage"* are executed.

In case of an init trigger event (*CYCLIC_INIT_TRIG* [c:c]=1) occurs in the same clock cycle as an exchange trigger event (*PL_TRIG_OUT* [c:c]=1 or *O_INSTR_PULL_NEXT* =1) or a shift command termination (*INSTR_END* =1), then the command is canceled (*INSTR_END* =0) and the init trigger event behavior will be applied.

In case of an exchange trigger event (*PL_TRIG_OUT* [c:c]=1 or *O_INSTR_PULL_NEXT* =1) occurs in the same clock cycle as a shift command termination (*INSTR_END* =1), then the command is canceled (*INSTR_END* =0) and the exchange trigger behavior will be applied.

On every writing of the command register **TIO[i]_G[g]_CH[c]_OCMD_SHIFT** / **TIO[i]_G[g]_CH[c]_OINST_SHIFT** via the configuration interface the internal register **TIO[i]_G[g]_CH[c]_SHIFTCNT** = 0 will be set.

Shift in left instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b010; **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1

S_OUT [c:c] will be stored in the LSB of operand **TIO[i]_G[g]_CH[c]_OINST.OP** of the instruction:
TIO[i]_G[g]_CH[c]_OINST.OP [23:1] = **TIO[i]_G[g]_CH[c]_OINST.OP** [22:0]; **TIO[i]_G[g]_CH[c]_OINST.OP** [0:0] = *S_OUT* [c:c]

Shift in right instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b011; **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1

S_OUT [c:c] will be stored in the MSB of operand **TIO[i]_G[g]_CH[c]_OINST.OP** of the instruction:
TIO[i]_G[g]_CH[c]_OINST.OP [22:0] = **TIO[i]_G[g]_CH[c]_OINST.OP** [23:1]; **TIO[i]_G[g]_CH[c]_OINST.OP** [23:23] = *S_OUT* [c:c]

For a Shift IN, it is mandatory to set **TIO[i]_INPUT_MODE** [c:c]=1 of the corresponding channel.

27.5.10.5.2 Shift INOUT Instruction

The Shift OUT instruction is selected by setting **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** to 0. In addition to the shift in functionality (*27.5.10.5.1 "Shift IN Instruction"*), the shift out function is enabled.

For a shift instruction, it is mandatory to set **TIO[i]_INPUT_MODE** [c:c]=1, **TIO[i]_CYCLIC_MODE** [c:c]=1 and **TIO[i]_INVERT** [c:c]=0 of the corresponding channel c. Otherwise, the behavior of output signal *O_OUT* [c:c] is undefined.

Until the *INSTR_END* = 1 occurs, on each clock cycle when *PL_UPDATE* [c:c] = 1 the signal *O_OUT* [c:c] gets loaded with the corresponding bit in the operand **TIO[i]_G[g]_CH[c]_OINST.OP** of the instruction.

Shift inout left instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b010; **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0

S_OUT [c:c] will be stored in the LSB of operand **TIO[i]_G[g]_CH[c]_OINST.OP** of the instruction:
TIO[i]_G[g]_CH[c]_OINST.OP [23:1] = **TIO[i]_G[g]_CH[c]_OINST.OP** [22:0]; **TIO[i]_G[g]_CH[c]_OINST.OP** [0:0] = *S_OUT* [c:c]

The MSB of operand **TIO[i]_G[g]_CH[c]_OINST.OP** of the instruction defines the new value of *O_OUT* [c:c].

- *PL_O_LOAD* [c:c] = *PL_UPDATE* [c:c];
- *PL_O_VAL* [c:c] = **TIO[i]_G[g]_CH[c]_OINST.OP** [23:23]

Shift inout right instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b011; **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0

S_OUT [c:c] will be stored in the MSB of operand **TIO[i]_G[g]_CH[c]_OINST.OP** of the instruction:
TIO[i]_G[g]_CH[c]_OINST.OP [22:0] = **TIO[i]_G[g]_CH[c]_OINST.OP** [23:1]; **TIO[i]_G[g]_CH[c]_OINST.OP** [23:23] = *S_OUT* [c:c]

The LSB of operand **TIO[i]_G[g]_CH[c]_OINST.OP** of the instruction defines the new value of *O_OUT* [c:c].

- *PL_O_LOAD* [c:c] = *PL_UPDATE* [c:c];
- *PL_O_VAL* [c:c] = **TIO[i]_G[g]_CH[c]_OINST.OP** [0:0]

27.5.10.5.3 PL_EVT Generation

For every shift instruction the *PL_EVT* [c:c] condition is defined as:
PL_EVT [c:c] = *INSTR_END*

The *PL_EVT* [c:c] is set on each instruction termination.

27.5.10.5.4 Instruction Termination Without Instruction Reload

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_-OCMD.INSTR_PULL_EN** = 0

If the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** = 0 (STOP_INSTRUCTION) is set.
A next instruction is not pulled if *INSTR_PULL* = 0.

A next instruction can only be issued by writing **TIO[i]_G[g]_CH[c]_OCMD** with a new instruction command or enable the last instruction again by setting **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** = 1.

27.5.10.5.5 Instruction Termination With Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_-OCMD.INSTR_PULL_EN** = 0|1

If the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_OCMD_SHIFT** will remain unchanged.
A next instruction is not pulled, *INSTR_PULL* = 0. On initialization, the internal register **TIO[i]_G[g]_CH[c]_SHIFTCNT** will be set to 0.

In case **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** = 1, the command is executed immediately. This allows to shift in data in an endless manner.

Note:

Freezing does not make sense if the shift out function is enabled (**TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0), because the initial shift out pattern stored in **TIO[i]_G[g]_CH[c]_OINST.OP** gets destroyed by the shift in pattern. This leads to the fact that the pattern stored in **TIO[i]_G[g]_CH[c]_OINST.OP** at *INSTR_END* = 1 will be used as the shift out pattern for the next instruction.

27.5.10.5.6 Instruction Termination With Instruction Reload

Allow pulling from an instruction buffer.
Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_-OCMD.INSTR_PULL_EN** = 1;

If the instruction terminates with *INSTR_END* = 1, the next instruction will be loaded from instruction buffer:
TIO[i]_G[g]_CH[c]_OINST.CMD = **TIO[i]_G[g]_CH[c]_SINST.CMD** ; **TIO[i]_G[g]_CH[c]_OINST.OP** = **TIO[i]_G[g]_CH[c]_SINST.OP**
A next instruction is pulled, *INSTR_PULL* = *INSTR_END* . On initialization, the internal register **TIO[i]_G[g]_CH[c]_SHIFTCNT** is set to 0.

The new loaded shift instruction is immediately executed in case **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** = 1.

27.5.10.5.7 Instruction Termination With Cyclic Buffer Usage

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1

If the instruction terminates with *INSTR_END* = 1, a next instruction will be loaded from the cyclic instruction buffer:
TIO[i]_G[g]_CH[c]_OINST_SHIFT (t+1) = **TIO[i]_G[g]_CH[c]_SINST** (t)
TIO[i]_G[g]_CH[c]_SINST (t+1) = **TIO[i]_G[g]_CH[c]_OINST_SHIFT** (t)
On initialization, the internal register **TIO[i]_G[g]_CH[c]_SHIFTCNT** is set to 0.

The new loaded shift instruction is immediately executed in case **TIO[i]_G[g]_CH[c]_OCMD_SHIFT.CMD_ACTIVE** = 1.

This configuration uses the cyclic instruction buffer of the channel resource (details see chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"*), a next instruction is never pulled via the signal *INSTR_PULL* from the S resource. Signal *INSTR_PULL* = 0 never changes.

27.5.10.5.8 Capture Buffer Usage On Instruction Termination

No capture buffer in use:

Configuration in use: **TIO[i]_G[g]_CH[c]_OINST.DATA_PUSH_EN** = 0

If the instruction terminates with *INSTR_END* = 1, no data push request is issued.
DATA_PUSH = 0

Capture buffer in use:

To enable that data from the resource of the channel [c] can be captured in the resource of the next channel [c+1], the S resource of the next channel has to be configured as a continue buffer.
Configuration in use: **TIO[i]_G[g]_CH[c]_OINST.DATA_PUSH_EN** = 1; **TIO[i]_G[g]_CH[c+1]_CTRL.PL_S_MODE** = 0b01

If the instruction terminates with *INSTR_END* = 1, a data push request is issued.
DATA_PUSH = *INSTR_END* .

Note:
In case of **TIO[i]_G[g]_CH[c]_OINST.DATA_PUSH_EN** = 1; **TIO[i]_G[g]_CH[c+1]_CTRL.PL_S_MODE** = 0b10 | 0b11 the behavior of the S resource is undefined. The S resource will not perform the counter instruction correctly nor the capture buffer functionality is operated correctly.

27.5.10.6 O Resource Capture Instructions

This instruction allows to capture, based on a programmable event, signal values together with a 24–bit time stamp.

Following inputs are taken into account:

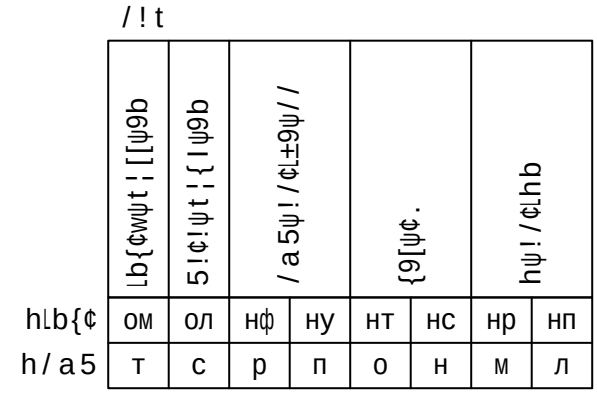
- *PL_TRIG_OUT* [c:c]: capture trigger event
- *S_OUT* [c:c] : value of **TIO[i]_S.CH[x]** x= 8*g+c
- *O_OUT* [c:c] : value of **TIO[i]_O.CH[x]** x= 8*g+c

This mode is selected with **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b111.

In the O resource capture mode the following bit fields in the command register exist:

- **TIO[i]_G[g]_CH[c]_OCMD_CAP.O_ACTION** : action on *O_OUT* [c:c] which has to be taken when capture occurs
- **TIO[i]_G[g]_CH[c]_OCMD_CAP.SEL_TB** : defines source which shall be captured
- **TIO[i]_G[g]_CH[c]_OCMD_CAP.CMD_ACTIVE_CC** : command active (capture active wait for event / capture stopped)
- **TIO[i]_G[g]_CH[c]_OCMD_CAP.INSTR_PULL_EN** : enable instruction pull from instruction buffer
- **TIO[i]_G[g]_CH[c]_OCMD_CAP.DATA_PUSH_EN** : enable data push to capture buffer

Figure 192 CMD Register Layout – Capture Instruction



They can be accessed via the multiview register names:

- **TIO[i]_G[g]_CH[c]_OCMD_CAP**
- **TIO[i]_G[g]_CH[c]_OINST_CAP**

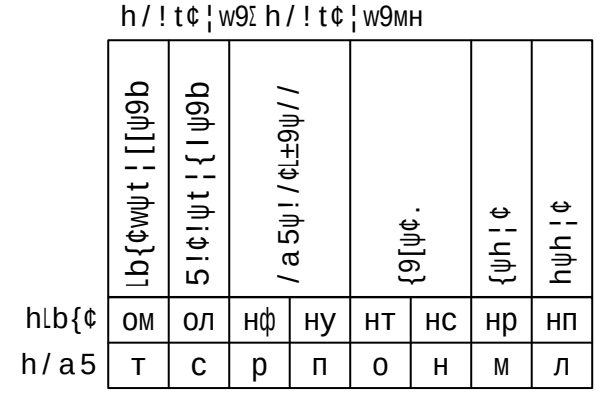
The bit field **TIO[i]_G[g]_CH[c]_OCMD_CAP.CMD_ACTIVE_CC** in the command register controls the capture instruction:

- 0b00 : capture occurred / stopped
- 0b10 : capture instruction active, wait for capture trigger

If a capture occurs the captured values of the capture instruction can be accessed by the bit fields in **TIO[i]_G[g]_CH[c]_OCAPTURE** :

- **TIO[i]_G[g]_CH[c]_OCAPTURE.OP** : captured value of the selected TB
- **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** : captured value of signal *S_OUT* [c:c]
- **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** : captured value of signal *O_OUT* [c:c]
- **TIO[i]_G[g]_CH[c]_OCAPTURE.SEL_TB** : selected TB at capture
- **TIO[i]_G[g]_CH[c]_OCAPTURE.CMD_ACTIVE_CC** : mode which was active at capture
- **TIO[i]_G[g]_CH[c]_OCAPTURE.INSTR_PULL_EN** : enable instruction pull from instruction buffer
- **TIO[i]_G[g]_CH[c]_OCAPTURE.DATA_PUSH_EN** : enable data push to capture buffer

Figure 193 CMD Register Layout – Capture Instruction Has Terminated



The execution of capture instructions can be influenced with the following configuration bits, the corresponding functions will be explained in the referred sections:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0 : see chapter *27.5.10.6.3 "Channel Output O_OUT[c:c] Controlled By Last Value"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 1 : see chapter *27.5.10.6.4 "Channel Output O_OUT[c:c] Controlled by S_OUT[c:c] Value"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1 : see chapter *27.5.10.6.7 "Instruction Termination With Instruction Freeze"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 : see chapter *27.5.10.6.9 "Instruction Termination With Cyclic Buffer Usage"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN** = 1 : see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 1 instruction buffer usage : see chapter *27.5.10.6.8 "Instruction Termination With Instruction Reload"*

- **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 1 capture buffer usage : see chapter *27.5.10.6.10 "Capture Buffer Usage On Instruction Termination"*

For the functional description in the next chapters (*27.5.10.6.1 "Select Source Of Capture Instruction"* – *27.5.10.6.2 "Capture Instruction"*) is assumed:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 0
- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0
- **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 0
- **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 0

27.5.10.6.1 Select Source Of Capture Instruction

The data which shall be captured is configured with the bit field **TIO[i]_G[g]_CH[c]_OCAPTURE.SEL_TB** in the command register. (see table *70 "TIO[i]_G[g]_CH[c]_OCAPTURE.SEL_TB Definition"*)

Table 70 TIO[i]_G[g]_CH[c]_OCAPTURE.SEL_TB Definition

Channel c TIO[i]_G[g]_CH[c]_OCAPTURE.SEL_TB	x = g*8 +c; g=floor(x/8) Source *TB* of channel c
0b00	*RS_TB0[g]*
0b01	*RS_TB1[g]*
0b10	*RS_TB2[g]*
0b11	**TIO[i]_G[g]_CH[c]_SINST.OP**

27.5.10.6.2 Capture Instruction

A capture instruction gets activated as soon as bit field **TIO[i]_G[g]_CH[c]_OCAPTURE.CMD_ACTIVE_CC** = 0b10 is set.

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 and *CYCLIC_INIT_TRIG* [c:c]=1 is fulfilled, then the current capture command is canceled (no *INSTR_END* =1 is set) and the command register reload actions explained in chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"* and *27.5.10.13.2 "Init Trigger Behavior"* are executed.

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 and *CYCLIC_INIT_TRIG* [c:c]=0 and *O_INSTR_PULL_NEXT* =1 is fulfilled, then the current capture command is canceled (no *INSTR_END* =1 is set) and the command register reload actions explained in chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"* and *27.5.10.13.1 "Exchange Trigger Behavior"* are executed.

In all other cases an active capture instruction waits until the capture trigger event *PL_TRIG_OUT* [c:c]=1 occurs.

Following data is captured on a capture trigger event *PL_TRIG_OUT* [c:c]=1:

- **TIO[i]_G[g]_CH[c]_OCAPTURE.OP** = *TB*
- **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** = *S_OUT* [c:c]
- **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** = *O_OUT* [c:c]

The instruction terminates, *INSTR_END* =1 is set.
TIO[i]_G[g]_CH[c]_OCAPTURE.CMD_ACTIVE_CC = 0b00 will be cleared.

At any point in time a capture instruction can be deactivated by clearing **TIO[i]_G[g]_CH[c]_OCAPTURE.CMD_ACTIVE_CC** = 0b00.

Any capture triggers *PL_TRIG_OUT* [c:c]=1, occurring while **TIO[i]_G[g]_CH[c]_OCAPTURE.CMD_ACTIVE_CC** =0b00 will be ignored. No capture will be initiated.

Be aware that the capture instruction is split into 2 phases:

- Instruction active, waiting for capture trigger; controlled with register alias **TIO[i]_G[g]_CH[c]_OCMD_CAP** . As long as **TIO[i]_G[g]_CH[c]_OCMD_CAP.CMD_ACTIVE_CC** = 0b10 is fulfilled the instruction is active and has not terminated.
- With occurrence of a capture event (*PL_TRIG_OUT* [c:c]=1) the second phase is entered; capture results are accessible with register alias **TIO[i]_G[g]_CH[c]_OCAPTURE**

27.5.10.6.3 Channel Output O_OUT[c:c] Controlled By Last Value

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 0

On **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0 the function to control the channel output *O_OUT* [c:c] on occurrence of a capture trigger is enabled.

If **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** is configured as 0 in a capture instruction, it is mandatory to set **TIO[i]_INPUT_MODE** [c:c]=1, **TIO[i]_CYCLIC_MODE** [c:c]=1 and **TIO[i]_INVERT** [c:c]=0 of the corresponding channel c. Otherwise, the behavior of output signal *O_OUT* [c:c] is undefined.

With the bit field **TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION** in the capture instruction register, signal changes in register **TIO[i]_O.CH[x]** of the TIO channel can be initiated. In case of *PL_O_LOAD* [c:c] = 1, **TIO[i]_O.CH[x]** = *PL_O_VAL* [c:c] will be loaded. Following actions can be configured (see table *71 "TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION Definition; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 0"*):

Table 71 TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION Definition; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 0

TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION	*PL_TRIG_OUT* [c:c]	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b––	0	0	–	No change on output **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT**
0b00	1	1	0	Set **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** =0
0b01	1	1	1	Set **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** =1
0b10	1	1	*O_OUT* [c:c]	Hold **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT**
0b11	1	1	not(*O_OUT* [c:c])	Invert **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT**

Note:

If *PL_TRIG_OUT* [c:c] and *CFG_REG_WSTB_FFO* [c:c] are asserted in the same clock cycle, then the actions defined on *O_OUT* [c:c] in Table *71 "TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION Definition; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 0"* shall not be applied. Those actions shall be instead applied on the bit value that has just been written to the storage element over *CFG_REG_DO_FFO* [c:c].

27.5.10.6.4 Channel Output O_OUT[c:c] Controlled by S_OUT[c:c] Value

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 1

If the configuration bit **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** =1 is set, the action on the channel output *O_OUT* [c:c] which is defined by the bit field **TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION** can depend on the value of the signal *S_OUT* [c:c]. This allows to setup a capture instruction which will alter the value of the signal *O_OUT* [c:c] with the capture trigger *PL_TRIG_OUT* [c:c] to the value of *S_OUT* [c:c] or not(*S_OUT* [c:c]).

On **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0 the function to control the channel output *O_OUT* [c:c] on occurrence of a capture trigger is enabled.

If **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** is configured as 0 in a capture instruction, it is mandatory to set **TIO[i]_INPUT_MODE** [c:c]=1, **TIO[i]_CYCLIC_MODE** [c:c]=1 and **TIO[i]_INVERT** [c:c]=0 of the corresponding channel c. Otherwise, the behavior of output signal *O_OUT* [c:c] is undefined.

With the bit field **TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION** in the capture instruction register, signal changes in register **TIO[i]_O.CH[x]** of the TIO channel can be initiated. In case of *PL_O_LOAD* [c:c] = 1, **TIO[i]_O.CH[x]** = *PL_O_VAL* [c:c] will be loaded. Following actions can be configured (see table *72 "TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION Definition; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 1"*):

Table 72 TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION Definition; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 1

TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION	*PL_TRIG_OUT* [c:c]	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b––	0	0	–	No change on output **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT**
0b00	1	1	0	Set **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** =0
0b01	1	1	1	Set **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** =1
0b10	1	1	**TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT**	Set output **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** to the value of **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT**

TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION	*PL_TRIG_OUT* [c:c]	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b11	1	1	not(**TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT**)	Set output **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** to the inverted value of **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT**

27.5.10.6.5 PL_EVT Generation

For every capture instruction the *PL_EVT* [c:c] condition is defined as: *PL_EVT* [c:c]= *INSTR_END* .

The *PL_EVT* [c:c] is set on each instruction termination.

27.5.10.6.6 Instruction Termination Without Instruction Reload

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 0

If the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_OCMD_CAP.CMD_ACTIVE_CC** = 0b00 (STOP_INSTRUCTION) is set.
A next instruction is not pulled, *INSTR_PULL* = 0.

A next instruction can only be issued by writing **TIO[i]_G[g]_CH[c]_OCMD_CAP** with a new instruction command.

27.5.10.6.7 Instruction Termination With Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 0|1; **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 0

If the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_OINST.CMD** will remain unchanged and **TIO[i]_G[g]_CH[c]_OINST.OP** = *TB* will be stored.
A next instruction is not pulled, *INSTR_PULL* = 0.

This results in an endless execution of the command in **TIO[i]_G[g]_CH[c]_OINST.CMD** . This allows to apply the capture instruction in an endless manner.

Note:

Due to freezing, the **TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION** bit field will never change. This prevents that the **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** / **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** values can be captured to the same bits already in use by the bit field **TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION** .

27.5.10.6.8 Instruction Termination With Instruction Reload

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 1; **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** =0

If the instruction terminates with *INSTR_END* = 1, the next instruction will be loaded from instruction buffer:

- **TIO[i]_G[g]_CH[c]_OCMD** = **TIO[i]_G[g]_CH[c]_SCMD**
- **TIO[i]_G[g]_CH[c]_OOP** = *TB*

The new loaded capture instruction is immediately executed and defines the value for the pending **TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION** . This prevents that the **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** / **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** values can be captured to the same bits already in use by the bit field **TIO[i]_G[g]_CH[c]_OINST_CAP.O_ACTION** .

27.5.10.6.9 Instruction Termination With Cyclic Buffer Usage

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1; **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 0

If the instruction terminates and cyclic buffer is enabled, the following actions are executed:

- **TIO[i]_G[g]_CH[c]_OCMD** (t+1) = **TIO[i]_G[g]_CH[c]_SCMD** (t)
- **TIO[i]_G[g]_CH[c]_OOP** (t+1) = *TB* (t)
- **TIO[i]_G[g]_CH[c]_SCMD** (t+1) = **TIO[i]_G[g]_CH[c]_OCMD** (t)

- **TIO[i]_G[g]_CH[c]_SOP** (t+1) = **TIO[i]_G[g]_CH[c]_OOP** (t)
- This instruction termination prevents that the **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** value and **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** value can be captured to the bit fields **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** , **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** .

This configuration uses the cyclic instruction buffer of the channel resource (details see chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"*), a next instruction is never pulled via the signal *INSTR_PULL* from the S resource. Signal *INSTR_PULL* =0 never changes.

27.5.10.6.10 Capture Buffer Usage On Instruction Termination

Configuration in use: **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 1; **TIO[i]_G[g]_CH[c+1]_CTRL.PL_S_MODE** = 0b01

To enable that the data from the resource of the channel [c] can be captured **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** =1 has to be configured and the S resource of the next channel [c+1] has to be configured as a continue buffer.

If the instruction terminates with *INSTR_END* = 1, a data push request is issued.

The captured values are directly stored to the capture buffer [c+1].

- **TIO[i]_G[g]_CH[c+1]_SCMD** [7:2]= **TIO[i]_G[g]_CH[c]_OCMD** [7:2]
- **TIO[i]_G[g]_CH[c+1]_SCMD** [1:1]= *S_OUT* [c:c]
- **TIO[i]_G[g]_CH[c+1]_SCMD** [0:0]= *O_OUT* [c:c]
- **TIO[i]_G[g]_CH[c+1]_SOP** = *TB*

The captured values are never stored to **TIO[i]_G[g]_CH[c]_OINST.CMD** and **TIO[i]_G[g]_CH[c]_OINST.OP** if capture buffer usage is enabled.

If **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1:

- **TIO[i]_G[g]_CH[c]_OCMD** and **TIO[i]_G[g]_CH[c]_OOP** keep unchanged.

If **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1:

- **TIO[i]_G[g]_CH[c]_OCMD** = **TIO[i]_G[g]_CH[c]_SCMD**
- **TIO[i]_G[g]_CH[c]_OOP** = **TIO[i]_G[g]_CH[c]_SOP**

If an instruction reload is enabled by **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** =1:

- **TIO[i]_G[g]_CH[c]_OCMD** = **TIO[i]_G[g]_CH[c]_SCMD**
- **TIO[i]_G[g]_CH[c]_OOP** = **TIO[i]_G[g]_CH[c]_SOP**

Note:
In case of **TIO[i]_G[g]_CH[c]_OINST.DATA_PUSH_EN** = 1; **TIO[i]_G[g]_CH[c+1]_CTRL.PL_S_MODE** = 0b10 | 0b11, the behavior of the S resource is undefined. The S resource will neither perform the counter instruction correctly nor the capture buffer functionality is operated correctly.

27.5.10.7 O Resource Compare Instructions

This instruction allows to set up an event, which will trigger when a counter / time base reaches a reference value. Based on this compare event defined changes of an output signal can be programmed.

There are two types of compare instructions: a single compare instruction and a dual compare instruction.

Following inputs are taken into account:

- *O_TRIG_OUT* [c:c] : trigger event allows to setup a defined *O_OUT* [c:c] value; see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- *PL_UPDATE* [c:c] : resolution for operation
- *S_OUT* [c:c] : value of **TIO[i]_S.CH[x]** with x = 8*g+c
- *O_OUT* [c:c] : value of **TIO[i]_O.CH[x]** with x = 8*g+c

This mode is selected with **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b10– | 0b110.

In the O resource compare mode the following bit fields in the command register exist:

- **TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION** : action on *O_OUT* [c:c] which has to be taken when compare occurs
- **TIO[i]_G[g]_CH[c]_OCMD_COMP.SEL_TB** : defines source which shall be captured
- **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** : command active (compare active / compare stopped)
- **TIO[i]_G[g]_CH[c]_OCMD_COMP.INSTR_PULL_EN** : enable instruction pull from instruction buffer
- **TIO[i]_G[g]_CH[c]_OCMD_COMP.DATA_PUSH_EN** : enable data push to capture buffer (single compare mode) and used to select in dual compare the sub-modes (serve first, etc.)

They can be accessed via the multiview register names:

- **TIO[i]_G[g]_CH[c]_OCMD_COMP**
- **TIO[i]_G[g]_CH[c]_OINST_COMP**

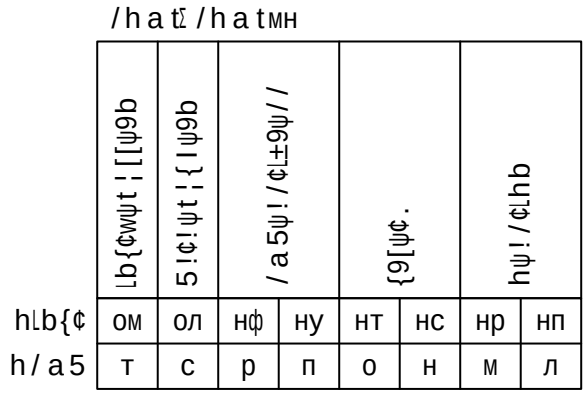
The bit field **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** in the command register controls the compare instruction:

- 0b00 : compare stopped
- 0b01 : dual compare active (see sections *27.5.10.7.8 "Dual Compare Instruction Serve First"* – *27.5.10.7.12 "Dual Compare; Channel Output O_OUT[c:c] Controlled by S_OUT[c:c] Value"*)
- 0b10 : single compare active (see sections *27.5.10.7.4 "Single Compare Instruction"* – *27.5.10.7.6 "Single Compare; Channel Output O_OUT[c:c] Controlled by S_OUT[c:c] Value"*)
- 0b11 : single compare active (see sections *27.5.10.7.4 "Single Compare Instruction"* – *27.5.10.7.6 "Single Compare; Channel Output O_OUT[c:c] Controlled by S_OUT[c:c] Value"*)

If a compare event occurs the captured values of the compare instruction can be accessed by the bit fields in **TIO[i]_G[g]_CH[c]_OCAPTURE** :

- **TIO[i]_G[g]_CH[c]_OCAPTURE.OP** : captured value of the selected *TB*
- **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** : captured value of signal *S_OUT* [c:c]
- **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** : captured value of signal *O_OUT* [c:c]
- **TIO[i]_G[g]_CH[c]_OCAPTURE.SEL_TB** : selected TB at capture
- **TIO[i]_G[g]_CH[c]_OCAPTURE.CMD_ACTIVE_CC** : enable/disable of instruction; after instruction termination **TIO[i]_G[g]_CH[c]_OCAPTURE.CMD_ACTIVE_CC** =0b00
- **TIO[i]_G[g]_CH[c]_OCAPTURE.INSTR_PULL_EN** : enable instruction pull from instruction buffer
- **TIO[i]_G[g]_CH[c]_OCAPTURE.DATA_PUSH_EN** : enable data push to capture buffer

Figure 194 CMD Register Layout – Compare Instruction



The execution of compare instructions can be influenced with the following configuration bits, the corresponding functions will be explained in the referred sections:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0 : see chapter *27.5.10.7.5 "Single Compare; Channel Output O_OUT[c:c] Controlled By Last Value"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 1 : see chapter *27.5.10.7.6 "Single Compare; Channel Output O_OUT[c:c] Controlled by S_OUT[c:c] Value"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1 : see chapter *27.5.10.7.16 "Instruction Termination With Instruction Freeze"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 : see chapter *27.5.10.7.18 "Instruction Termination With Cyclic Buffer Usage"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN** = 1 : see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_O_TRIG_OUT_EN** = 1 : see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*

- **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 1 instruction buffer usage : see chapter *27.5.10.7.17 "Instruction Termination With Instruction Reload"*
- **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 1 capture buffer usage : see chapter *27.5.10.7.19 "Capture Buffer Usage on Instruction Termination"*

For the functional description in the next chapters (*27.5.10.7.1 "Select Source Of Compare Instruction"* – *27.5.10.7.4 "Single Compare Instruction"*), the following is assumed:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1
- **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 0
- **TIO[i]_G[g]_CH[c]_CTRL.PL_O_TRIG_OUT_EN** = 1
- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0
- **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 0
- **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 0

27.5.10.7.1 Select Source Of Compare Instruction

The data which shall be compared is configured with the bit field **TIO[i]_G[g]_CH[c]_OCMD_COMP.SEL_TB** in the command register. (see table *73 "TIO[i]_G[g]_CH[c]_OCMD_COMP.SEL_TB Definition"*)

Table 73 TIO[i]_G[g]_CH[c]_OCMD_COMP.SEL_TB Definition

Channel c TIO[i]_G[g]_CH[c]_OCMD_COMP.SEL_TB	x = g*8 +c; g=floor(x/8) Source *TB* of channel c
0b00	*RS_TB0[g]*
0b01	*RS_TB1[g]*
0b10	*RS_TB2[g]*
0b11	**TIO[i]_G[g]_CH[c]_SINST.OP**

27.5.10.7.2 Select Equal Compare Instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b110

With this O resource mode the "equal" compare is selected. A compare event trigger signal *CMP_EVT* [c:c] will be generated if the compare instruction is active and the compare condition is true.

At any point in time a compare instruction can be deactivated by clearing **TIO[i]_G[g]_CH[c]_OINST_COMP.CMD_ACTIVE_CC** =0b00.

Any compare event trigger *CMP_EVT* [c:c]=1 occurring while **TIO[i]_G[g]_CH[c]_OINST_COMP.CMD_ACTIVE_CC** =0b00 will be ignored.

If the compare is activated with **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b01 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b10 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b11:

- If *TB* != **TIO[i]_G[g]_CH[c]_OINST_COMP.OP** is fulfilled the compare event will never occur, *CMP_EVT* [c:c] = 0 will be set.
- If *TB* = **TIO[i]_G[g]_CH[c]_OINST_COMP.OP** is fulfilled the compare event will occur on the selected update resolution *PL_UPDATE* [c:c]. The signal *CMP_EVT* [c:c] = *PL_UPDATE* [c:c] will be set.

27.5.10.7.3 Select "greater than or equal" Compare Instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b100 | 0b101

With this O resource mode the "greater than or equal" compare is selected.

Attention:

The implemented greater than or equal to compare supports the concept of cyclic event compare (with the time base assumed to count in the forward direction) as defined in *3.11.1 "Cyclic Event Compare"* . The timebases which are generated by the TBU can be in use with **TIO[i]_G[g]_CH[c]_OCMD_COMP.SEL_TB** != 0b11. This leads to the limitation that in case **TIO[i]_G[g]_CH[c]_OCMD_COMP.SEL_TB** == 0b11 is used and the cyclic compare shall not be applied, the range for both the TIO local counter **TIO[i]_G[g]_CH[c]_SINST.OP** and **TIO[i]_G[g]_CH[c]_OINST_COMP.OP** are limited to {0, ..., $2^{22}-1$}. By limiting the operands' ranges, the 'greater than or equal' compare instruction will degenerate into a conventional greater than or equal to compare. Same is true, if in the resource selection block *27.5.2 "TIO_PL Resource Selection"* one of the possible local values **TIO[i]_G[g]_CH[1]_SOP.OP** or **TIO[i]_G[g]_CH[5]_SOP.OP** is selected as *RS_TB1[g]* or *RS_TB2[g]* .

At any point of time a compare instruction can be deactivated by clearing **TIO[i]_G[g]_CH[c]_OINST_COMP.CMD_ACTIVE_CC** =0b00.

Any compare event trigger *CMP_EVT* [c:c]=1 occurring while **TIO[i]_G[g]_CH[c]_OINST_COMP.CMD_ACTIVE_CC** =0b00 will be ignored.

If the compare is activated with **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b01 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b10 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b11:

- If *TB* < **TIO[i]_G[g]_CH[c]_OINST_COMP.OP** is fulfilled the compare event will never occur, *CMP_EVT* [c:c] = 0 will be set.
- If *TB* >= **TIO[i]_G[g]_CH[c]_OINST_COMP.OP** is fulfilled the compare event will occur on the selected update resolution *PL_UPDATE* [c:c]. The signal *CMP_EVT* [c:c] = *PL_UPDATE* [c:c] will be set.

27.5.10.7.4 Single Compare Instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_EN** = 0; **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b10 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b11

A single compare instruction gets activated as soon as bit field **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b10 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b11 is set.

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 and (*PL_TRIG_OUT* [c:c] = 1 or *CYCLIC_INIT_TRIG* [c:c] = 1 or *O_INSTR_PULL_NEXT* = 1) is fulfilled, an asynchronous event for the cyclic buffer is active, the actual compare instruction is canceled (no *INSTR_END* = 1 is set). The command register reload actions explained in chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"* are executed.

In all other cases an active compare instruction waits until the compare event *CMP_EVT* [c:c] = 1 occurs.

The instruction terminates, *INSTR_END* = *CMP_EVT* [c:c] is set. **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b00 will be set.

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 and *INSTR_END* =1 is fulfilled the command register reload actions explained in chapter *27.5.10.7.18 "Instruction Termination With Cyclic Buffer Usage"* are executed.

At any point in time a compare instruction can be deactivated by setting **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b00.

A single compare instruction with **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1 will never alter the value of the channel output *O_OUT* [c:c], *PL_O_LOAD* [c:c] = 0 is set.

27.5.10.7.5 Single Compare; Channel Output O_OUT[c:c] Controlled By Last Value

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_EN** = 0; **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b10 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b11; **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 0

On **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0 the function to control the channel output *O_OUT* [c:c] on occurrence of a compare event is enabled.
The functionality explained in chapters (*27.5.10.7.1 "Select Source Of Compare Instruction"* – *27.5.10.7.4 "Single Compare Instruction"*) also applies to **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0.

If **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** is configured as 0 in a single compare instruction, it is mandatory to set **TIO[i]_INPUT_MODE** [c:c]=1, **TIO[i]_CYCLIC_MODE** [c:c]=1 and **TIO[i]_INVERT** [c:c]=0 of the corresponding channel c. Otherwise, the behavior of output signal *O_OUT* [c:c] is undefined.

With this instruction 2 signals can influence the channel output *O_OUT* [c:c].

- *O_TRIG_OUT* [c:c]: load trigger event, can be used to setup a defined value of *O_OUT* [c:c] for the active compare instruction
- *CMP_EVT* [c:c]: compare trigger, can be used to setup a defined value of *O_OUT* [c:c] at the end of the compare instruction

Note:

There is a delay of 1 system clock cycle until the output signal *O_OUT* [c:c] gets its new value triggered by a compare event or load trigger event.

In case of occurrence of both signals *O_TRIG_OUT* [c:c] and *CMP_EVT* [c:c] in the same clock cycle the action defined for the *O_TRIG_OUT* [c:c] signal will be applied on the channel output.

If the cyclic buffer is active (**TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1) and in case of occurrence of both signals *O_TRIG_OUT* [c:c] and (*PL_TRIG_OUT* [c:c] or *O_INSTR_PULL_NEXT*) in the same clock cycle the action defined for the *O_TRIG_OUT* [c:c] signal will be applied on the channel output.

If the compare instruction is deactivated by setting **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b00, the channel output *O_OUT* [c:c] will never change on any events occurring at *O_TRIG_OUT* [c:c].

If the channel output control is disabled by **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1, the channel output *O_OUT* [c:c] will never change on any events occurring at *O_TRIG_OUT* [c:c] or *CMP_EVT* [c:c].

In case **TIO[i]_G[g]_CH[c]_CTRL.PL_O_TRIG_OUT_EN** = 0, the channel output *O_OUT* [c:c] will never change on any events occurring at *O_TRIG_OUT* [c:c].

With the bit fields **TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION** , **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** in the compare instruction register, signal changes in register **TIO[i]_O.CH[x]** of the TIO channel can be initiated. In case of *PL_O_LOAD* [c:c] = 1, **TIO[i]_O.CH[x]** = *PL_O_VAL* [c:c] will be loaded. Following actions can be configured (see table *74 "Compare Instruction O_OUT Control; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 0"*):

Table 74 Compare Instruction O_OUT Control; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 0

TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC	TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION	*CMP_EVT* [c:c]	*O_TRIG_OUT* [c:c]	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b00	0b--	-	-	0	-	Compare deactivated: No change on output *O_OUT* [c:c]
0b1-	0b--	0	0	0	-	No change on output *O_OUT* [c:c]
0b1-	0b00	1	0	1	0	Compare event sets output *O_OUT* [c:c]=0
0b1-	0b01	1	0	1	1	Compare event sets output *O_OUT* [c:c]=1
0b1-	0b10	1	0	1	*O_OUT* [c:c]	Compare event holds output *O_OUT* [c:c]
0b1-	0b11	1	0	1	not(*O_OUT* [c:c])	Compare event inverts output *O_OUT* [c:c]
0b10	0b00	-	1	1	not(*O_OUT* [c:c])	Oload trigger event inverts output *O_OUT* [c:c]
0b10	0b01	-	1	1	not(*O_OUT* [c:c])	Oload trigger event inverts output *O_OUT* [c:c]
0b10	0b10	-	1	1	0	Oload trigger event sets output *O_OUT* [c:c]=0
0b10	0b11	-	1	1	0	Oload trigger event sets output *O_OUT* [c:c]=0
0b11	0b00	-	1	1	*O_OUT* [c:c]	Oload trigger event holds output *O_OUT* [c:c]
0b11	0b01	-	1	1	*O_OUT* [c:c]	Oload trigger event holds output *O_OUT* [c:c]
0b11	0b10	-	1	1	1	Oload trigger event sets output *O_OUT* [c:c]=1
0b11	0b11	-	1	1	1	Oload trigger event sets output *O_OUT* [c:c]=1

27.5.10.7.6 Single Compare; Channel Output O_OUT[c:c] Controlled by S_OUT[c:c] Value

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_EN** = 0; **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b10 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b11; **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 1

If the configuration bit **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** =1 is set, the action on the channel output *O_OUT* [c:c] which is defined by the bit fields **TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION** , **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** can depend on the value of the signal *S_OUT* [c:c]. This allows to setup a compare instruction which will alter the value of the signal *O_OUT* [c:c] to the value of *S_OUT* [c:c] or not (*S_OUT* [c:c]).

The functionality explained in chapter *27.5.10.7.5 "Single Compare; Channel Output O_OUT[c:c] Controlled By Last Value"* also applies to **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 1, just the *O_OUT* [c:c] behavior on an init / compare trigger event will be defined differently (see table *75 "Compare Instruction O_OUT[c:c] Control; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 1"*).

If **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** is configured as 0 in a single compare instruction, it is mandatory to set **TIO[i]_INPUT_MODE** [c:c]=1, **TIO[i]_CYCLIC_MODE** [c:c]=1 and **TIO[i]_INVERT** [c:c]=0 of the corresponding channel c. Otherwise, the behavior of output signal *O_OUT* [c:c] is undefined.

Note:

There is a delay of 1 system clock cycle until the output signal *O_OUT* [c:c] gets its new value triggered by a compare event or load trigger event.

Table 75 Compare Instruction O_OUT[c:c] Control; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 1

TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC	TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION	*CMP_EVT* [c:c]	*O_TRIG_OUT* [c:c]	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b00	0b--	-	-	0	-	Compare deactivated: No change on output *O_OUT* [c:c]
0b1-	0b--	0	0	0	-	No change on output *O_OUT* [c:c]
0b1-	0b00	1	0	1	0	Compare event sets output *O_OUT* [c:c]=0
0b1-	0b01	1	0	1	1	Compare event sets output *O_OUT* [c:c]=1
0b1-	0b10	1	0	1	*S_OUT* [c:c]	Compare event sets output *O_OUT* [c:c] to the value of *S_OUT* [c:c]
0b1-	0b11	1	0	1	not(*S_OUT* [c:c])	Compare event sets output *O_OUT* [c:c] to the inverted value of *S_OUT* [c:c]
0b10	0b00	-	1	1	not(*S_OUT* [c:c])	Oload trigger event sets output *O_OUT* [c:c] to the inverted value of *S_OUT* [c:c]
0b10	0b01	-	1	1	not(*S_OUT* [c:c])	Oload trigger event sets output *O_OUT* [c:c] to the inverted value of *S_OUT* [c:c]
0b10	0b10	-	1	1	0	Oload trigger event sets output *O_OUT* [c:c]=0
0b10	0b11	-	1	1	0	Oload trigger event sets output *O_OUT* [c:c]=0
0b11	0b00	-	1	1	*S_OUT* [c:c]	Oload trigger event sets output *O_OUT* [c:c]
0b11	0b01	-	1	1	*S_OUT* [c:c]	Oload trigger event sets output *O_OUT* [c:c] to the value of *S_OUT* [c:c]
0b11	0b10	-	1	1	1	Oload trigger event sets output *O_OUT* [c:c]=1

TIO[i]_G[g]_CH[-c]_OCMD_COMP.-CMD_ACTIVE_CC	TIO[i]_G[g]_CH[-c]_OCMD_COMP.-O_ACTION	*CMP_EVT* [c:c]	*O_TRIG_O-UT* [c:c]	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b11	0b11	–	1	1	1	Oload trigger event sets output *O_OUT* [c:c]=1

27.5.10.7.7 Dual Compare Instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 0 or **TIO[i]_G[g]_-CH[c]_CTRL2.DUAL_CMP_MST_EN** = 1; **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b01

Definition: Dual compare instruction uses 2 parallel active compare resources. Depending on the configuration, a dual compare instruction can be terminated by one or both compare events. For details see the following sections *27.5.10.7.8 "Dual Compare Instruction Serve First"* , *27.5.10.7.9 "Dual Compare Instruction Serve Both"* and *27.5.10.7.10 "Dual Compare Instruction Serve Both Master First"* .

A dual compare instruction allows to setup 2 compare instructions in neighbored channels [c] and [c−1] with c = 1..7.

One compare is setup in the O resource of channel [c] which is the master channel (**TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b10− | 0b110 and **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 1 have to be set) and the second compare is setup in the O resource of channel [c−1] which is the slave channel (**TIO[i]_G[g]_CH[c−1]_CTRL.PL_O_MODE** = 0b10− or **TIO[i]_G[g]_CH[c−1]_CTRL.PL_O_MODE** = 0b110 and **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 0 have to be set).

A dual compare may not be setup with a master channel [c] with c = 0.

Resources in use with a dual compare instruction have to be in one channel group.

To operate a dual compare, the following communication signals exist between a master and a slave channel:
o master channel [c] to slave channel [c−1]:

- *DUALCMP_END* [c:c] : 1 = terminate dual compare instruction; 0 = no change

o Slave channel [c−1] to master channel [c]:

- *O_ACTION_PREV[c]* : configuration bit field **TIO[i]_G[g]_CH[c−1]_OCMD_COMP.O_ACTION**
- *CMD_ACTIVE_CC_PREV[c]* : configuration bit field **TIO[i]_G[g]_CH[c−1]_OCMD_COMP.CMD_ACTIVE_CC**
- *CMP_EVT* _PREV[c:c] : compare trigger event of O resource instruction [c−1]
- *PL_ODIS_PREV* [c:c] : configuration bit field **TIO[i]_G[g]_CH[c−1]_CTRL.PL_ODIS**

Both compare instructions have to be setup while the channels are disabled: **TIO[i]_CENDIS** = 0b11 << ((g*8+c)−1).

It is mandatory to enable the both channels synchronously with an atomic write: **TIO[i]_SENDIS** = 0b11 << ((g*8+c)−1).

If a dual compare shall be triggered by a *FUPD* [c:c]=1, it is mandatory that **TIO[i]_ENDIS** [x:x]=1 (x = g*8+c) is set.
Further, all limitations of *PL_UPDATE* [c:c] are also applied to *FUPD* [c:c].

After enabling the dual compare instruction in both channels [c] and [c−1] the active compare instruction waits until the configured compare event(s) occurred. For details please see the following sections.

If the configured compare event(s) occurred the master channel terminates the instruction and set signals *INSTR_END* = 1 and *DUALCMP_E-ND* [c:c] = 1.

Signal *DUALCMP_END* [c:c] is routed to the slave channel as *DUALCMP_END_NEXT* [c−1:c−1] = *DUALCMP_END* [c:c] and also triggers the slave channel to terminate the instruction.

If signal *DUALCMP_END* [c−1:c−1] = 1 then the slave channel set signal *INSTR_END* [c−1:c−1] = 1.

Both channels terminate their instruction synchronously.

The slave channel never set signal *DUALCMP_END* [c−1:c−1] = 1.

The bit fields **TIO[i]_G[g]_CH[c−1]_OCMD_COMP.CMD_ACTIVE_CC** and **TIO[i]_G[g]_CH[c−1]_OCMD_COMP.O_ACTION** are not in use at slave channel [c−1], they are evaluated in the master channel [c].

A dual compare slave will never change the channel output *O_OUT* [c−1:c−1] .

A dual compare slave will never set *PL_EVT* [c:c] = 1.

If the instruction terminates (*INSTR_END* =1) and no reload is active (**TIO[i]_G[g]_CH[c−1]_CTRL.PL_FREEZE_O_EN** =0 and **TIO[i]_G[g]_-CH[c−1]_CTRL.PL_CYCLIC_BUFF** =0 and **TIO[i]_G[g]_CH[c−1]_OINST_COMP.INSTR_PULL_EN** =0) then **TIO[i]_G[g]_CH[c−1]_OINST_COM-P.CMD_ACTIVE_CC** =0b00 is set.

The modes defined in chapters (*27.5.10.7.1 "Select Source Of Compare Instruction"* – *27.5.10.7.3 "Select "greater than or equal" Compare Instruction"* , *27.5.10.7.13 "Compare Instruction With Capture"*) will also apply for the dual compare slave operation.

The result of the compare trigger *CMP_EVT* [c-1:c-1] of the slave channel will be transferred to the master channel via the signal *CMP_EVT_PREV* [c:c].

Attention:

It is mandatory for a dual compare instruction that *PL_UPDATE* [c:c] <= 1/2 cluster clock frequency. Further *PL_UPDATE* [c:c] must be single pulses of cluster clock frequency.
Further all limitations of *PL_UPDATE* [c:c] are applied to *FUPD* [c:c] too.

When the master channel [c] is configured as a cyclic buffer and the slave is not, it is possible to cancel a dual compare instruction by one of the following trigger signals *CYCLIC_INIT_TRIG* [c:c]='1', *PL_TRIG_OUT* [c:c]='1' or *O_INSTR_PULL_NEXT* ='1' and **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1.

When the slave channel [c-1] is configured as a cyclic buffer and the master is not, it is not allowed to use the cancel feature.

Attention:

When both channels are configured as cyclic buffers, it is possible to cancel a dual compare instruction by one of the following trigger signals *CYCLIC_INIT_TRIG* [c:c]='1', *PL_TRIG_OUT* [c:c]='1' and **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1.

- If this feature is used it is mandatory that both dual compare channels [c] and [c-1] are running with the same resolution *PL_UPDATE* [c:c] and that the cancel trigger signals triggers both channels at the same time. Otherwise, the future behavior is undefined.
- It is not allowed to use *O_INSTR_PULL_NEXT* to cancel the instruction.
- If a primary input signal is used as source of *CYCLIC_INIT_TRIG* [c:c] and/or *PL_TRIG_OUT* [c:c] the input selection block can be used to route the primary input signal to both channels of the dual compare application. Then the trigger signals for the slave channel [c-1] and the master channel [c] are synchronous.
- If the feature is used, it is not allowed to use *TRIG_OUT_PREV* [c:c], *PL_TRIG_OUT_PREV* [c:c] or *PL_EVT* [c:c] as source for *PL_TRIG_OUT* [c:c].
- If the feature is used, it is not allowed to use *TRIG_OUT_PREV* [c:c], *PL_TRIG_OUT_PREV* [c:c] or *PL_EVT* [c:c] as source for trigger output to be used as *CYCLIC_INIT_TRIG* [c:c].
- If the cancel feature is not used, the two dual compare channels may run at different *PL_UPDATE* [c:c] resolutions.

Note:

Be aware that the bit field **TIO[i]_G[g]_CH[c]_OCMD_COMP.DATA_PUSH_EN** in the instruction register will be used to decode the serve both compare mode. A dual compare instruction can never be used together with a capture buffer.
The usage of **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b01 and **TIO[i]_G[g]_CH[c]_OCMD_COMP.DATA_PUSH_EN** = 1 will force *DATA_PUSH* = 0.

27.5.10.7.8 Dual Compare Instruction Serve First

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 0 or **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 1; **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b01; **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = **TIO[i]_G[g]_CH[c-1]_CTRL.PL_ODIS**

Definition: Dual compare instruction serve first; uses 2 parallel active compare resources, the first occurring compare event will terminate the dual compare instruction.

The dual compare instruction "serve first" operates as described in section *27.5.10.7.7 "Dual Compare Instruction"* with the behavior details documented subsequently:

The output *O_OUT* [c:c] control mode is selected by **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** and *PL_ODIS_PREV* [c:c] (*PL_ODIS_PREV* [c:c] = **TIO[i]_G[g]_CH[c-1]_CTRL.PL_ODIS**):

- Compare triggers have no impact on *O_OUT* [c:c]: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1; *PL_ODIS_PREV* [c:c] = 1;
  On initialization, the signals *CMP_EVT_EN* [c:c] and *CMP_EVT_PREV_EN* [c:c] are set to 0. Compare events which have occurred during a previous activation will be cleared.
- First compare trigger controls *O_OUT* [c:c]: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0; *PL_ODIS_PREV* [c:c] = 0;
  On initialization, the signals *CMP_EVT_EN* [c:c] and *CMP_EVT_PREV_EN* [c:c] are set to 1. Compare events which have occurred during a previous activation will be cleared.

If the conditions **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 and (*PL_TRIG_OUT* [c:c] = 1 or *CYCLIC_INIT_TRIG* [c:c] = 1 or *O_INSTR_PULL_NEXT* = 1) are fulfilled, an asynchronous event for the cyclic buffer is active, the actual compare instruction is canceled (no *INSTR_END* = 1 is set). The command register reload actions explained in chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"* are executed. Same is applied in the slave channel [c-1].

In all other cases an active compare instruction waits until the termination of the serve first instruction occurs, which is defined as:
As soon as the master channel or the slave channel compare trigger occurs (*INSTR_END* = (*CMP_EVT* [c:c] or *CMP_EVT_PREV* [c:c])).

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 and *INSTR_END* = 1 is fulfilled the command register reload actions explained in chapter *27.5.10.7.18 "Instruction Termination With Cyclic Buffer Usage"* are executed. Same is applied in the slave channel [c-1].

27.5.10.7.9 Dual Compare Instruction Serve Both

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 0 or **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 1; **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b01; **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 1

Definition: Dual compare instruction serve both; uses 2 parallel active compare resources, after both compare trigger events have occurred the dual compare serve both instructions will terminate. Any ordering of the events is possible, even when both events occur at the same system clock cycle, the instruction will terminate.

The dual compare instruction "serve both" operates as described in section *27.5.10.7.7 "Dual Compare Instruction"* with the behavior details documented subsequently:

The serve both output *O_OUT* [c:c] control mode is selected by **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** and *PL_ODIS_PREV* [c:c].
o Compare triggers have no impact on *O_OUT* [c:c]: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1; *PL_ODIS_PREV* [c:c] = 1

- On initialization, the signals *CMP_EVT_EN* [c:c] and *CMP_EVT_PREV_EN* [c:c] are set to 0. Compare events which have occurred during a previous activation will be cleared.

o Both compare triggers control *O_OUT* [c:c]: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0; *PL_ODIS_PREV* [c:c] = 0

- On initialization, the signals *CMP_EVT_EN* [c:c] and *CMP_EVT_PREV_EN* [c:c] are set to 1. Compare events which have occurred during a previous activation will be cleared.
- In case *CMP_EVT* [c:c] = 1, the signal *CMP_EVT_EN* [c:c] = 0 is set.
- In case *CMP_EVT_PREV* [c:c] = 1, the signal *CMP_EVT_PREV_EN* [c:c]= 0 is set.

o First compare trigger controls *O_OUT* [c:c]: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0; *PL_ODIS_PREV* [c:c] = 1

- On initialization, the compare events which have occurred during a previous activation will be cleared and the signals *CMP_EVT_EN* [c:c] = *CMP_EVT_PREV_EN* [c:c] = 1 are in use until the first compare trigger event occurs:
- In case *CMP_EVT* [c:c]= 1 or *CMP_EVT_PREV* [c:c] = 1, the signals *CMP_EVT_PREV_EN* [c:c] = 0 and *CMP_EVT_PREV_EN* [c:c] = 0 are set.

o Second compare trigger controls *O_OUT* [c:c]: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1; *PL_ODIS_PREV* [c:c] = 0

- On initialization, the compare events which have occurred during a previous activation will be cleared and the signals *CMP_EVT_EN* [c:c] = *CMP_EVT_PREV_EN* [c:c] = 0 are in use until the first compare trigger event occurs:
- In case *CMP_EVT* [c:c] =1, the signal *CMP_EVT_PREV_EN* [c:c]= 1 is set.
- In case *CMP_EVT_PREV* [c:c] = 1, the signal *CMP_EVT_EN* [c:c] = 1 is set.

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 and (*PL_TRIG_OUT* [c:c]= 1 or *CYCLIC_INIT_TRIG* [c:c] = 1 or *O_INSTR_PULL_NEXT* = 1) is fulfilled, an asynchronous event for the cyclic buffer is active, the actual compare instruction is canceled (no *INSTR_END* = 1 is set). The command register reload actions explained in chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"* are executed. Same is applied in the slave channel [c−1].

In all other cases an active compare instruction waits until the termination of the serve both instruction occurs, which is defined as:
As soon as the master channel and the slave channel compare trigger events have occurred.

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 and *INSTR_END* = 1 is fulfilled the command register reload actions explained in chapter *27.5.10.7.18 "Instruction Termination With Cyclic Buffer Usage"* are executed. Same is applied in the slave channel [c−1].

Note:

If a Dual Compare Serve Both instruction is used in combination with "Second compare trigger controls O_OUT" and both compare events occur at the same time then the instruction is finished and O_OUT is not changed.

27.5.10.7.10 Dual Compare Instruction Serve Both Master First

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 0 or **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_MST_EN** = 1; **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b01; **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** != **TIO[i]_G[g]_CH[c−1]_CTRL.PL_ODIS**

Definition: Dual compare instruction serve both master first; uses 2 compare resources, first the compare in the master channel is active, after occurrence of the compare event of the master channel the slave channel compare will be active and the instruction will terminate as soon as the slave channel compare event will occur.

The dual compare instruction "serve both master first" operates as described in section *27.5.10.7.7 "Dual Compare Instruction"* with the behavior details documented subsequently:

The serve both master first output *O_OUT* [c:c] control mode is selected by **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** and *PL_ODIS_PREV* [c:c].
o Both compare triggers control *O_OUT* [c:c]: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 0; *PL_ODIS_PREV* [c:c] = 1

- On initialization, the compare events which have occurred during a previous activation will be cleared and the signal *CMP_EVT_EN* [c:c] = 1; *CMP_EVT_PREV_EN* [c:c] = 0 is in use.
- If a compare event occurs at the master channel [c] the signals *CMP_EVT_EN* [c:c] = 0 and *CMP_EVT_PREV_EN* [c:c] = 1 are set.

o Second compare trigger controls *O_OUT* [c:c]: **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** = 1; *PL_ODIS_PREV* [c:c] = 0

- On initialization, the compare events which have occurred during a previous activation will be cleared and the signal *CMP_EVT_EN* [c:c] = 0; *CMP_EVT_PREV_EN* [c:c] = 0 is in use.
- If a compare event occurs at the master channel [c] the signals *CMP_EVT_EN* [c:c]= 0 and *CMP_EVT_PREV_EN* [c:c]= 1 are set.

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 and (*PL_TRIG_OUT* [c:c] = 1 or *CYCLIC_INIT_TRIG* [c:c] = 1 or *O_INSTR_PULL_NEXT* = 1) is fulfilled, an asynchronous event for the cyclic buffer is active, the actual compare instruction is canceled (no *INSTR_END* =1 is set). The command register reload actions explained in chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"* are executed. Same is applied in the slave channel [c−1].

In all other cases, an active compare instruction waits until the termination of the serve both master first instruction occurs, which is defined as:
The master channel will terminate first as soon as the master channel compare event occurs followed by a slave channel compare trigger.

If the condition **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 and *INSTR_END* = 1 is fulfilled, the command register reload actions explained in chapter *27.5.10.7.18 "Instruction Termination With Cyclic Buffer Usage"* are executed. Same is applied in the slave channel [c−1].

27.5.10.7.11 Dual Compare; Channel Output O_OUT[c:c] Controlled by Last Value

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 0

With the dual compare strategy changes on *O_OUT* [c:c] can only be initiated by the compare match events *CMP_EVT* [c:c] and *CMP_EVT_PREV* [c:c], events on *O_TRIG_OUT* [c:c] have no impact.

If **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** is configured as 0 or *PL_ODIS_PREV* [c:c] is configured as 0 in a dual compare instruction, it is mandatory to set **TIO[i]_INPUT_MODE** [c:c]=1, **TIO[i]_CYCLIC_MODE** [c:c]=1 and **TIO[i]_INVERT** [c:c]=0 of the corresponding channel c. Otherwise, the behavior of output signal *O_OUT* [c:c] is undefined.

Each compare match event is masked by the associated *CMP_EVT_EN* [c:c], *CMP_EVT_PREV_EN* [c:c] signal.

Dependent on which channel ([c], [c−1]) generates the match *CMP_EVT* [c:c] or *CMP_EVT_PREV* [c:c] = *CMP_EVT* [c−1:c−1] the corresponding match behavior **TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION** or **TIO[i]_G[g]_CH[c−1]_OCMD_COMP.O_ACTION** (alias signal *O_ACTION_PREV[c]*) gets applied on the output *O_OUT* [c:c].

In case of a simultaneous match ((*CMP_EVT_EN* [c:c] AND *CMP_EVT* [c:c]) = (*CMP_EVT_PREV_EN* [c:c] AND *CMP_EVT_PREV* [c:c]) = 1 the match behavior is decoded in a special way using *CMD_ACTIVE_CC_PREV[c]* = **TIO[i]_G[g]_CH[c−1]_OCMD_COMP.CMD_ACTIVE_CC** . Details can be seen in table *76 "Dual Compare Instruction Control; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 0"* .

Note:

There is a delay of 1 system clock cycle until the output signal *O_OUT* gets its new value triggered by a compare event.

Table 76 Dual Compare Instruction Control; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 0

O_ACTION[c]	*CMP_EVT_EN* [c:c] AND *CMP_EVT* [c:c]	*O_ACTION_PREV[c]*	*CMP_EVT_PREV_EN* [c:c] AND *CMP_EVT_PREV* [c:c]	*CMD_ACTIVE_CC_PREV[c]*	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b−−	0	0b−−	0	0b−−	0	–	No change on output *O_OUT* [c:c]
0b00	1	0b−−	0	0b−−	1	0	Master compare event sets output *O_OUT* [c:c]=0
0b01	1	0b−−	0	0b−−	1	1	Master compare event sets output *O_OUT* [c:c]=1

O_ACTION[c]	*CMP_EVT_EN* [c:c] AND *CMP_EVT* [c:c]	*O_ACTION_PREV[c]*	*CMP_EVT_PREV_EN* [c:c] AND *CMP_EVT_PREV* [c:c]	*CMD_ACTIVE_CC_PREV[c]*	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b10	1	0b--	0	0b--	1	*O_OUT* [c:c]	Master compare event holds output *O_OUT* [c:c]
0b11	1	0b--	0	0b--	1	not(*O_OUT* [c:c])	Master compare event inverts output *O_OUT* [c:c]
0b--	0	0b00	1	0b--	1	0	Slave compare event sets output *O_OUT* [c:c]=0
0b--	0	0b01	1	0b--	1	1	Slave compare event sets output *O_OUT* [c:c]=1
0b--	0	0b10	1	0b--	1	*O_OUT* [c:c]	Slave compare event holds output *O_OUT* [c:c]
0b--	0	0b11	1	0b--	1	not(*O_OUT* [c:c])	Slave compare event inverts output *O_OUT* [c:c]
0b--	1	0b--	1	0b00	1	0	Simultaneous compare event sets output *O_OUT* [c:c]=0
0b--	1	0b--	1	0b01	1	1	Simultaneous compare event sets output *O_OUT* [c:c]=1
0b--	1	0b--	1	0b10	1	*O_OUT* [c:c]	Simultaneous compare event holds output *O_OUT* [c:c]
0b--	1	0b--	1	0b11	1	not(*O_OUT* [c:c])	Simultaneous compare event inverts output *O_OUT* [c:c]

27.5.10.7.12 Dual Compare; Channel Output O_OUT[c:c] Controlled by S_OUT[c:c] Value

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 1

If the configuration bit **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN** = 1 is set, the action on the channel output *O_OUT* [c:c] which is defined by the bit fields **TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION** and **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** can depend on the value of the signal *S_OUT* [c:c]. This allows to setup a compare instruction which will alter the value of the signal *O_OUT* [c:c] to the value of *S_OUT* [c:c] or not (*S_OUT* [c:c]).

If **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS** is configured as 0 or *PL_ODIS_PREV* [c:c] is configured as 0 in a dual compare instruction, it is mandatory to set **TIO[i]_INPUT_MODE** [c:c]=1, **TIO[i]_CYCLIC_MODE** [c:c]=1 and **TIO[i]_INVERT** [c:c]=0 of the corresponding channel c. Otherwise, the behavior of output signal *O_OUT* [c:c] is undefined.

With the dual compare strategy changes on *O_OUT* [c:c] can only be initiated by the compare match events *CMP_EVT* [c:c] and *CMP_EVT_PREV* [c:c], events on *O_TRIG_OUT* [c:c] have no impact.

Each compare match event is masked by the associated *CMP_EVT_EN* [c:c], *CMP_EVT_PREV_EN* [c:c] signal.

Dependent on which channel ([c], [c−1]) generates the match *CMP_EVT* [c:c] or *CMP_EVT_PREV* [c:c] = *CMP_EVT* [c−1:c−1] the corresponding match behavior **TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION** or **TIO[i]_G[g]_CH[c−1]_OCMD_COMP.O_ACTION** (alias signal *O_ACTION_PREV[c]*) gets applied on the output *O_OUT* [c:c].

In case of a simultaneous match ((*CMP_EVT_EN* [c:c] AND *CMP_EVT* [c:c]) = (*CMP_EVT_PREV_EN* [c:c] AND *CMP_EVT_PREV* [c:c]) = 1 the match behavior is decoded in a special way using *CMD_ACTIVE_CC_PREV[c]* = **TIO[i]_G[g]_CH[c-1]_OCMD_COMP.CMD_ACTIVE_CC** . Details can be seen in table *77 "Dual Compare Instruction O_OUT[c:c] Control; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 1"* .

Note:

There is a delay of 1 system clock cycle until the output signal *O_OUT* gets its new value triggered by a compare event.

Table 77 Dual Compare Instruction O_OUT[c:c] Control; TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN = 1

O_ACTION[c]	*CMP_EVT_EN* [c:c] AND *CMP_EVT* [c:c]	*O_ACTION_PREV[c]*	*CMP_EVT_PREV_EN* [c:c] AND *CMP_EVT_PREV* [c:c]	*CMD_ACTIVE_CC_PREV[c]*	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b--	0	0b--	0	0b--	0	-	No change on output *O_OUT* [c:c]
0b00	1	0b--	0	0b--	1	0	Master compare event sets output *O_OUT* [c:c]=0
0b01	1	0b--	0	0b--	1	1	Master compare event sets output *O_OUT* [c:c]=1
0b10	1	0b--	0	0b--	1	*S_OUT* [c:c]	Master compare event sets output *O_OUT* [c:c] to the value of *S_OUT* [c:c]
0b11	1	0b--	0	0b--	1	not(*S_OUT* [c:c])	Master compare event sets output *O_OUT* [c:c] to the inverted value of *S_OUT* [c:c]
0b--	0	0b00	1	0b--	1	0	Slave compare event sets output *O_OUT* [c:c]=0
0b--	0	0b01	1	0b--	1	1	Slave compare event sets output *O_OUT* [c:c]=1
0b--	0	0b10	1	0b--	1	*S_OUT* [c:c]	Slave compare event sets output *O_OUT* [c:c] to the value of *S_OUT* [c:c]
0b--	0	0b11	1	0b--	1	not(*S_OUT* [c:c])	Slave compare event sets output *O_OUT* [c:c] to the inverted value of *S_OUT* [c:c]
0b--	1	0b--	1	0b00	1	0	Simultaneous compare event sets output *O_OUT* [c:c]=0
0b--	1	0b--	1	0b01	1	1	Simultaneous compare event sets output *O_OUT* [c:c]=1

O_ACTION[c]	*CMP_EVT_EN* [c:c] AND *CMP_EVT* [c:c]	*O_ACTION_PREV[c]*	*CMP_EVT_PREV_EN* [c:c] AND *CMP_EVT_PREV* [c:c]	*CMD_ACTIVE_CC_PREV[c]*	Output *PL_O_LOAD* [c:c]	Output *PL_O_VAL* [c:c]	Description
0b--	1	0b--	1	0b10	1	*S_OUT* [c:c]	Simultaneous compare event sets output *O_OUT* [c:c] to the value of *S_OUT* [c:c]
0b--	1	0b--	1	0b11	1	not(*S_OUT* [c:c])	Simultaneous compare event sets output *O_OUT* [c:c] to the value of *S_OUT* [c:c]

27.5.10.7.13 Compare Instruction With Capture

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b101; (**TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0 and **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0)

In addition to the functionality defined in the previous chapters, it is possible to capture data at the end of the compare instruction.

Note:
The usage of the compare instruction with capture is only possible if no reload option is used (no instruction freeze, no cyclic buffer usage, no instruction pull) or capture buffer usage is enabled.

This allows to capture the values of the *S_OUT* [c:c] / *O_OUT* [c:c] signals at the point of time an instruction end condition occurs. Furthermore, it can be beneficial if a "greater than or equal" compare terminates, to capture the *TB* value at the instruction end of the compare.

The capture gets initiated if *INSTR_END* = 1 occurs.

Following data is captured on the instruction end of a compare instruction:

- **TIO[i]_G[g]_CH[c]_OCAPTURE.OP** = *TB*
- **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** = *S_OUT* [c:c]
- **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** = *O_OUT* [c:c]

Captured data is accessible with register alias **TIO[i]_G[g]_CH[c]_OCAPTURE** .

27.5.10.7.14 PL_EVT Generation

For every compare instruction the PL_EVT condition is defined as: *PL_EVT* [c:c] = *INSTR_END*

The *PL_EVT* [c:c] is set on each instruction termination.

Note:
A dual compare slave will never set *PL_EVT* [c:c] = 1.

27.5.10.7.15 Instruction Termination Without Instruction Reload

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 0

If the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b00 (instruction stopped) is set.

A next instruction is not pulled, *INSTR_PULL* = 0.

A next instruction can only be issued by writing **TIO[i]_G[g]_CH[c]_OCMD_COMP** with a new instruction command.

27.5.10.7.16 Instruction Termination With Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 0|1

If the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_OCMD** will remain unchanged. On initialization, the internal register *CMP_EVT_EN* [c:c], *CMP_EVT_PREV_EN* [c:c] will be set according to the active dual compare instruction. In addition, compare events which have occurred during a previous activation will be cleared.

A next instruction is not pulled, *INSTR_PULL* = 0.

This results in an endless execution of the command in **TIO[i]_G[g]_CH[c]_OCMD** . This allows to apply the compare instruction in an endless manner.

Note:

Due to freezing, the **TIO[i]_G[g]_CH[c]_OINST** register will never change. This prevents that the *S_OUT* [c:c] / *O_OUT* [c:c] and *TB* values can be captured.

27.5.10.7.17 Instruction Termination With Instruction Reload

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_OCMD.INSTR_PULL_EN** = 1

If the instruction terminates with *INSTR_END* = 1, the next instruction will be loaded from instruction buffer (*INSTR_PULL* = *INSTR_END*):

- **TIO[i]_G[g]_CH[c]_OCMD** = **TIO[i]_G[g]_CH[c]_SCMD**
- **TIO[i]_G[g]_CH[c]_OOP** = **TIO[i]_G[g]_CH[c]_SOP**
- On initialization, the internal registers *CMP_EVT_EN* [c:c] and *CMP_EVT_PREV_EN* [c:c] will be set according to the active dual compare instruction. In addition, compare events which have occurred during a previous activation will be cleared.

The new loaded compare instruction is immediately executed and defines the value for the pending **TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION** . This prevents that the *S_OUT* [c:c] / *O_OUT* [c:c] values can be captured to the same bits already in use by the bit field **TIO[i]_G[g]_CH[c]_OCMD_COMP.O_ACTION** .

27.5.10.7.18 Instruction Termination With Cyclic Buffer Usage

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1

A next instruction will be loaded from the cyclic instruction buffer, the following actions are executed:

- **TIO[i]_G[g]_CH[c]_OINST** (t+1) = **TIO[i]_G[g]_CH[c]_SINST** (t)
- **TIO[i]_G[g]_CH[c]_SINST** (t+1) = **TIO[i]_G[g]_CH[c]_OINST** (t)
- On initialization, the internal registers *CMP_EVT_EN* [c:c] and *CMP_EVT_PREV_EN* [c:c] will be set according to the active dual compare instruction. In addition, compare events which have occurred during a previous activation will be cleared.
- This instruction termination prevents that the *S_OUT* [c:c] / *O_OUT* [c:c] / *TB* values can be captured to the bit fields **TIO[i]_G[g]_CH[c]_OCAPTURE.O_OUT** , **TIO[i]_G[g]_CH[c]_OCAPTURE.S_OUT** , **TIO[i]_G[g]_CH[c]_OCAPTURE.OP** .

This configuration uses the cyclic instruction buffer of the channel resource (details see chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"*), a next instruction is never pulled via the signal *INSTR_PULL* from the S resource. Signal *INSTR_PULL* = 0 never changes.

27.5.10.7.19 Capture Buffer Usage on Instruction Termination

Configuration in use: **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 1; **TIO[i]_G[g]_CH[c+1]_CTRL.PL_S_MODE** = 0b01

To enable that data from the resource of the channel [c] can be captured **TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** = 1 has to be configured and the S resource of the next channel [c+1] has to be configured as a continue buffer.

If a dual compare is configured *DATA_PUSH* = 0 is forced. A capture buffer can't be configured for dual compare instructions.

In all other cases **TIO[i]_G[g]_CH[c]_OCMD_CAP.CMD_ACTIVE_CC** != 0b01 or **TIO[i]_G[g]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** != 0b01 or **TIO[i]_G[g]_CH[c]_CTRL2.DUAL_CMP_EN** = 0, if the instruction terminates with *INSTR_END* = 1, a data push request is issued (*DATA_PUSH* = *INSTR_END*).

o Compare without capture (**TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b100 or **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b110):

- The current OINST register is copied to the capture buffer. **TIO[i]_G[g]_CH[c+1]_SINST** = **TIO[i]_G[g]_CH[c]_OINST**

o Compare with capture (PL_O_MODE=0b101):

- The captured values are directly stored to the capture buffer [c+1].
- **TIO[i]_G[g]_CH[c+1]_SCMD** [7:2] = **TIO[i]_G[g]_CH[c]_OCMD** [7:2]

- **TIO[i]_G[g]_CH[c+1]_SCMD** [1:1] = *S_OUT* [c:c]
- **TIO[i]_G[g]_CH[c+1]_SCMD** [0:0] = *O_OUT* [c:c]
- **TIO[i]_G[g]_CH[c+1]_SOP** = *TB*

The captured values are never stored to **TIO[i]_G[g]_CH[c]_OINST.CMD** and **TIO[i]_G[g]_CH[c]_OINST.OP** if capture buffer usage is enabled.

Note:

In case of **TIO[i]_G[g]_CH[c]_OINST.DATA_PUSH_EN** = 1; **TIO[i]_G[g]_CH[c+1]_CTRL.PL_S_MODE** = 0b10 | 0b11, the behavior of the S resource is undefined. The S resource will neither perform the counter instruction correctly nor the capture buffer functionality is operated correctly.

27.5.10.8 S Resource Counter Instructions

This instruction allows to set up a counter, this can be used to perform measurements on signals or use it as a resource to generate periodic signals with definable length.

Following inputs are taken into account:

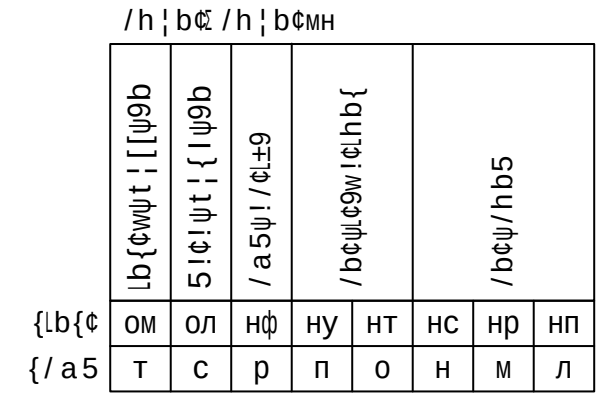
- *S_OUT* [c:c] : value, falling edge; rising edge of **TIO[i]_S.CH[x]** x = 8*g+c
- *PL_UPDATE* [c:c] : resolution for operation
- *S_TRIG_OUT* [c:c] : start trigger event; see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- *PL_TRIG_OUT* [c:c]: stop trigger event
- *CYCLIC_INIT_TRIG* [c:c]: init trigger event fo cyclic_buffer *68 "CYCLIC_INIT_TRIG[c:c] Enable Function "*

This mode is selected with **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** = 0b10 | 0b11.

In the S resource counter mode, the following bit fields exist in the command register:

- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_COND** : defines count condition
- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** : defines number of count iterations
- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CMD_ACTIVE** : command active (counting in progress / counting stopped)
- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.INSTR_PULL_EN** : enable instruction pull from instruction buffer
- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.DATA_PUSH_EN** : enable data push to capture buffer

Figure 195 CMD Register Layout – Count Instruction



They can be accessed via the multiview register names:

- **TIO[i]_G[g]_CH[c]_SCMD_COUNT**
- **TIO[i]_G[g]_CH[c]_SINST_COUNT**
- **TIO[i]_G[g]_CH[c]_SINST_COUNT12**

The bit field **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CMD_ACTIVE** in the command register controls the counting of the instruction:

- 0 : counting stopped
- 1 : counting active; register **TIO[i]_G[g]_CH[c]_SINST.OP** will be increased / decreased depending on **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_COND** (table *78 "CNT_COND Definition"*)

The execution of count instructions can be influenced with the following configuration bits, the corresponding functions will be explained in the referred sections:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 1 : see chapter *27.5.10.8.6 "Instruction Termination With Instruction Freeze"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_S_TRIG_OUT_EN** = 1 : see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 : see chapter *27.5.10.8.8 "Instruction Termination With Cyclic Buffer Usage"*
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN** = 1 : see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- **TIO[i]_G[g]_CH[c]_SCMD.INSTR_PULL_EN** = 1 instruction buffer usage : see chapter *27.5.10.8.7 "Instruction Termination With Instruction Reload"*
- **TIO[i]_G[g]_CH[c]_SCMD.DATA_PUSH_EN** = 1 capture buffer usage : see chapter *27.5.10.8.9 "Capture Buffer Usage On Instruction Termination"*

For the functional description in the next chapters (*27.5.10.8.1 "Select Counting Condition of Instruction"* – *27.5.10.8.3 "Count Instruction Stopping Disabled"*) is assumed:

- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 0
- **TIO[i]_G[g]_CH[c]_CTRL.PL_S_TRIG_OUT_EN** = 1
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN** = 0
- **TIO[i]_G[g]_CH[c]_SCMD.INSTR_PULL_EN** = 0
- **TIO[i]_G[g]_CH[c]_SCMD.DATA_PUSH_EN** = 0

27.5.10.8.1 Select Counting Condition of Instruction

The counting is configured with the bit field CNT_COND in the command register.

Table 78 CNT_COND Definition

CNT_COND	Count behavior	Description

0b000	IF *PL_UPDATE* [c:c]='1' THEN
	IF *S_OUT* [c:c]='0' THEN
	TIO[i]_G[g]_CH[c]_SINST.OP = **TIO[i]_G[g]_CH[c]_SINST.OP** +1;
	ELSE
	TIO[i]_G[g]_CH[c]_SINST.OP = **TIO[i]_G[g]_CH[c]_SINST.OP** −1;
	END IF;
	END IF;
		Increment on update resolution if signal is low;
		Decrement on update resolution if signal is high

0b001	IF *PL_UPDATE* [c:c]='1' AND NOT(*S_OUT* [c:c])='1' THEN
	TIO[i]_G[g]_CH[c]_SINST.OP = **TIO[i]_G[g]_CH[c]_SINST.OP** +1;
	END IF;
		Increment on update resolution if signal is low

0b010	IF *PL_UPDATE* [c:c]='1' AND *S_OUT* [c:c]='1' THEN
	TIO[i]_G[g]_CH[c]_SINST.OP = **TIO[i]_G[g]_CH[c]_SINST.OP** +1;
	END IF;
		Increment on update resolution if signal is high

0b011	IF *PL_UPDATE* [c:c]='1' THEN
	TIO[i]_G[g]_CH[c]_SINST.OP = **TIO[i]_G[g]_CH[c]_SINST.OP** +1;
	END IF;
		Increment on update resolution

0b100	reserved	

0b101	IF falling edge *S_OUT* [c:c] THEN
	TIO[i]_G[g]_CH[c]_SINST.OP = **TIO[i]_G[g]_CH[c]_SINST.OP** +1;
	END IF;
		Increment on each falling edge

0b110	IF rising edge *S_OUT* [c:c] THEN
	TIO[i]_G[g]_CH[c]_SINST.OP = **TIO[i]_G[g]_CH[c]_SINST.OP** +1;
	END IF;
		Increment on each rising edge

CNT_COND	Count behavior	Description
0b111	IF edge *S_OUT* [c:c] THEN **TIO[i]_G[g]_CH[c]_SINST.OP** = **TIO[i]_G[g]_CH[c]_SINST.OP** +1; END IF;	Increment on any edge

In case one of the modes CNT_COND = 0b000 | 0b001 | 0b010 | 0b011 is selected, the register **TIO[i]_G[g]_CH[c]_SINST.OP** will be incremented / decremented based on the selected resolution *PL_UPDATE* [c:c].

In case one of the modes **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_COND** = 0b101 | 0b110 | 0b111 is selected the register **TIO[i]_G[g]_CH[c]_SINST.OP** will be incremented once on each occurred edge of *S_OUT* [c:c].

27.5.10.8.2 Control Iterations of Count Instruction

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** = 0b10

How long a counting instruction is executed is configurable with the bit field **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** in the command register:

- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** = 0: no further activation; counting is already active or stopped
- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** = 1: counting gets active 1 time
- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** = 2: counting gets active 2 times
- **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** = 3: counting is active endless

On every writing of the command register by the configuration interface, the instruction initializes the internal register *ITERA_CNT[c]* to **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** .

The count instruction behavior is defined by the occurrence of the start / stop trigger events (*S_TRIG_OUT* [c:c] / *PL_TRIG_OUT* [c:c]) and the actual state of the instruction *ITERA_CNT[c]* , *CMD_ACTIVE_COUNT* [c:c]. The progress of a count instruction is executed based on a priority scheme, the ordering of execution priorities are defined subsequently:

1. PRIO0 condition is validated first, if the condition matches, the corresponding PRIO0 actions are executed, otherwise continue with PRIO1 validation.
2. PRIO1 condition is validated first, if the condition matches, the corresponding PRIO1 actions are executed, otherwise continue with PRIO2 validation.
3. PRIO2 condition is validated next, if the condition matches, the corresponding PRIO2 actions are executed, otherwise continue with PRIO3 validation.
4. PRIO3 condition is validated next, if the condition matches, the corresponding PRIO3 actions are executed, otherwise continue with PRIO4 validation.
5. PRIO4 condition is validated next, if the condition matches, the corresponding PRIO4 actions are executed

PRIO0 condition: If **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 and (*CYCLIC_INIT_TRIG* [c:c]= 1 or *PL_TRIG_OUT* [c:c] = 1); an asynchronous event for the cyclic buffer is active, the actual instruction is canceled.

o PRIO0 actions:

- Execute command register reload actions explained in chapters *27.5.10.13 "Cyclic Instruction Buffer Usage"* ; including the **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** state for further execution;

PRIO1 condition: If *ITERA_CNT[c]* = 0 and **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1 and (*S_TRIG_OUT* [c:c] = 1 or *PL_TRIG_OUT* [c:c] = 1); A start or stop trigger event occurs and terminates the instruction.

o PRIO1 actions:

- Set the terminating condition *INSTR_END* = 1 for 1 clock cycle;
- **TIO[i]_G[g]_CH[c]_SINST.OP** will not increment / decrement in this state
- Based on **TIO[i]_G[g]_CH[c]_SCMD.DATA_PUSH_EN** follow functional description explained in *27.5.10.8.9 "Capture Buffer Usage On Instruction Termination"*
- Based on **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** =1 execute command register reload actions explained in chapters *27.5.10.8.6 "Instruction Termination With Instruction Freeze"* ; including the **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** state for further execution;
- Based on **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1 execute command register reload actions explained in chapters *27.5.10.8.8 "Instruction Termination With Cyclic Buffer Usage"* ; including the **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** state for further execution;
- Based on **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 1 execute command register reload actions explained in chapters *27.5.10.8.7 "Instruction Termination With Instruction Reload"* ; including the **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** state for further execution;

- Execute command register reload actions explained in chapters *27.5.10.8.5 "Instruction Termination Without Instruction Reload"* ; including the **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** state for further execution;

PRIO2 condition: If *ITERA_CNT[c]* = 3 and *S_TRIG_OUT* [c:c] = 1; A start trigger occurs in endless counting

o PRIO2 actions:

- In case **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1 and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 0: set **TIO[i]_G[g]_CH[c]_SINST.OP** = 0.
- In case **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 0 and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 0: set **TIO[i]_G[g]_CH[c]_SINST.OP** = 0; **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1.
- In case **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1 and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 1: set **TIO[i]_G[g]_CH[c]_SINST.OP** = *OOP_PREV* .
- In case **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 0 and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 1: set **TIO[i]_G[g]_CH[c]_SINST.OP** = *OOP_PREV* ; set **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1.

PRIO3 condition: If (*ITERA_CNT[c]* = 1 or *ITERA_CNT[c]* = 2) and *S_TRIG_OUT* [c:c] = 1; A start trigger occurs iteration counter will be decremented and counting of instruction is started

o PRIO3 actions:

- In case **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1 and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 0: set *ITERA_CNT[c]* = *ITERA_CNT[c]* − 1; set **TIO[i]_G[g]_CH[c]_SINST.OP** = 0.
- In case **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 0 and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 0: set *ITERA_CNT[c]* = *ITERA_CNT[c]* − 1; set **TIO[i]_G[g]_CH[c]_SINST.OP** = 0; set **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1.
- In case **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1 and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 1: set *ITERA_CNT[c]* = *ITERA_CNT[c]* − 1; set **TIO[i]_G[g]_CH[c]_SINST.OP** = *OOP_PREV* .
- In case **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 0 and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 1: set *ITERA_CNT[c]* = *ITERA_CNT[c]* − 1; set **TIO[i]_G[g]_CH[c]_SINST.OP** = *OOP_PREV* ; set **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1.

PRIO4 condition: If *PL_TRIG_OUT* [c:c] = 1 and **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 1 ; A stop trigger occurs and counting of instruction is stopped

o PRIO4 actions:

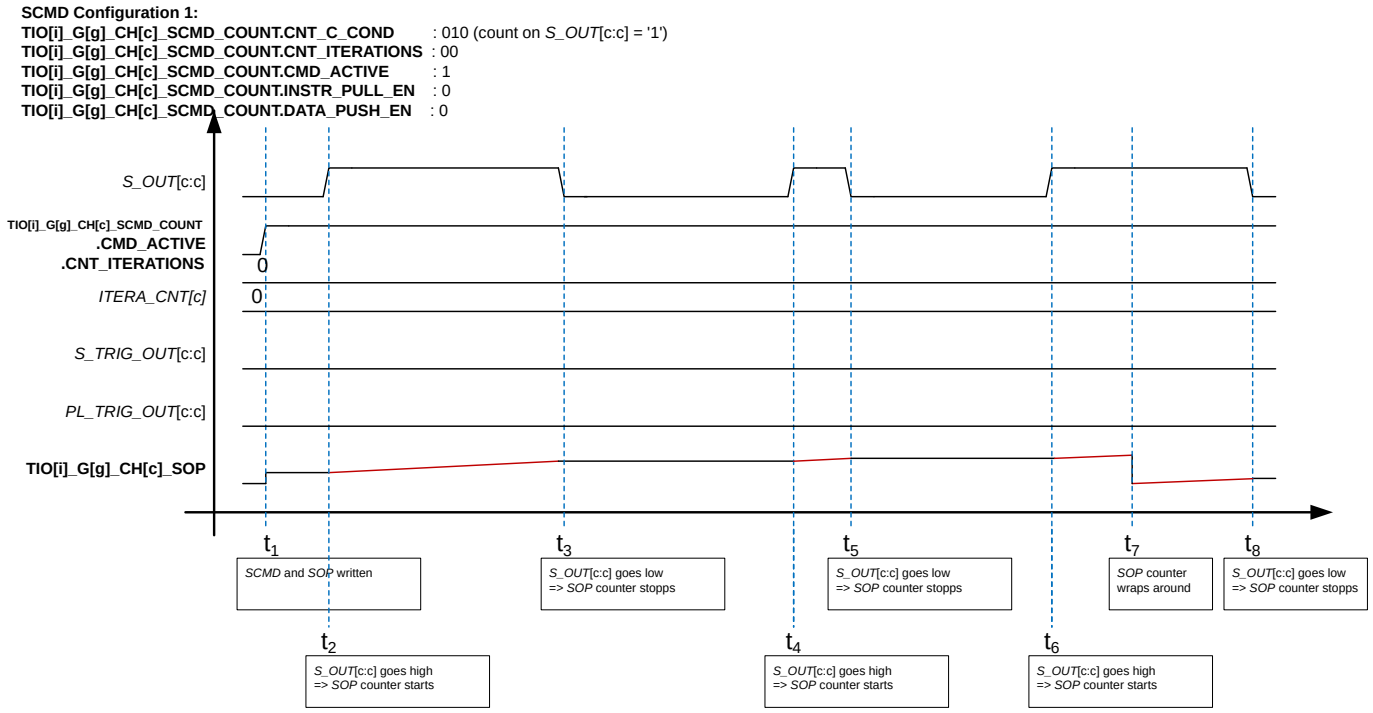
- Set **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 0
- **TIO[i]_G[g]_CH[c]_SINST.OP** will increment / decrement in this state

Find next some application examples.

TIO[i]_G[g]_CH[c]_SINST_COUNT.CNT_ITERATIONS = 0; no start / no stop trigger event:

Example: (see figure *196 "Endless Counting Signal S_OUT[c:c] = 1"*) Increment register **TIO[i]_G[g]_CH[c]_SINST.OP** on *PL_UPDATE* [c:c] resolution if condition *S_OUT* [c:c] = 1 is met, counting will never end due to not occurrence of start trigger event *S_TRIG_OUT* [c:c] = 0 or stop trigger event *PL_TRIG_OUT* [c:c] = 0.

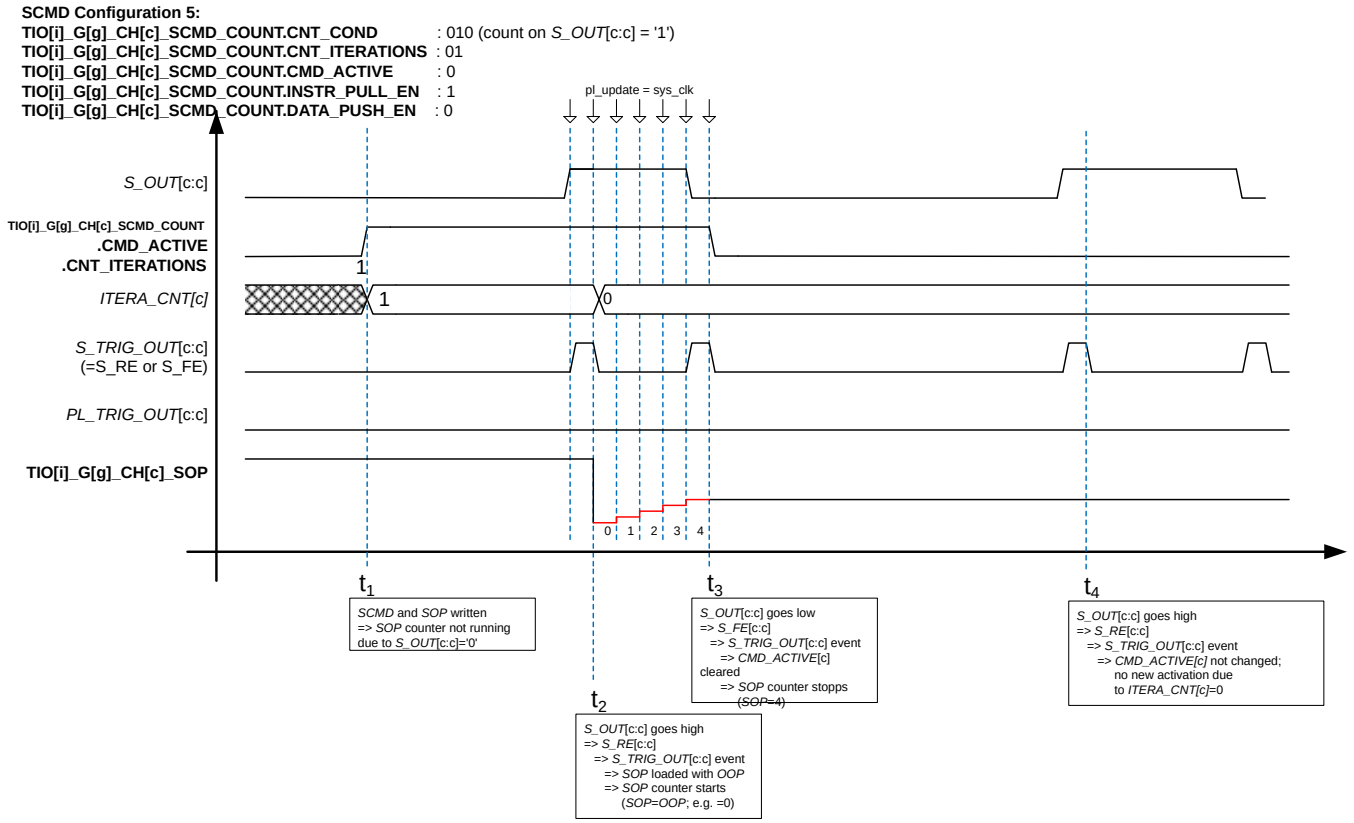
Figure 196 Endless Counting Signal S_OUT[c:c] = 1



TIO[i]_G[g]_CH[c]_SINST_COUNT.CNT_ITERATIONS = 1 | 2; no stop trigger event:

Example: (see figure *197 "Counting Duration of 1 High Pulse Signal S_OUT[c:c] "*) count in **TIO[i]_G[g]_CH[c]_SINST.OP** high pulse length based on *PL_UPDATE* [c:c] resolution. Perform this counting for 1 high pulse only.

Figure 197 Counting Duration of 1 High Pulse Signal S_OUT[c:c]

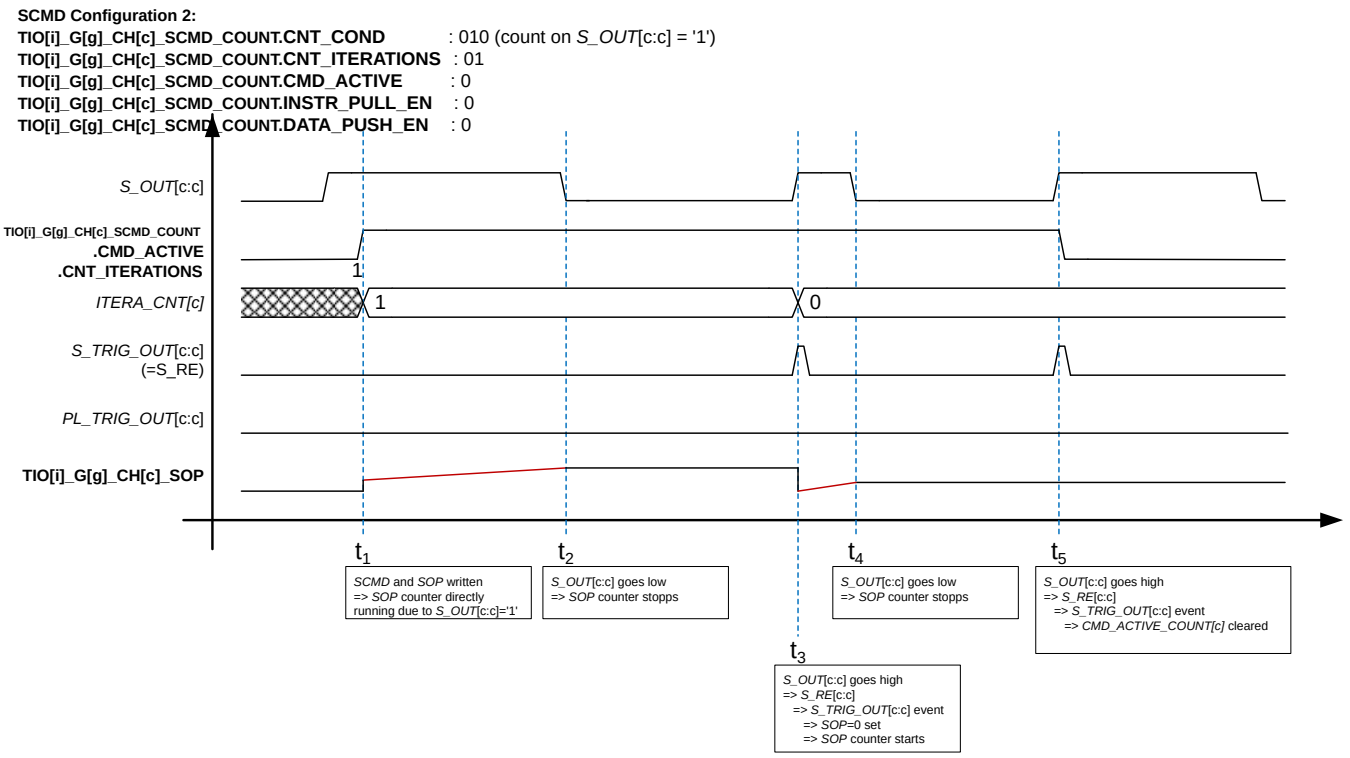


Note:

Using the count instruction to measure the duration of a pulse or period by using *S_TRIG_OUT* [c:c] = *S_RE* / *S_FE* as start /stop triggers the duration of the measured input signal has to be adjusted by 1 if *PL_UPDATE* [c:c] = cls_clk. Duration of pulse / period = **TIO[i]_G[g]_CH[c]_SINST.OP** + 1.
This is due to the fact that the event *S_TRIG_OUT* starting the counting initiates a reload of the **TIO[i]_G[g]_CH[c]_SINST.OP** with 0. In this

reload state **TIO[i]_G[g]_CH[c]_SINST.OP** will not be incremented.
If *PL_UPDATE* [c:c] has a lower resolution than the cluster clock, a reload of the counter and incrementing the counter is possible within a *PL_UPDATE* [c:c] period.

Figure 198 Counting Duration of 2 Pulses Signal S_OUT[c:c] = 1

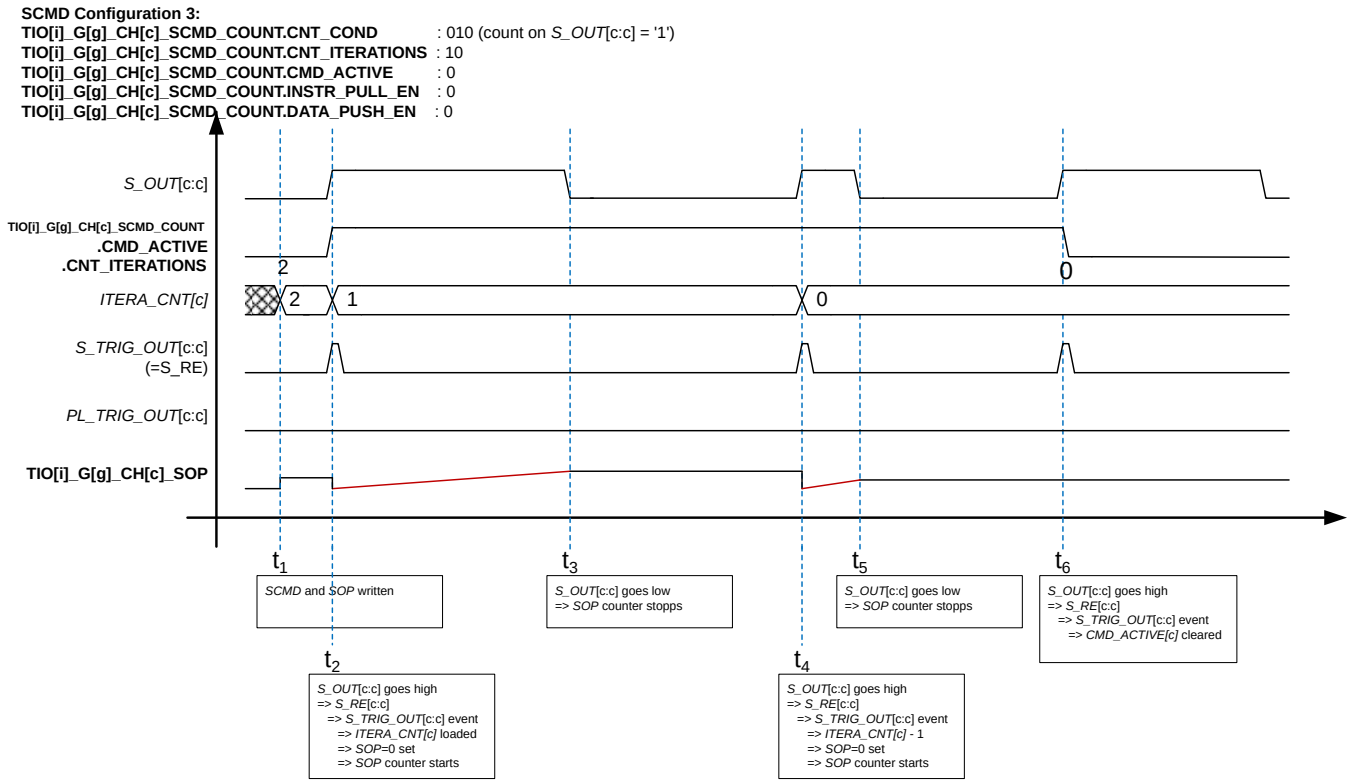


Due to writing-in of **TIO[i]_G[g]_CH[c]_SINST.CMD** / **TIO[i]_G[g]_CH[c]_SINST.OP** at t1 while *S_OUT* [c:c]= 1 is fulfilled, **TIO[i]_G[g]_CH[c]_SINST.OP** is incremented immediately until *S_OUT* [c:c]= 0 at t2 stops counting.

Next example shows how it can be ensured that counting is started with rising edge of *S_OUT* [c:c] (see figure *199 "Counting 2 Pulses Signal S_OUT[c:c] = 1; Start with S_OUT[c:c] Rising Edge"*).

In this case, the instruction is written at t1 with **TIO[i]_G[g]_CH[c]_SINST_COUNT.CMD_ACTIVE** = 0.

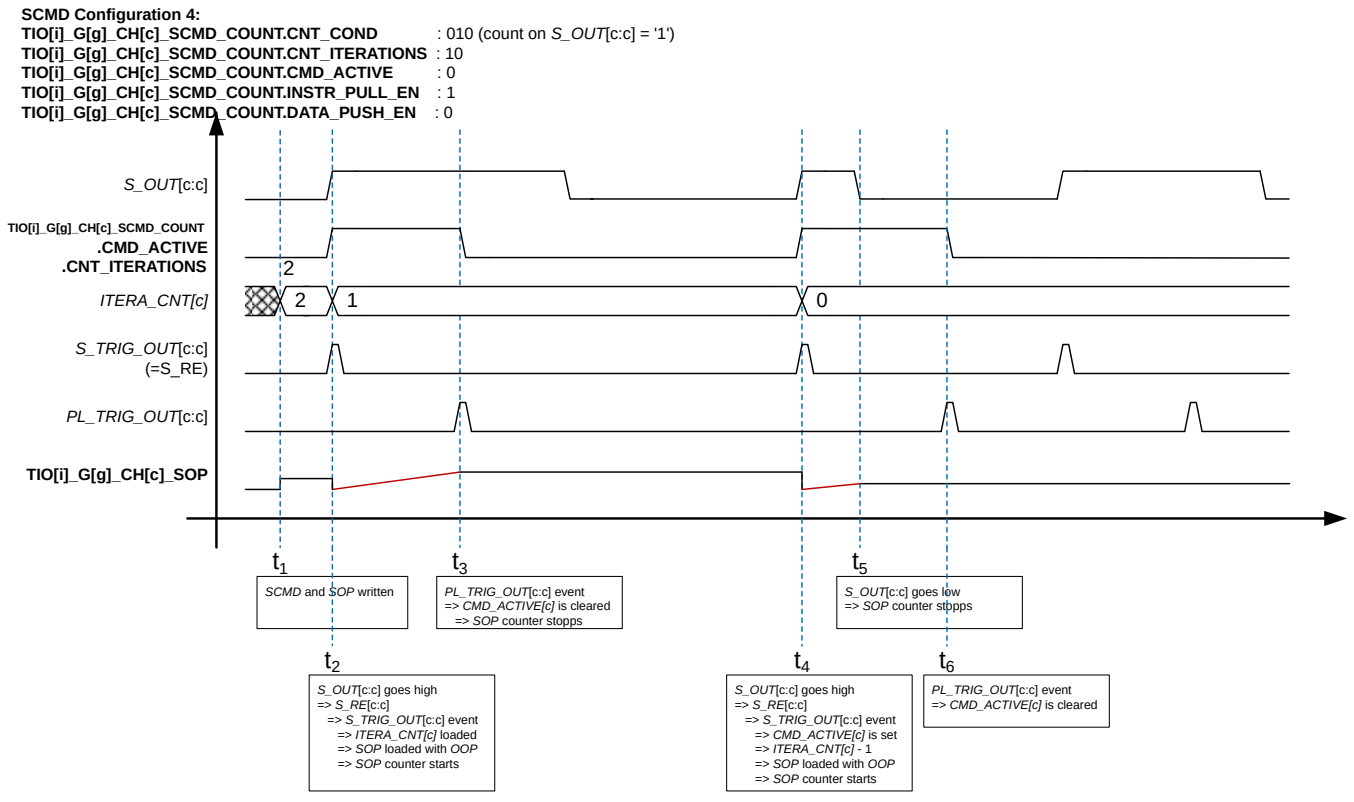
Figure 199 Counting 2 Pulses Signal S_OUT[c:c] = 1; Start with S_OUT[c:c] Rising Edge



TIO[i]_G[g]_CH[c]_SINST_COUNT.CNT_ITERATIONS = 1 | 2; start and stop trigger event:

Example: (see figure *200 "Counting 2 Pulses Signal S_OUT[c:c] = 1; Start with S_OUT[c:c] Rising Edge; Stop on PL_TRIG_OUT[c:c]"*) count in **TIO[i]_G[g]_CH[c]_SINST.OP** high pulse length based on *PL_UPDATE* [c:c] resolution. Counting will be stopped by *PL_TRIG_OUT* [c:c] event.

Figure 200 Counting 2 Pulses Signal S_OUT[c:c] = 1; Start with S_OUT[c:c] Rising Edge; Stop on PL_TRIG_OUT[c:c]



TIO[i]_G[g]_CH[c]_SINST_COUNT.CNT_ITERATIONS = 3:

Note:

This counter instruction will never terminate. A termination can be realized by writing the **TIO[i]_G[g]_CH[c]_SINST_COUNT.CNT_ITERATIONS** bit field in the command register to a value != 3. The internal register *ITERA_CNT[c]* = **TIO[i]_G[g]_CH[c]_SINST_COUNT.CNT_ITERATIONS** will be set and the count instruction will execute as specified in *27.5.10.8.2 "Control Iterations of Count Instruction"* .

27.5.10.8.3 Count Instruction Stopping Disabled

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** = 0b11

If a count instruction is executed in this mode, all events on *PL_TRIG_OUT* [c:c] are ignored, counting will never stop on *PL_TRIG_OUT* [c:c] = 1.

All rules related to starting / terminating a count instruction based on a start trigger event *S_TRIG_OUT* [c:c] will behave as pointed out in chapter *27.5.10.8.2 "Control Iterations of Count Instruction"* .

27.5.10.8.4 PL_EVT[c:c] Generation

Any counter instruction terminating with *INSTR_END* = 1 will never influence the *PL_EVT* [c:c] condition. The *PL_EVT* [c:c] condition is generated by the O resource instruction termination.

27.5.10.8.5 Instruction Termination Without Instruction Reload

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_SCMD.INSTR_PULL_EN** = 0

If the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CMD_ACTIVE** = 0 (STOP_INSTRUCTION) is set, **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** = 0 is set, **TIO[i]_G[g]_CH[c]_SINST.OP** will not change.
A next instruction is not pulled, *INSTR_PULL* = 0.

A next instruction can only be issued by writing **TIO[i]_G[g]_CH[c]_SCMD** with a new instruction command.

27.5.10.8.6 Instruction Termination With Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_SCMD.INSTR_PULL_EN** = 0|1

If the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_SINST.CMD** will remain unchanged. **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CMD_ACTIVE** will not be changed due to instruction freezing. On *INSTR_END* =1, the instruction initializes the internal register *ITERA_CNT[c]* to **TIO[i]_G[g]_CH[c]_SCMD_COUNT.CNT_ITERATIONS** .
A next instruction is not pulled, *INSTR_PULL* = 0.

If case **TIO[i]_G[g]_CH[c]_SCMD.INSTR_PULL_EN** =0 and the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_SINST.OP** = 0 will be set.

If case **TIO[i]_G[g]_CH[c]_SCMD.INSTR_PULL_EN** =1 and the instruction terminates with *INSTR_END* = 1, **TIO[i]_G[g]_CH[c]_SINST.OP** = *OOP_PREV* will be set.

This results in an endless execution of the command in **TIO[i]_G[g]_CH[c]_SINST.CMD** .

27.5.10.8.7 Instruction Termination With Instruction Reload

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0; **TIO[i]_G[g]_CH[c]_SCMD.INSTR_PULL_EN** = 1

If the instruction terminates with *INSTR_END* = 1, the next instruction will be loaded from instruction buffer:
TIO[i]_G[g]_CH[c]_SINST.CMD = *OCMD_PREV* ; **TIO[i]_G[g]_CH[c]_SINST.OP** = *OOP_PREV*

On *INSTR_END* =1, the instruction initializes the internal register *ITERA_CNT[c]* to *OCMD_PREV* .

A next instruction is pulled, *INSTR_PULL* = *INSTR_END* .

The new loaded counter instruction is immediately executed.

27.5.10.8.8 Instruction Termination With Cyclic Buffer Usage

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 0; **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1

If the instruction terminates with *INSTR_END* = 1 and (*PL_TRIG_OUT* [c:c]=1 or *CYCLIC_INIT_TRIG* [c:c]=1) the next instruction will be loaded from the cyclic instruction buffer as defined in chapters *27.5.10.13.1 "Exchange Trigger Behavior"* and *27.5.10.13.2 "Init Trigger Behavior"* .

If the instruction terminates with *INSTR_END* = 1 and *PL_TRIG_OUT* [c:c]=0 and *CYCLIC_INIT_TRIG* [c:c]=0, the next instruction will be loaded from the O resource of the cyclic buffer as defined next:

- **TIO[i]_G[g]_CH[c]_SINST.CMD** (t+1) = **TIO[i]_G[g]_CH[c]_OINST.CMD** (t);
- **TIO[i]_G[g]_CH[c]_SINST.OP** (t+1) = **TIO[i]_G[g]_CH[c]_OINST.OP** (t);
- **TIO[i]_G[g]_CH[c]_OINST.CMD** (t+1) = **TIO[i]_G[g]_CH[c]_SINST.CMD** (t);
- **TIO[i]_G[g]_CH[c]_OINST.OP** (t+1) = **TIO[i]_G[g]_CH[c]_SINST.OP** (t);
- On *INSTR_END* =1, the instruction initializes the internal register *ITERA_CNT[c]* (t+1) to **TIO[i]_G[g]_CH[c]_OCMD.CMD** [4:3](t).
- A next instruction is not pulled, *INSTR_PULL* = 0.

This configuration uses the cyclic instruction buffer of the channel resource (details see chapter *27.5.10.13 "Cyclic Instruction Buffer Usage"*), a next instruction is never pulled via the signal *INSTR_PULL* from the O resource of the channel [c−1]. Signal *INSTR_PULL* = 0 never changes.

27.5.10.8.9 Capture Buffer Usage On Instruction Termination

No capture buffer in use:

Configuration in use: **TIO[i]_G[g]_CH[c]_SCMD.DATA_PUSH_EN** = 0

If the instruction terminates with *INSTR_END* = 1, no data push request is issued.
DATA_PUSH = 0.

Capture buffer in use:

To enable that data from the S resource of channel [c] can be captured in the O resource of the same channel, the O resource has to be configured as a continue buffer.

Configuration in use: **TIO[i]_G[g]_CH[c]_SCMD.DATA_PUSH_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b001

If the instruction terminates with *INSTR_END* = 1, a data push request is issued.
DATA_PUSH = *INSTR_END* .

Note:
In case of **TIO[i]_G[g]_CH[c]_SINST.DATA_PUSH_EN** = 1; **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 2 | 3 | 4 | 5 | 6 | 7, the behavior of the O resource is undefined. The O resource will neither perform the instruction correctly nor the capture buffer functionality is operated correctly.

27.5.10.9 S Resource Start Buffer

The S resource operates as a start buffer if **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** = 0b00 is set.
In this case the S resource in channel [c] stores an instruction: 32−bit data (8−bit command; 24−bit operand). This instruction can be used to reload the actual instruction in the O resource of channel [c] using an *INSTR_PULL* =1 request (see chapters *27.5.10.5.6 "Instruction Termination With Instruction Reload"* , *27.5.10.6.8 "Instruction Termination With Instruction Reload"* , *27.5.10.7.17 "Instruction Termination With Instruction Reload"*).
An instruction buffer with length = 1 is generated for an instruction executed in the O resource of this channel.

After a reset the internal state *S_BUFF_FILLED* is set to 0. This flag is used internally to be able to generate the control signal *S_BUFFER_EMPTY* (details see chapter *27.5.10.9.3 "PL_EVT Generation"*).

Every write to the registers **TIO[i]_G[g]_CH[c]_SCMD** and **TIO[i]_G[g]_CH[c]_SINST** sets the internal state of the buffer *S_BUFF_FILLED* = 1.

Any *O_DATA_PUSH_PREV* = 1 request to store data from the O resource of channel [c−1] will be ignored. A start buffer can never store data from the previous resource.

27.5.10.9.1 Start Buffer Behavior Without Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 0

o Behavior on *S_BUFF_FILLED* = 1:

- On each *S_INSTR_PULL_NEXT* = 1 request, the content of the buffer is consumed:

- Internal state changes to *S_BUFF_FILLED* = 0
- Operand register is loaded with **TIO[i]_G[g]_CH[c]_SINST.OP** = 0x0
- Command register is loaded with **TIO[i]_G[g]_CH[c]_SINST.CMD** = 0x0 (stop instruction)
- Due to start buffer configuration signal *O_INSTR_PULL* = 0 will be driven

o Behavior on *S_BUFF_FILLED* = 0:

- On each *S_INSTR_PULL_NEXT* = 1 request, the buffer status will not change:

- Internal state *S_BUFF_FILLED* = 0 will not change
- Operand register **TIO[i]_G[g]_CH[c]_SINST.OP** = 0x0 will not change
- Command register **TIO[i]_G[g]_CH[c]_SINST.CMD** = 0x0 will not change
- Due to start buffer configuration signal *O_INSTR_PULL* = 0 will be driven

27.5.10.9.2 Start Buffer Behavior With Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 1

The content of the buffer will be freezed, any request *S_INSTR_PULL_NEXT* = 1 will never empty the buffer:

- Internal state *S_BUFF_FILLED* = 1 will never change
- Operand register **TIO[i]_G[g]_CH[c]_SINST.OP** will not change
- Command register **TIO[i]_G[g]_CH[c]_SINST.CMD** will not change
- Due to start buffer configuration signal *INSTR_PULL* = 0 will be driven

27.5.10.9.3 PL_EVT Generation

For a start buffer resource the condition *S_BUFFER_EMPTY* is defined as:
S_BUFFER_EMPTY = 1 for 1 system clock, if internal state *S_BUFF_FILLED* changes from 1 -> 0

Depending on the configured resources in a channel the condition S_BUFFER_EMPTY is used as *PL_EVT* (details see table *65 "PL_EVT[c:c] Selection Dependent On TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE / TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE"*).

27.5.10.10 S Resource Continue Buffer

The S resource operates as a continue buffer if **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** = 0b01 is set.

In this case the S resource in channel [c] stores an instruction: 32-bit data (8-bit command; 24-bit operand). This instruction can be used to reload the actual instruction in the O resource of channel [c] using a *INSTR_PULL* =1 request (see chapters *27.5.10.5.6 "Instruction Termination With Instruction Reload"* , *27.5.10.6.8 "Instruction Termination With Instruction Reload"* , *27.5.10.7.17 "Instruction Termination With Instruction Reload"*).

A continue buffer can be used for 2 purposed:

- Instruction buffer: using a start buffer followed by n continue buffers allows to define an instruction buffer with length n+1 for an instruction executed in the O resource.
- Capture buffer: using the O resource of channel [c-1] for instruction execution, and using a continue buffer S resource in channel [c], a capture buffer of length 1 is established. This capture buffer can be continued with a further O resource in the channel [c] to extend the capture buffer to length = 2. Capture buffers with length >2 can be generated by using capture buffers in the following channels k (k > c).

After a reset the internal state *S_BUFF_FILLED* is set to 0. This flag is used internally to be able to generate the control signal *S_BUFFER_EMPTY* (details see chapter *27.5.10.10.3 "PL_EVT Generation"*).

Every write to the registers **TIO[i]_G[g]_CH[c]_SCMD** and **TIO[i]_G[g]_CH[c]_SINST** sets the internal state of the buffer *S_BUFF_FILLED* = 1.

27.5.10.10.1 Continue Buffer Behavior Without Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 0

Behavior on *S_BUFF_FILLED* = 1:

On each *S_INSTR_PULL_NEXT* = 1 request, the content of the buffer is loaded with new values:

- Operand register is loaded with **TIO[i]_G[g]_CH[c]_SINST.OP** [c] = *OOP_PREV*
- Command register is loaded with **TIO[i]_G[g]_CH[c]_SINST.CMD** [c] = *OCMD_PREV*
- Due to continue buffer configuration signal *O_INSTR_PULL* = *S_INSTR_PULL_NEXT* will be driven.

- If *OCMD_PREV* = 0x0 the internal state *S_BUFF_FILLED* = 0 will be set.

Behavior on *S_BUFF_FILLED* = 0:

On each *S_INSTR_PULL_NEXT* = 1 request, the buffer status will not change:

- Internal state *S_BUFF_FILLED* = 0 will not change
- Operand register **TIO[i]_G[g]_CH[c]_SINST.OP** = 0x0 will not change
- Command register **TIO[i]_G[g]_CH[c]_SINST.CMD** = 0x0 will not change
- Due to continue buffer configuration signal *O_INSTR_PULL* = *S_INSTR_PULL_NEXT* will be driven.

On each *O_DATA_PUSH_PREV* = 1 request, the content of the buffer is loaded with new values:

- Operand register is loaded with **TIO[i]_G[g]_CH[c]_SINST.OP** = *OOP_PREV*
- Command register is loaded with **TIO[i]_G[g]_CH[c]_SINST.CMD** = *OCMD_PREV*
- Due to continue buffer configuration signal *S_DATA_PUSH* = *O_DATA_PUSH_PREV* will be driven.

Any continue buffer can only be used exclusively as instruction buffer (using *S_INSTR_PULL_NEXT* = 1 request) or capture buffer (using *O_DATA_PUSH_PREV* =1 requests). Only in case of incorrect usage of continue buffers, both requests could occur. This can lead to an unexpected operation of the application.

27.5.10.10.2 Continue Buffer Behavior With Instruction Freeze

The configuration **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** = 0b01; **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 1 is illegal!

A valid configuration can be established by using a start buffer with instruction freeze *27.5.10.9.2 "Start Buffer Behavior With Instruction Freeze"* .

27.5.10.10.3 PL_EVT Generation

For a continue buffer resource the condition *S_BUFFER_EMPTY* is defined as:
S_BUFFER_EMPTY = 1 for 1 system clock, if internal state *S_BUFF_FILLED* changes from 1 -> 0

Depending on the configured resources in a channel the condition *S_BUFFER_EMPTY* is used as *PL_EVT* [c:c] (details see table *65 "PL_EVT[c:c] Selection Dependent On TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE / TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE"*).

27.5.10.11 O Resource Start Buffer

The O resource operates as a start buffer if **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b000 is set.

In this case the O resource of channel [c] stores an instruction: 32-bit data (8-bit command; 24-bit operand). This instruction can be used to reload the actual instruction in the S resource of channel [c+1] using a *INSTR_PULL* =1 request (see chapters *27.5.10.8.7 "Instruction Termination With Instruction Reload"*).

An instruction buffer with length = 1 is generated for an instruction executed in the S resource of the next channel.

After a reset the internal state *O_BUFF_FILLED* is set to 0. This flag is used internally to be able to generate the control signal *O_BUFFER_EMPTY* (details see chapter *27.5.10.11.3 "PL_EVT Generation"*).

Every write to the registers **TIO[i]_G[g]_CH[c]_OCMD** and **TIO[i]_G[g]_CH[c]_OINST** sets the internal state of the buffer *O_BUFF_FILLED* = 1.

Any *S_DATA_PUSH_PREV* = 1 request to store data from the S resource of the channel [c] will be ignored. A start buffer can never store data from the S resource of channel [c].

27.5.10.11.1 Start Buffer Behavior Without Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0

Behavior on *O_BUFF_FILLED* = 1:

On each *O_INSTR_PULL_NEXT* = 1 request, the content of the buffer is consumed:

- Internal state gets changed to *O_BUFF_FILLED* = 0
- Operand register gets loaded with **TIO[i]_G[g]_CH[c]_OINST.OP** = 0x0
- Command register gets loaded with **TIO[i]_G[g]_CH[c]_OINST.CMD** = 0x0 (stop instruction)

- Due to start buffer configuration signal *S_INSTR_PULL* = 0 will be driven

Behavior on *O_BUFF_FILLED* = 0:

On each *O_INSTR_PULL_NEXT* = 1 request, the buffer status will not change:

- Internal state *O_BUFF_FILLED* = 0 will not change
- Operand register **TIO[i]_G[g]_CH[c]_OINST.OP** = 0 will not change
- Command register **TIO[i]_G[g]_CH[c]_OINST.CMD** = 0x0 will not change
- Due to start buffer configuration signal *S_INSTR_PULL* = 0 will be driven

27.5.10.11.2 Start Buffer Behavior With Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1

The content of the buffer will be freezed, any request *O_INSTR_PULL_NEXT* = 1 will never empty the buffer:

- Internal state *O_BUFF_FILLED* = 1 will never change
- Operand register **TIO[i]_G[g]_CH[c]_OINST.OP** will not change
- Command register **TIO[i]_G[g]_CH[c]_OINST.CMD** will not change
- Due to start buffer configuration signal *S_INSTR_PULL* = 0 will be driven

27.5.10.11.3 PL_EVT Generation

For a start buffer resource the condition *O_BUFFER_EMPTY* is defined as:
O_BUFFER_EMPTY = 1 for 1 system clock, if internal state *O_BUFF_FILLED* changes from 1 –> 0

Depending on the configured resources in a channel the condition *O_BUFFER_EMPTY* is used as *PL_EVT* [c:c] (details see table *65 "PL_EVT[c:c] Selection Dependent On TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE / TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE"*).

27.5.10.12 O Resource Continue Buffer

The O resource operates as a continue buffer if **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b001 is set.

In this case the O resource in channel [c] stores an instruction: 32 bit data (8 bit command; 24 bit operand). This instruction can be used to reload the actual instruction in the S resource of channel [c+1] using a *O_INSTR_PULL* =1 request (see chapters *27.5.10.5.6 "Instruction Termination With Instruction Reload"* , *27.5.10.6.8 "Instruction Termination With Instruction Reload"* , *27.5.10.7.17 "Instruction Termination With Instruction Reload"*).

A continue buffer can be used for 2 purposes:

- Instruction buffer using a start buffer followed by n continue buffers allows to define an instruction buffer with length n+1 for an instruction executed in the S resource.
- Capture buffer: using the S resource of channel [c] for instruction execution, and using a continue buffer O resource in channel [c], a capture buffer of length 1 is established. This capture buffer can be continued with a further S resource in the channel [c+1] to extend the capture buffer to length = 2. Capture buffers with length >2 can be generated by using capture buffers in the following channels k (k > c).

After a reset the internal state *O_BUFF_FILLED* is set to 0. This flag is used internally to be able to generate the control signal *O_BUFFER_EMPTY* (details see chapter *27.5.10.12.3 "PL_EVT Generation"*).

Every write to the registers **TIO[i]_G[g]_CH[c]_OCMD** and **TIO[i]_G[g]_CH[c]_OINST** sets the internal state of the buffer *O_BUFF_FILLED* = 1.

27.5.10.12.1 Continue Buffer Behavior Without Instruction Freeze

Configuration in use: **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0

Behavior on *O_BUFF_FILLED* = 1:

On each *O_INSTR_PULL_NEXT* = 1 request, the content of the buffer is loaded with new values:

- Operand register gets loaded with **TIO[i]_G[g]_CH[c]_OINST.OP** = **TIO[i]_G[g]_CH[c]_SINST.OP**
- Command register gets loaded with **TIO[i]_G[g]_CH[c]_OINST.CMD** = **TIO[i]_G[g]_CH[c]_SINST.CMD**
- Due to continue buffer configuration signal *S_INSTR_PULL* = *O_INSTR_PULL_NEXT* will be driven.

- If **TIO[i]_G[g]_CH[c]_SINST.CMD** = 0x0 the internal state *O_BUFF_FILLED* = 0 will be set.

Behavior on *O_BUFF_FILLED* = 0:

On each *O_INSTR_PULL_NEXT* = 1 request, the buffer status will not change:

- Internal state *O_BUFF_FILLED* = 0 will not change
- Operand register **TIO[i]_G[g]_CH[c]_OINST.OP** = 0x0 will not change
- Command register **TIO[i]_G[g]_CH[c]_OINST.CMD** = 0x0 will not change
- Due to continue buffer configuration signal *S_INSTR_PULL* = *O_INSTR_PULL_NEXT* will be driven.

On each *S_DATA_PUSH_PREV* = 1 request, the content of the buffer is loaded with new values:

Operand register gets loaded with **TIO[i]_G[g]_CH[c]_OINST.OP** = **TIO[i]_G[g]_CH[c]_SINST.OP**
Command register gets loaded with **TIO[i]_G[g]_CH[c]_OINST.CMD** = **TIO[i]_G[g]_CH[c]_SINST.CMD**
Due to continue buffer configuration signal *O_DATA_PUSH* = *S_DATA_PUSH_PREV* will be driven.

Any continue buffer can only be used exclusively as instruction buffer (using *O_INSTR_PULL_NEXT* = 1 request) or capture buffer (using *S_DATA_PUSH_PREV* =1 requests). Only in case of incorrect usage of continue buffers, both requests could occur. This can lead to an unexpected operation of the application.

27.5.10.12.2 Continue Buffer Behavior With Instruction Freeze

The configuration **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b001; **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1 is illegal!

A valid configuration can be established by using a start buffer with instruction freeze (see chapter *27.5.10.11.2 "Start Buffer Behavior With Instruction Freeze"*).

27.5.10.12.3 PL_EVT Generation

For a continue buffer resource the condition *O_BUFFER_EMPTY* is defined as:
O_BUFFER_EMPTY = 1 for 1 system clock, if internal state *O_BUFF_FILLED* changes from 1 –> 0

Depending on the configured resources in a channel the condition *O_BUFFER_EMPTY* is used as *PL_EVT* [c:c] (details see table *65 "PL_EVT[c:c] Selection Dependent On TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE / TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE"*).

27.5.10.13 Cyclic Instruction Buffer Usage

The cyclic buffer is enabled with the configuration of **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1, the bit fields **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0 and **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = 0 must be cleared.

In this configuration, the S and O resource instruction registers of one channel [c] are connected in a cyclic manner.

Following inputs are taken into account:

- *CYCLIC_INIT_TRIG* [c:c] : init trigger event see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"*
- *PL_TRIG_OUT* [c:c]: exchange trigger event
- *O_INSTR_PULL_NEXT* : exchange trigger from channel [c+1]

Any *O_DATA_PUSH_PREV* = 1 request to store data from the O resource of the channel [c–1] will be ignored. A cyclic buffer can never store data from the O resource of channel [c–1].

If the cyclic buffer is enabled with **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1 the signal *O_INSTR_PULL* will never be set (*O_INSTR_PULL* = 0).

Any init or exchange trigger event (*CYCLIC_INIT_TRIG* [c:c] or *PL_TRIG_OUT* [c:c] or *O_INSTR_PULL_NEXT*)=1 will cancel the actual executed instruction in the S or O resource.

Any exchange trigger event (*PL_TRIG_OUT* [c:c] or *O_INSTR_PULL_NEXT*)=1 exchanges the buffer contents. An init trigger event (*CYCLIC_INIT_TRIG* [c:c]=1) might exchange the buffer contents (more details follow in *27.5.10.13.2 "Init Trigger Behavior"*). But further functions are not performed, *INSTR_PULL* and *DATA_PUSH* will keep 0. Only an active instruction in the O resource can set triggered by an *INSTR_END* =1 in the same clock cycle a *DATA_PUSH* = (**TIO[i]_G[g]_CH[c]_OCMD.DATA_PUSH_EN** and *INSTR_END*). E.g. issue a capture buffer push into the next channel resource.

Note:

It is mandatory not to configure a count instruction at the S resource and a shift or compare or capture instruction at the O resource while cyclic buffer feature is active (**TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** =1) otherwise the behavior of channel [c] is undefined.

Note:

It is mandatory to write the S resource (**TIO[i]_G[g]_CH[c]_SINST** or **TIO[i]_G[g]_CH[c]_SINST.CMD**) and O resource (**TIO[i]_G[g]_CH[c]_OINST** or **TIO[i]_G[g]_CH[c]_OINST.CMD**) registers, even if one of the two buffers should be filled with no active command e.g. zeros, to ensure that the buffer filled flag of both buffers is set. Once the filled flags are set for the two buffers, they will not clear on upcoming init or exchange trigger events regardless of the values stored in the S resource and O resource.

27.5.10.13.1 Exchange Trigger Behavior

An exchange of the content of the S and O resource registers is initiated, if one of the exchange triggers is active (*PL_TRIG_OUT* [c:c] | *O_INSTR_PULL_NEXT*) = 1.

- **TIO[i]_G[g]_CH[c]_SINST** at (t+1) = **TIO[i]_G[g]_CH[c]_OINST** at (t);
- **TIO[i]_G[g]_CH[c]_OINST** at (t+1) = **TIO[i]_G[g]_CH[c]_SINST** at (t);
- In case the S resource operates a count instruction **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** = 0b1–, the instruction initializes by setting the internal register *ITERA_CNT[c]* at (t+1) to **TIO[i]_G[g]_CH[c]_OINST.CMD** [4:3] at (t).
- In case the O resource operates a shift instruction **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b01–, the instruction initializes by setting the internal register **TIO[i]_G[g]_CH[c]_SHIFTCNT** to 0.
- In case the O resource operates a compare instruction **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b10–| 0b110, the instruction initializes by setting the internal register *CMP_EVT_EN* [c:c], *CMP_EVT_PREV_EN* [c:c] according to the active dual compare instruction. In addition, compare events which have occurred during a previous activation will be cleared.

27.5.10.13.2 Init Trigger Behavior

The init trigger behavior (signal *CYCLIC_INIT_TRIG* [c:c] = 1) is enabled by **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN** =1. See table *68 "CYCLIC_INIT_TRIG[c:c] Enable Function "* .

Caution: The bit field **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** has a different functionality if the channel operates as cyclic buffer.

With the bit fields **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** and **TIO[i]_G[g]_CH[c]_OINST.INSTR_PULL_EN** , it is possible to define the initial state of the instruction register contents.

If the S resource operates instructions, the one register of **TIO[i]_G[g]_CH[c]_OINST** , **TIO[i]_G[g]_CH[c]_SINST** which is marked with **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** = 0 defines the content for the initial instruction which is loaded to **TIO[i]_G[g]_CH[c]_SINST** in case a init trigger event *CYCLIC_INIT_TRIG* [c:c] = 1 occurs.

If the O resource operates instructions, the one register of **TIO[i]_G[g]_CH[c]_OINST** , **TIO[i]_G[g]_CH[c]_SINST** which is marked with **TIO[i]_G[g]_CH[c]_OINST.INSTR_PULL_EN** = 1 defines the content for the initial instruction which is loaded to **TIO[i]_G[g]_CH[c]_OINST** in case a init trigger event *CYCLIC_INIT_TRIG* [c:c]= 1 occurs.

If **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN** =1 is not allowed to configure both bit fields **TIO[i]_G[g]_CH[c]_OINST.INSTR_PULL_EN** and **TIO[i]_G[g]_CH[c]_SINST.INSTR_PULL_EN** to 0x0 or 0x1, the behavior of the design is undefined.

On every init trigger event *CYCLIC_INIT_TRIG* [c:c] = 1 and **TIO[i]_G[g]_CH[c]_OINST.INSTR_PULL_EN** = 0 the register exchange is initiated:

- **TIO[i]_G[g]_CH[c]_SINST** at (t+1) = **TIO[i]_G[g]_CH[c]_OINST** at (t);
- **TIO[i]_G[g]_CH[c]_OINST** at (t+1) = **TIO[i]_G[g]_CH[c]_SINST** at (t);

On every init trigger event *CYCLIC_INIT_TRIG* [c:c] = 1:

- In case the S resource operates a count instruction **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** = 0b1–, the instruction initializes by setting the internal register *ITERA_CNT[c]* at (t+1) to **TIO[i]_G[g]_CH[c]_OINST.CMD** [4:3] at (t) if **TIO[i]_G[g]_CH[c]_OINST.INSTR_PULL_EN** at (t) = 0. If **TIO[i]_G[g]_CH[c]_OINST.INSTR_PULL_EN** at (t) = 1, the instruction initializes by setting the internal register *ITER_CNT[c]* at (t+1) to **TIO[i]_G[g]_Ch[c]_SINST.CMD** [4:3] at (t).
- In case the O resource operates a shift instruction **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b01–, the instruction initializes by setting the internal register **TIO[i]_G[g]_CH[c]_SHIFTCNT** to 0.
- In case the O resource operates a compare instruction **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** = 0b10–| 0b110, the instruction initializes by setting the internal register *CMP_EVT_EN* [c:c], *CMP_EVT_PREV_EN* [c:c] according to the active dual compare instruction. In addition, compare events which have occurred during a previous activation will be cleared.

In case of an exchange trigger event *PL_TRIG_OUT* [c:c] or *O_INSTR_PULL_NEXT* occurs in the same clock cycle as an init trigger event *CYCLIC_INIT_TRIG* [c:c] the init trigger behavior defined in this chapter will be applied.

27.5.10.13.3 Cyclic Instruction Buffer Examples

Next table shows examples for using a cyclic instruction buffer.

Note:

The application has to ensure that S resource [c] is not configured to operate a count instruction while the O resource c is configured to operate a shift or compare or capture instruction while the cyclic buffer is active **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 1. The behavior of the channel [c] would be undefined if this configuration will be in use.

Table 79 Examples for Using a Cyclic Buffer in a Channel

Channel c S resource	Channel c O resource	Application example
Start buffer	Shift instruction	Generate a serial bit pattern with length n:2-4..48. First pattern part is located in the O resource, second part is located in the S resource. Once started, this pattern will be repeated in an endless manner.
Start buffer	Compare instruction	Generate a PWM with offset: Use as TB a local *SOP* resource in channel 1 or *SOP* resourcein channel 5 which defines the PWM master period. The O resource defines the offset1 with corresponding signal level; the S resource defines the offset2 with corresponding signal level
Start buffer	Capture instruction	Trigger capture on every S edge, use 2 different capture sources; O resource defines TB1 capture; S resource defines TB0 capture
Count instruction	Continue buffer	Define 2 different count instructions (e.g. accumulate signal level=1, count rising edges) which shall be executed in a cyclic manner; change counting based on trigger event.
Start buffer	Continue buffer	Cyclic buffer of channel [c] defines 2 different start counter values for a count resource in channel [c+1].

27.5.10.14 Operand Usage

The default data width of operands in the PL Processing block is 24-bit.

To allow porting of applications with lesser resolution (12-bit) on the TIO_PL channels, following resources can be configured in a way that the 24-bit data path is separated in two 12-bit data paths.

The S resource counter and the O resource compare support a separated data path. For the register **TIO[i]_G[g]_CH[c]_OOP** exists a multi view with bit fields to separately access the two 12-bit operands.

- Bit field **TIO[i]_G[g]_CH[c]_OOP12.OPA** accesses the lower 12-bit
- Bit field **TIO[i]_G[g]_CH[c]_OOP12.OPB** accesses the higher 12-bit

All following registers support two 12-bit operands:

- **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA** ; **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB**
- **TIO[i]_G[g]_CH[c]_SOP12.OPA** ; **TIO[i]_G[g]_CH[c]_SOP12.OPB**
- **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA** ; **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB**

Depending on the configuration bit field **TIO[i]_G[g]_OP_USAGE.MODE[c]** , it can be selected which slice of the 12-bit data path is in operation.

If **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA** is used for a compare instruction the lower bits **TIO[i]_G[g]_CH[c]_OINST_COMP.OP** [11:0] are compared with the lower bits [11:0] of the TB signal.
If **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB** is used for a compare instruction the upper bits **TIO[i]_G[g]_CH[c]_OINST_COMP.OP** [23:12] are compared with the upper bits [23:12] of the TB signal.

Table 80 OP_USAGE Mode Configurations

TIO[i]_G[g]_OP_USAGE.MODE[c]	Compare function active	Count operation active
0b000	**TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA**	**TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA** and **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB**

TIO[i]_G[g]_OP_USAGE.MODE[c]	Compare function active	Count operation active
0b001	**TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB**	**TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA** and **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB**
0b010	**TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA**	**TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA**
0b011	**TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB**	**TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB**
0b100	**TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA**	**TIO[i]_G[g]_CH[c]_SINST_COUNT.OP**
0b101	**TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB**	**TIO[i]_G[g]_CH[c]_SINST_COUNT.OP**
0b110	**TIO[i]_G[g]_CH[c]_OINST_COMP.OP**	**TIO[i]_G[g]_CH[c]_SINST_COUNT.OP**
0b111	**TIO[i]_G[g]_CH[c]_OINST_COMP.OP**	**TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA** and **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB**

27.5.10.14.1 Compare Data Path

The compare block is split up in 2 separated 12-bit compare slices, both capable of generating the conditions:

- CMPA_EQUAL and CMPA_GREATER
- CMPB_EQUAL and CMPB_GREATER

Depending on the values of **TIO[i]_G[g]_OP_USAGE.MODE[c]** the compare results CMP_EQ and CMP_GEQ are generated.

Table 81 Compare Slice Function

TIO[i]_G[g]_OP_USAGE.MODE[c]	Compare function	CMP_EQ	CMP_GEQ
0b0-0 0b100	**TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA**	CMPA_EQUAL	CMPA_EQUAL or CMPA_GREATER
0b0-1 0b101	**TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB**	CMPB_EQUAL	CMPB_EQUAL or CMPB_GREATER
0b11-	**TIO[i]_G[g]_CH[c]_OINST_COMP.OP**	CMPA_EQUAL and CMPB_EQUAL	(CMPB_EQUAL and CMPA_EQUAL) or CMPB_GREATER or (CMPB_EQUAL and CMPA_GREATER)

27.5.10.14.2 Counter Data Path

The counter block is split up in 2 separated 12-bit counter slices, both capable to increment or decrement:

- Operation **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA** + STEPA / **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA** - STEPA generates the result COUNTA with CARRYA
- Operation **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB** + STEPB / **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB** - STEPB generates the result COUNTB

Depending on the values of **TIO[i]_G[g]_OP_USAGE.MODE[c]** the counter results COUNTA / COUNTB are generated.

Table 82 Counter Slice Function

TIO[i]_G[g]_OP_USAGE.MODE[c]	Count function	t: STEP	t+1: COUNTB	t+1: COUNTA
0b---	-	0	COUNTB	COUNTA
0b00- 0b111	inc	1	COUNTB + STEP	COUNTA + STEP
0b00- 0b111	dec	1	COUNTB - STEP	COUNTA - STEP
0b010	inc	1	COUNTB	COUNTA + STEP
0b010	dec	1	COUNTB	COUNTA - STEP
0b011	inc	1	COUNTB + STEP	COUNTA
0b011	dec	1	COUNTB - STEP	COUNTA

TIO[i]_G[g]_OP_USAGE.MODE[c]	Count function	t: STEP	t+1: COUNTB	t+1: COUNTA
0b10–0b110	inc	1	COUNTB + CARRYA	COUNTA + STEP
0b10–0b110	dec	1	COUNTB – CARRYA	COUNTA – STEP

The register **TIO[i]_G[g]_OP_USAGE** hosts the bit field **TIO[i]_G[g]_OP_USAGE.MODE[c]** for all 8 channels of the channel group. Additionally, the bit fields **TIO[i]_G[g]_OP_USAGE.WRITE_EN[c]** (c ∈ {0, 1,..., 7}) exist, which implement a write protection for the bit field **TIO[i]_G[g]_OP_USAGE.MODE[c]** .

A write operation to this register will only alter the state of the corresponding bit fields **TIO[i]_G[g]_OP_USAGE.MODE[c]** if the **TIO[i]_G[g]_OP_USAGE.WRITE_EN[c]** bit is 1. Otherwise the bit field **TIO[i]_G[g]_OP_USAGE.MODE[c]** will not change.

This allows atomic reconfiguration of a single channel without influencing others. Furthermore, atomic synchronous reconfiguration of multiple channels in a channel group is possible.

27.5.11 TIO_PL Software Reset of Channels

Each channel x is assigned a unique bit x in the configuration register **TIO[i]_PL_SWRST** for software reset of the TIO_PL Processing block. Index g, c are defined as x = g*8 +c.

A software reset on signal *TIO[j]_G[g]_RESET_CH[c]* can only be initiated if the channel is disabled **TIO[i]_ENDIS.CH[x]** = 0.

Writing via the configuration interface the value 0b1 to the bit field **TIO[i]_PL_SWRST.CH[x]** while **TIO[i]_ENDIS.CH[x]** = 0, will immediately reset the content of the following registers to the initial hardware reset state.

- **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_ODIS**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_SEL_IN**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_O_TRIG_OUT_EN**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_S_TRIG_OUT_EN**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_INIT_TRIG_EN**
- **TIO[i]_G[g]_CH[c]_CTRL2**
- **TIO[i]_G[g]_CH[c]_SINST**
- **TIO[i]_G[g]_CH[c]_SCMD**
- **TIO[i]_G[g]_CH[c]_SOP**
- **TIO[i]_G[g]_CH[c]_OINST**
- **TIO[i]_G[g]_CH[c]_OCMD**
- **TIO[i]_G[g]_CH[c]_OOP**
- **TIO[i]_G[g]_CH[c]_SHIFTCNT**
- Internal signals inside the TIO_PL processing block of channel c: *S_BUFF_FILLED* , *O_BUFF_FILLED* , *CMP_EVT_EN* [c:c], *CMP_EVT_PREV_EN* [c:c], *DUALCMP_END* [c:c], *DUALCMP_END_NEXT* [c:c], *FSM_DUALCMP_STATE* [c:c], *ITERA_CNT[c]* , *SHIFT_CNT*

Note:

The following fields which control the *PL_TRIG_OUT* [c:c] functionality will not be altered by a software reset. This ensures that the *PL_TRIG_OUT* [c:c] generation can be used independent of the TIO_PL processing functionality.

- **TIO[i]_PLTRIG_OUT_GATE_EN.CH[x]**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_EN_S_RE**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_EN_S_FE**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_EN_O_RE**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_EN_O_FE**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_EN_PREV_TRIG**

- **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_EN_PL_EVT**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_EN_PREV_PL_TRIG**
- **TIO[i]_G[g]_CH[c]_CTRL.PL_TRIG_OUT_UPD_EN**

Atomic reset of multiple channels is possible by writing a 1 on each associated bit. E.g: Writing 0x0F to register **TIO[i]_PL_SWRST** will reset the TIO_PL processing block of channel 0, channel 1, channel 2 and channel 3.

27.6 Application Information

The TIO module can be used for multiple applications. The list in the following is a starting point. They are not yet worked out in detail. This will follow on request.

27.6.1 TIO Applications

Following examples, that can be used with a MCS–channel or CPU are planned:

- Single channel pulse measurement
- Single channel PWM measurement
- Single channel periodic sampling (serial shift input)
- Single channel pulse generation
- Single channel PWM generation
- Single channel serial shift output
- N channel pulse measurement
- N channel PWM measurement
- N channel periodic sampling (serial shift input)
- N channel synchronous pulse generation
- N channel synchronous PWM generation
- N channel synchronous serial shift output
- AB decoder

27.6.2 TIO_PL Applications

The same applications as listed in the previous chapter can be supported with the TIO_PL extension. They will operate with higher performance e.g.:

- Lower resolution of minimum pulse generation / measurement
- Lower resolution of minimum PWM generation / measurement
- Higher shift in / shift out frequency
- Lesser interaction of application software with TIO hardware
- No / reduced impact of variation of CPU / MCS access latency on measurement accuracy / signal generation accuracy

In addition, other applications can be supported:

- Output signal generation on single compare match
- Output signal generation on dual compare match
- Instruction buffer usage length = k
- Capture buffer usage length = k
- Instruction and capture buffer usage length = k

27.6.2.1 Single Channel Pulse Measurement

Task: Use channel group g=0: Measure high pulse length of **TIO[i]_S.CH[x]** with *PL_UPDATE* [c:c] resolution; trigger IRQ when pulse measurement finished.

Solution:

- Use counter mode: **TIO[i]_G[0]_CH[c]_CTRL.PL_S_MODE** = 10
- Count on high level *S_OUT* [c:c] : **TIO[i]_G[0]_CH[c]_SCMD_COUNT.CNT_COND** = 0b10
- Reset counter on rising edge *S_OUT* [c:c]: **TIO[i]_G[0]_CH[c]_CTRL.TRIG_OUT_EN_S_RE** = 1
- Enable IRQ on falling edge *S_OUT* [c:c]: **TIO[i]_G[0]_CH[c]_IRQ_EN.S_FE_IRQ_EN** = 1
- Triggered by IRQ pulse high length can be read with **TIO[i]_G[0]_CH[c]_SOP**

27.6.2.2 Single Channel Pulse Generation

Task: Use channel group g=0: Generate endless trigger pulse on *PL_UPDATE* [c:c] resolution when global *RS_TB1[0]* = 0x50; no impact on **TIO[i]_O.CH[x]**

Solution:

- Use equal compare instruction: **TIO[i]_G[0]_CH[c]_CTRL.PL_O_MODE** = 0b110
- No impact on **TIO[i]_O.CH[x]** : **TIO[i]_G[0]_CH[c]_CTRL.PL_ODIS** = 1
- Repeat compare endless: **TIO[i]_G[0]_CH[c]_CTRL.PL_FREEZE_O_EN** = 1
- Set compare value: **TIO[i]_G[0]_CH[c]_OOP** = 0x50
- Select compare source *RS_TB1[0]* : **TIO[i]_G[0]_CH[c]_OCMD_COMP.SEL_TB** = 0b01
- Activate compare: **TIO[i]_G[0]_CH[c]_OCMD_COMP.CMD_ACTIVE_CC** = 0b10

27.7 TIO Configuration Registers Description

Definition: alias register
An alias register defines an additional address which points to the same register. Typically different behavior is implemented when alias register are accessed.
In the following chapters **TIO[i]_S** – **TIO[i]_ICYCLIC_MODE** , **TIO[i]_TRIG_OUT_GATE_EN** – **TIO[i]_SPLTRIG_OUT_GATE_EN** the behavior of alias registers is described dependent on the name prefix:
TIO[i]_xxx : defines the alias register for writing the value of the register xxx
TIO[i]_Cxxx: defines the alias register for clearing individual bits of the register xxx
TIO[i]_Sxxx: defines the alias register for setting individual bits of the register xxx
TIO[i]_Ixxx: defines the alias register for inverting individual bits of the register xxx

Definition: multiview register
A multiview register defines an alternative representation of a register. Multiview registers for a particular register share its address. The content of the register will be interpreted differently due to different bit field definitions. On a software layer this correlates to the usage of unions.
Following register names are used to define multiviews:

TIO[i]_G[g]_CH[c]_SINST_COUNT multiview for **TIO[i]_G[g]_CH[c]_SINST**

TIO[i]_G[g]_CH[c]_SINST_COUNT12 multiview for **TIO[i]_G[g]_CH[c]_SINST**

TIO[i]_G[g]_CH[c]_OINST_SHIFT multiview for **TIO[i]_G[g]_CH[c]_OINST**

TIO[i]_G[g]_CH[c]_OINST_COMP multiview for **TIO[i]_G[g]_CH[c]_OINST**

TIO[i]_G[g]_CH[c]_OINST_COMP12 multiview for **TIO[i]_G[g]_CH[c]_OINST**

TIO[i]_G[g]_CH[c]_OINST_CAP multiview for **TIO[i]_G[g]_CH[c]_OINST**

TIO[i]_G[g]_CH[c]_OCAPTURE multiview for **TIO[i]_G[g]_CH[c]_OINST**

TIO[i]_G[g]_CH[c]_OCAPTURE12 multiview for **TIO[i]_G[g]_CH[c]_OINST**

TIO[i]_G[g]_CH[c]_SCMD_COUNT multiview for **TIO[i]_G[g]_CH[c]_SCMD**

TIO[i]_G[g]_CH[c]_OCMD_SHIFT multiview for **TIO[i]_G[g]_CH[c]_OCMD**

TIO[i]_G[g]_CH[c]_OCMD_COMP multiview for **TIO[i]_G[g]_CH[c]_OCMD**

TIO[i]_G[g]_CH[c]_OCMD_CAP multiview for **TIO[i]_G[g]_CH[c]_OCMD**

TIO[i]_G[g]_CH[c]_SOP12 multiview for **TIO[i]_G[g]_CH[c]_SOP**

TIO[i]_G[g]_CH[c]_OOP12 multiview for **TIO[i]_G[g]_CH[c]_OOP**

27.7.1 TIO[i]_S

Description	TIO[i] signal sampling register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_S
Address	$0x20000 * i + 0x3C00$
C-Name	GTM.CLS[i].TIO.S

Interface: MCS[i]

Name	TIO[i]_S
Address	$0x3C00$
C-Name	

CH[x]	
Description	Value of channel [x]
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	TIO_OUT_RST
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Signal level low 1 : Signal level high

27.7.2 TIO[i]_CS

Description	TIO[i] clear signal sampling register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CS
Address	$0x20000 * i + 0x3C40$
C-Name	GTM.CLS[i].TIO.CS

Interface: MCS[i]

Name	TIO[i]_CS
Address	$0x3C40$
C-Name	

CH[x]	
Description	Clear channel [x]
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	TIO_OUT_RST
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Signal level low 1 : Signal level high
W-Coding	0 : No action 1 : Set signal level low

27.7.3 TIO[i]_SS

Description	TIO[i] set signal sampling register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_SS
Address	$0x20000 * i + 0x3C80$
C-Name	GTM.CLS[i].TIO.SS

Interface: MCS[i]

Name	TIO[i]_SS

Address	$0x3C80$
C-Name	

CH[x]	
Description	Set channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	-
Initial value	TIO_OUT_RST
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Signal level low 1 : Signal level high
W-Coding	0 : No action 1 : Set signal level high

27.7.4 TIO[i]_IS

Description	TIO[i] invert signal sampling register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_IS
Address	$0x20000 * i + 0x3CC0$
C-Name	GTM.CLS[i].TIO.IS

Interface: MCS[i]

Name	TIO[i]_IS
Address	$0x3CC0$
C-Name	

CH[x]	
Description	Invert channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$

CH[x]	
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToToggle
Condition	–
Initial value	TIO_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Signal level low 1 : Signal level high
W-Coding	0 : No action 1 : Invert signal level

27.7.5 TIO[i]_O

Description	TIO[i] output register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_O
Address	$0x20000 * i + 0x3C04$
C-Name	GTM.CLS[i].TIO.O

Interface: MCS[i]

Name	TIO[i]_O
Address	$0x3C04$
C-Name	

CH[x]	
Description	Value driven on output of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

CH[x]	
Initial value	TIO_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Signal level low 1 : Signal level high

27.7.6 TIO[i]_CO

Description	TIO[i] clear output register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CO
Address	$0x20000 * i + 0x3C44$
C-Name	GTM.CLS[i].TIO.CO

Interface: MCS[i]

Name	TIO[i]_CO
Address	$0x3C44$
C-Name	

CH[x]	
Description	Clear channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	TIO_OUT_RST
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Signal level low 1 : Signal level high
W-Coding	0 : No action 1 : Set signal level low

27.7.7 TIO[i]_SO

Description	TIO[i] set output register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_SO
Address	$0x20000 * i + 0x3C84$
C-Name	GTM.CLS[i].TIO.SO

Interface: MCS[i]

Name	TIO[i]_SO
Address	$0x3C84$
C-Name	

CH[x]	
Description	Set channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	-
Initial value	TIO_OUT_RST
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Signal level low 1 : Signal level high
W-Coding	0 : No action 1 : Set signal level high

27.7.8 TIO[i]_IO

Description	TIO[i] invert output register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER

Clock Active Cond	TIO[i]_CLK_ENABLE == 1

Interface: CPU

Name	TIO[i]_IO
Address	$0x20000 * i + 0x3CC4$
C-Name	GTM.CLS[i].TIO.IO

Interface: MCS[i]

Name	TIO[i]_IO
Address	$0x3CC4$
C-Name	

CH[x]	
Description	Invert channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToToggle
Condition	-
Initial value	TIO_OUT_RST
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Signal level low 1 : Signal level high
W-Coding	0 : No action 1 : Invert signal level

27.7.9 TIO[i]_ENDIS

Description	TIO[i] enable/disable register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	CTIO[i] == 1
Storage Type	REGISTER
Clock Active Cond	TIO[i]_CLK_ENABLE == 1

Interface: CPU

Name	TIO[i]_ENDIS
Address	$0x20000 * i + 0x3C08$

C–Name	GTM.CLS[i].TIO.ENDIS

Interface: MCS[i]

Name	TIO[i]_ENDIS
Address	$0x3C08$
C–Name	

CH[x]	
Description	Enable/Disable request of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Channel [x] is disabled 1 : Channel [x] is enabled

27.7.10 TIO[i]_CENDIS

Description	TIO[i] disable register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CENDIS
Address	$0x20000 * i + 0x3C48$
C–Name	GTM.CLS[i].TIO.CENDIS

Interface: MCS[i]

Name	TIO[i]_CENDIS
Address	$0x3C48$
C–Name	

CH[x]	
Description	Disable request of channel [x].

CH[x]	
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : Channel [x] is disabled 1 : Channel [x] is enabled
W–Coding	0 : No action 1 : Request for disabling channel [x]

27.7.11 TIO[i]_SENDIS

Description	TIO[i] enable register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_SENDIS
Address	$0x20000 * i + 0x3C88$
C–Name	GTM.CLS[i].TIO.SENDIS

Interface: MCS[i]

Name	TIO[i]_SENDIS
Address	$0x3C88$
C–Name	

CH[x]	
Description	Enable request of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False

CH[x]	
ModifiedWriteValue	oneToSet
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : Channel [x] is disabled 1 : Channel [x] is enabled
W–Coding	0 : No action 1 : Request for enabling channel [x]

27.7.12 TIO[i]_IENDIS

Description	TIO[i] toggle enable/disable register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_IENDIS
Address	$0x20000 * i + 0x3CC8$
C–Name	GTM.CLS[i].TIO.IENDIS

Interface: MCS[i]

Name	TIO[i]_IENDIS
Address	$0x3CC8$
C–Name	

CH[x]	
Description	Toggle state request of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToToggle
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

CH[x]	
R-Coding	0 : Channel [x] is disabled 1 : Channel [x] is enabled
W-Coding	0 : No action 1 : Request for toggling state of channel [x] (enabled -> disabled; disabled -> enabled)

27.7.13 TIO[i]_INVERT

Description	TIO[i] signal invert register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_INVERT
Address	$0x20000 * i + 0x3C0C$
C-Name	GTM.CLS[i].TIO.INVERT

Interface: MCS[i]

Name	TIO[i]_INVERT
Address	$0x3C0C$
C-Name	

CH[x]	
Description	Enable/Disable signal inversion of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Channel [x] inversion inactive 1 : Channel [x] inversion active

27.7.14 TIO[i]_CINVERT

Description	TIO[i] clear signal invert register

Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CINVERT
Address	$0x20000 * i + 0x3C4C$
C-Name	GTM.CLS[i].TIO.CINVERT

Interface: MCS[i]

Name	TIO[i]_CINVERT
Address	$0x3C4C$
C-Name	

CH[x]	
Description	Disable signal inversion of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Channel [x] inversion inactive 1 : Channel [x] inversion active
W-Coding	0 : No action 1 : Request for disabling signal inversion on channel [x]

27.7.15 TIO[i]_SINVERT

Description	TIO[i] set signal invert register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_SINVERT
Address	$0x20000 * i + 0x3C8C$
C-Name	GTM.CLS[i].TIO.SINVERT

Interface: MCS[i]

Name	TIO[i]_SINVERT
Address	$0x3C8C$
C-Name	

CH[x]	
Description	Enable signal inversion of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Channel [x] inversion inactive 1 : Channel [x] inversion active
W-Coding	0 : No action 1 : Request for enabling signal inversion on channel [x]

27.7.16 TIO[i]_IINVERT

Description	TIO[i] toggle signal invert register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_IINVERT
Address	$0x20000 * i + 0x3CCC$
C-Name	GTM.CLS[i].TIO.IINVERT

Interface: MCS[i]

Name	TIO[i]_IINVERT

Address	$0x3CCC$
C-Name	

CH[x]	
Description	Invert signal inversion of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToToggle
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Channel [x] inversion inactive 1 : Channel [x] inversion active
W-Coding	0 : No action 1 : Request for toggle of signal inversion on channel [x] (active -> inactive; inactive -> active)

27.7.17 TIO[i]_INPUT_MODE

Description	TIO[i] input mode register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_INPUT_MODE
Address	$0x20000 * i + 0x3C10$
C-Name	GTM.CLS[i].TIO.INPUT_MODE

Interface: MCS[i]

Name	TIO[i]_INPUT_MODE
Address	$0x3C10$
C-Name	

CH[x]	
Description	Enable/Disable input mode of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$

CH[x]	
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Channel [x] input mode is disabled 1 : Channel [x] input mode is enabled

27.7.18 TIO[i]_CINPUT_MODE

Description	TIO[i] disable input mode register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CINPUT_MODE
Address	$0x20000 * i + 0x3C50$
C-Name	GTM.CLS[i].TIO.CINPUT_MODE

Interface: MCS[i]

Name	TIO[i]_CINPUT_MODE
Address	$0x3C50$
C-Name	

CH[x]	
Description	Disable input mode of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0

CH[x]	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Channel [x] input mode is disabled 1 : Channel [x] input mode is enabled
W-Coding	0 : No action 1 : Request for disabling input mode on channel [x]

27.7.19 TIO[i]_SINPUT_MODE

Description	TIO[i] enable input mode register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIO}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_SINPUT_MODE
Address	$0x20000 * i + 0x3C90$
C-Name	GTM.CLS[i].TIO.SINPUT_MODE

Interface: MCS[i]

Name	TIO[i]_SINPUT_MODE
Address	$0x3C90$
C-Name	

CH[x]	
Description	Enable input mode of channel [x].
Loop	$x = \{n : 0 <= n <= \mathrm{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Channel [x] input mode is disabled 1 : Channel [x] input mode is enabled

CH[x]	
W-Coding	0 : No action 1 : Request for enabling input mode on channel [x]

27.7.20 TIO[i]_IINPUT_MODE

Description	TIO[i] enable input mode register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_IINPUT_MODE
Address	$0x20000 * i + 0x3CD0$
C-Name	GTM.CLS[i].TIO.IINPUT_MODE

Interface: MCS[i]

Name	TIO[i]_IINPUT_MODE
Address	$0x3CD0$
C-Name	

CH[x]	
Description	Toggle input mode of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToToggle
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Channel [x] is disabled 1 : Channel [x] is enabled
W-Coding	0 : No action 1 : Request for toggling input mode of channel [x] (enabled -> disabled; disabled -> enabled)

27.7.21 TIO[i]_CYCLIC_MODE

Description	TIO[i] cyclic mode register

Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CYCLIC_MODE
Address	$0x20000 * i + 0x3C14$
C-Name	GTM.CLS[i].TIO.CYCLIC_MODE

Interface: MCS[i]

Name	TIO[i]_CYCLIC_MODE
Address	$0x3C14$
C-Name	

CH[x]	
Description	Enable/Disable cyclic mode of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Channel [x] cyclic mode is disabled 1 : Channel [x] cyclic mode is enabled

27.7.22 TIO[i]_CCYCLIC_MODE

Description	TIO[i] disable cyclic mode register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CCYCLIC_MODE

Address	$0x20000 * i + 0x3C54$
C-Name	GTM.CLS[i].TIO.CCYCLIC_MODE

Interface: MCS[i]

Name	TIO[i]_CCYCLIC_MODE
Address	$0x3C54$
C-Name	

CH[x]	
Description	Disable cyclic mode of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : Channel [x] cyclic mode is disabled 1 : Channel [x] cyclic mode is enabled
W-Coding	0 : No action 1 : Request for disabling cyclic mode on channel [x]

27.7.23 TIO[i]_SCYCLIC_MODE

Description	TIO[i] enable cyclic mode register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_SCYCLIC_MODE
Address	$0x20000 * i + 0x3C94$
C-Name	GTM.CLS[i].TIO.SCYCLIC_MODE

Interface: MCS[i]

Name	TIO[i]_SCYCLIC_MODE
Address	$0x3C94$

C–Name	

CH[x]	
Description	Enable cyclic mode of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : Channel [x] cyclic mode is disabled 1 : Channel [x] cyclic mode is enabled
W–Coding	0 : No action 1 : Request for enabling cyclic mode on channel [x]

27.7.24 TIO[i]_ICYCLIC_MODE

Description	TIO[i] enable cyclic mode register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_ICYCLIC_MODE
Address	$0x20000 * i + 0x3CD4$
C–Name	GTM.CLS[i].TIO.ICYCLIC_MODE

Interface: MCS[i]

Name	TIO[i]_ICYCLIC_MODE
Address	$0x3CD4$
C–Name	

CH[x]	
Description	Toggle cyclic mode of channel [x].
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$

CH[x]	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToToggle
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : Channel [x] is disabled 1 : Channel [x] is enabled
W–Coding	0 : No action 1 : Request for toggling cyclic mode of channel [x] (enabled –> disabled; disabled –> enabled)

27.7.25 TIO[i]_FUPD

Description	TIO[i] force update register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIO}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_FUPD
Address	$0x20000 * i + 0x3D00$
C–Name	GTM.CLS[i].TIO.FUPD

Interface: MCS[i]

Name	TIO[i]_FUPD
Address	$0x3D00$
C–Name	

CH[x]	
Description	issue immediately a signal pulse on the update signal of channel [x]
Loop	$x = \{n : 0 <= n <= \mathrm{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0

CH[x]	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No action
W-Coding	0 : No action 1 : Issue single update pulse
	Note: This bit is cleared automatically after write.

27.7.26 TIO[i]_HW_CONF

Description	TIO[i] configuration register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIO}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_HW_CONF
Address	$0x20000 * i + 0x3D04$
C-Name	GTM.CLS[i].TIO.HW_CONF

Interface: MCS[i]

Name	TIO[i]_HW_CONF
Address	$0x3D04$
C-Name	

NTIO_CH8	
Description	signals availability of number of channels
Loop	–
Bit Range	$[1 : 0]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	$\mathrm{NTIO\_CH8}$
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

NTIO_CH8	
R-Coding	0b01 : 8 channels available 0b10 : 16 channels available 0b11 : 24 channels available

TIO_PLUS	
Description	signals availability of TIOplus functionality
Loop	-
Bit Range	$[4:4]$
Access Type	R
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	TIO_PLUS
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : TIOplus functionality not implemented 1 : TIOplus functionality available

27.7.27 TIO[i]_RSEL_CTRL1

Description	TIO[i] resource selection control register 1
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_RSEL_CTRL1
Address	$0x20000 * i + 0x3D08$
C-Name	GTM.CLS[i].TIO.RSEL_CTRL1

Interface: MCS[i]

Name	TIO[i]_RSEL_CTRL1
Address	$0x3D08$
C-Name	

SEL_CLKEN6_[g]	
Description	select source of RS_CLKEN[g][6:6] for channels [g]*8 .. [g]*8+7
Loop	$g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}$
Bit Range	$[g + 24 : g + 24]$

SEL_CLKEN6_[g]	
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Use *CCM[i]_CLK_RES* [6:6] as *RS_CLKEN[g]* [6:6] 1 : Use local *TRIG_OUT[g*8+6]* signal as *RS_CLKEN[g]* [6:6]

SEL_CLKEN7_[g]	
Description	select source of RS_CLKEN[g][7:7] for channels [g]*8 .. [g]*8+7
Loop	$g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}$
Bit Range	$[g + 28 : g + 28]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Use *CCM[i]_CLK_RES* [7:7] resolution as *RS_CLKEN[g]* [7:7] 1 : Use local *TRIG_OUT[g*8+7]* signal as *RS_CLKEN[g]* [7:7]

27.7.28 TIO[i]_RSEL_CTRL2

Description	TIO[i] resource selection control register 2
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_RSEL_CTRL2
Address	$0x20000 * i + 0x3D0C$
C–Name	GTM.CLS[i].TIO.RSEL_CTRL2

Interface: MCS[i]

Name	TIO[i]_RSEL_CTRL2

Address	$0x3D0C$
C-Name	

SEL_TB1_[g]	
Description	select source of RS_TB1[g] for channels [g]*8 .. [g]*8+7
Loop	$g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}$
Bit Range	$[g + 4 : g + 4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Use global *CCM[i]_TBU_TS1* as *RS_TB1[g]* 1 : Use local **TIO[i]_G[g]_CH[1]_SINST.OP** register as *RS_TB1[g]*

SEL_TB2_[g]	
Description	select source of RS_TB2[g] for channels [g]*8 .. [g]*8+7
Loop	$g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}$
Bit Range	$[g + 8 : g + 8]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Use global *CCM[i]_TBU_TS2* as *RS_TB2[g]* 1 : Use local **TIO[i]_G[g]_CH[5]_SINST.OP** register as *RS_TB2[g]*

27.7.29 TIO[i]_PL_SWRST

Description	TIO[i] software reset for TIO Plus functionality
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER

Clock Active Cond	$\mathrm{TIO}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_PL_SWRST
Address	$0x20000 * i + 0x3D10$
C-Name	GTM.CLS[i].TIO.PL_SWRST

Interface: MCS[i]

Name	TIO[i]_PL_SWRST
Address	$0x3D10$
C-Name	

CH[x]	
Description	reset TIO_Plus resources of channel [x]
Loop	$x = \{n : 0 <= n <= \mathrm{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	-
Initial value	0
Protect Enable Cond	TIO[i]_CH_EN[x] == 1
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No action
W-Coding	0 : No action 1 : Reset registers of this channel (details see chapter *27.5.11 "TIO_PL Software Reset of Channels"*)
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected, it can only be written if channel [x] is disabled by **TIO[i]_ENDIS.CH[x]** = 0.

27.7.30 TIO[i]_TRIG_OUT_GATE_EN

Description	TIO[i] enable Trigger Output, output gating register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIO}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_TRIG_OUT_GATE_EN
Address	$0x20000 * i + 0x3C18$
C-Name	GTM.CLS[i].TIO.TRIG_OUT_GATE_EN

Interface: MCS[i]

Name	TIO[i]_TRIG_OUT_GATE_EN
Address	$0x3C18$
C-Name	

CH[x]	
Description	enable gating of Trigger Output events of channel [x]
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : No gating of Trigger Output events active 1 : In case **TIO[i]_O.CH[x]** = 0 no Trigger Output event occurs

27.7.31 TIO[i]_CTRIG_OUT_GATE_EN

Description	TIO[i] clear Trigger Output, output gating register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CTRIG_OUT_GATE_EN
Address	$0x20000 * i + 0x3C58$
C-Name	GTM.CLS[i].TIO.CTRIG_OUT_GATE_EN

Interface: MCS[i]

Name	TIO[i]_CTRIG_OUT_GATE_EN
Address	$0x3C58$

C–Name	

CH[x]	
Description	disable gating of Trigger Output events of channel [x]
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No gating of Trigger Output events active 1 : In case **TIO[i]_O.CH[x]** = 0 no Trigger Output event occurs
W–Coding	0 : No action 1 : Channel [x] use no gating of Trigger Output events

27.7.32 TIO[i]_STRIG_OUT_GATE_EN

Description	TIO[i] set Trigger Output, output gating register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\_\text{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_STRIG_OUT_GATE_EN
Address	$0x20000 * i + 0x3C98$
C–Name	GTM.CLS[i].TIO.STRIG_OUT_GATE_EN

Interface: MCS[i]

Name	TIO[i]_STRIG_OUT_GATE_EN
Address	$0x3C98$
C–Name	

CH[x]	
Description	enable gating of Trigger Output events of channel [x]
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$

CH[x]	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No gating of Trigger Output events active 1 : In case **TIO[i]_O.CH[x]** = 0 no Trigger Output event occurs
W–Coding	0 : No action 1 : Channel [x] use **TIO[i]_O.CH[x]** gating of Trigger Output event

27.7.33 TIO[i]_PLTRIG_OUT_GATE_EN

Description	TIO[i] enable PL_TRIG_OUT output gating register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIO}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_PLTRIG_OUT_GATE_EN
Address	$0x20000 * i + 0x3C1C$
C–Name	GTM.CLS[i].TIO.PLTRIG_OUT_GATE_EN

Interface: MCS[i]

Name	TIO[i]_PLTRIG_OUT_GATE_EN
Address	$0x3C1C$
C–Name	

CH[x]	
Description	enable gating of PL_TRIG_OUT events of channel [x]
Loop	$x = \{n : 0 <= n <= \mathrm{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0

CH[x]	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : No gating of *PL_TRIG_OUT* [c:c] events active 1 : In case Signal **TIO[i]_O.CH[x]** = 0 no PL_TRIG_OUT[c:c] event occurs

27.7.34 TIO[i]_CPLTRIG_OUT_GATE_EN

Description	TIO[i] clear PL_TRIG_OUT output gating register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIO}[i]\_\mathrm{CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_CPLTRIG_OUT_GATE_EN
Address	$0x20000 * i + 0x3C5C$
C-Name	GTM.CLS[i].TIO.CPLTRIG_OUT_GATE_EN

Interface: MCS[i]

Name	TIO[i]_CPLTRIG_OUT_GATE_EN
Address	$0x3C5C$
C-Name	

CH[x]	
Description	disable gating of PL_TRIG_OUT events of channel [x]
Loop	$x = \{n : 0 <= n <= \mathrm{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No gating of *PL_TRIG_OUT* [c:c] events active 1 : In case **TIO[i]_O.CH[x]** = 0 no *PL_TRIG_OUT* [c:c] event occurs
W-Coding	0 : No action 1 : Channel [x] use no gating of *PL_TRIG_OUT* events

27.7.35 TIO[i]_SPLTRIG_OUT_GATE_EN

Description	TIO[i] set PL_TRIG_OUT output gating register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_SPLTRIG_OUT_GATE_EN
Address	$0x20000 * i + 0x3C9C$
C-Name	GTM.CLS[i].TIO.SPLTRIG_OUT_GATE_EN

Interface: MCS[i]

Name	TIO[i]_SPLTRIG_OUT_GATE_EN
Address	$0x3C9C$
C-Name	

CH[x]	
Description	enable gating of PL_TRIG_OUT events of channel [x]
Loop	$x = \{n : 0 <= n <= \text{NTIO\_CH8} * 8 - 1\}$
Bit Range	$[x : x]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToSet
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No gating of *PL_TRIG_OUT* [c:c] events active 1 : In case **TIO[i]_O.CH[x]** = 0 no *PL_TRIG_OUT* [c:c] event occurs
W-Coding	0 : No action 1 : Channel [x] use **TIO[i]_O.CH[x]** gating of *PL_TRIG_OUT* [c:c] events

27.7.36 TIO[i]_G[g]_CH[c]_CTRL

Description	TIO[i] group [g] channel [c] control register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER

Clock Active Cond	TIO[i]_CLK_ENABLE == 1

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_CTRL
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3000$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].CTRL

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_CTRL
Address	$0x400 * g + 0x40 * c + 0x3000$
C-Name	

TRIG_OUT_EN_S_RE	
Description	TIO[i] channel [c] Trigger Output source: Rising edge S[c] enable
Loop	-
Bit Range	[0 : 0]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

TRIG_OUT_EN_S_FE	
Description	TIO[i] channel [c] Trigger Output source: Falling edge S[c] enable
Loop	-
Bit Range	[1 : 1]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

TRIG_OUT_EN_O_RE	
Description	TIO[i] channel [c] Trigger Output source: Rising edge O[c] enable
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

TRIG_OUT_EN_O_FE	
Description	TIO[i] channel [c] Trigger Output source: Falling edge O[c] enable
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

TRIG_OUT_EN_PREV_TRIG	
Description	TIO[i] channel [c] Trigger Output source: TRIG_OUT[c]-1enable
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

TRIG_OUT_EN_PREV_TRIG	
RW-Coding	0 : Source disabled 1 : Source enabled

TRIG_OUT_EN_PL_EVT	
Description	TIO[i] channel [c] Trigger Output source: PL_EVT[c] enable
Loop	-
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

TRIG_OUT_EN_PREV_PL_TRIG	
Description	TIO[i] channel [c] Trigger Output source: PL_TRIG_OUT[c]-1enable
Loop	-
Bit Range	[6 : 6]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

PL_CYCLIC_INIT_TRIG_EN	
Description	Enable usage of TRIG_OUT[c] as init trigger of cyclic buffer functionality in channel [c].
Loop	-
Bit Range	[7 : 7]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

PL_CYCLIC_INIT_TRIG_EN	
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : *TRIG_OUT* [c:c] signal disabled 1 : *TRIG_OUT* [c:c] signal enabled
	Details for selection of mode of operations see *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"* .

UPDATE_SRC	
Description	Select update source of channel [c].
Loop	–
Bit Range	[11 : 8]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	14
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	TIO_PLUS == 0 && c == 0 && g == 0 0 : *RS_CLKEN[g]* [0:0] selected 1 : *RS_CLKEN[g]* [1:1] selected 2 : *RS_CLKEN[g]* [2:2] selected 3 : *RS_CLKEN[g]* [3:3] selected 4 : *RS_CLKEN[g]* [4:4] selected 5 : *RS_CLKEN[g]* [5:5] selected 6 : *RS_CLKEN[g]* [6:6] selected 7 : *RS_CLKEN[g]* [7:7] selected 8 : *TRIG_OUT* [c:c] selected 9 : Input tied to '0'; no update will occur 14 : 0; no update will occur 15 : 1; update on GTM system clock selected
RW-Coding	TIO_PLUS == 0 && !(c == 0 && g == 0) 0 : *RS_CLKEN[g]* [0:0] selected 1 : *RS_CLKEN[g]* [1:1] selected 2 : *RS_CLKEN[g]* [2:2] selected 3 : *RS_CLKEN[g]* [3:3] selected 4 : *RS_CLKEN[g]* [4:4] selected 5 : *RS_CLKEN[g]* [5:5] selected 6 : *RS_CLKEN[g]* [6:6] selected 7 : *RS_CLKEN[g]* [7:7] selected ▽

UPDATE_SRC	
	△ 8 : *TRIG_OUT* [c:c] selected 9 : *TRIG_OUT* [c−1:c−1] selected 14 : 0; no update will occur 15 : 1; update on GTM system clock selected
RW−Coding	TIO_PLUS == 1 && c == 0 && g == 0 0 : *RS_CLKEN[g]* [0:0] selected 1 : *RS_CLKEN[g]* [1:1] selected 2 : *RS_CLKEN[g]* [2:2] selected 3 : *RS_CLKEN[g]* [3:3] selected 4 : *RS_CLKEN[g]* [4:4] selected 5 : *RS_CLKEN[g]* [5:5] selected 6 : *RS_CLKEN[g]* [6:6] selected 7 : *RS_CLKEN[g]* [7:7] selected 8 : *TRIG_OUT* [c:c] selected 9 : Input tied to '0'; no update will occur 10 : *PL_EVT* [c:c] selected 11 : Input tied to '0'; no update will occur 14 : 0; no update will occur 15 : 1; update on GTM system clock selected
RW−Coding	TIO_PLUS == 1 && !(c == 0 && g == 0) 0 : *RS_CLKEN[g]* [0:0] selected 1 : *RS_CLKEN[g]* [1:1] selected 2 : *RS_CLKEN[g]* [2:2] selected 3 : *RS_CLKEN[g]* [3:3] selected 4 : *RS_CLKEN[g]* [4:4] selected 5 : *RS_CLKEN[g]* [5:5] selected 6 : *RS_CLKEN[g]* [6:6] selected 7 : *RS_CLKEN[g]* [7:7] selected 8 : *TRIG_OUT* [c:c] selected 9 : *TRIG_OUT* [c−1:c−1] selected 10 : *PL_EVT* [c:c] selected 11 : *PL_TRIG_OUT* [c−1:c−1] selected 14 : 0; no update will occur 15 : 1; update on GTM system clock selected

PL_S_MODE	
Description	Select mode of operation for S resource channel [c].
Loop	–
Bit Range	[13 : 12]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync

PL_S_MODE	
RW-Coding	0b00 : S resource operating as start buffer 0b01 : S resource operating as continue buffer 0b10 : S resource operating as counter; enable stop on *PL_TRIG_OUT* [c:c] 0b11 : S resource operating as counter; disable stop on *PL_TRIG_OUT* [c:c]
	Details for selection of mode of operations see *27.5.10 "TIO_PL Processing"* .

PL_FREEZE_S_EN	
Description	Enable S buffer freeze in channel [c].
Loop	-
Bit Range	$[14 : 14]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{TIO\_PLUS} == 1$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : S buffer freeze disabled 1 : S buffer freeze enabled
	Details for selection of mode of operations see corresponding chapters "Instruction termination with instruction freeze". **Note:** If **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** is set to 1, **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0 must be set, otherwise the behavior of the instruction execution is undefined.

PL_CYCLIC_BUFF	
Description	Enable circular buffer functionality in channel [c].
Loop	-
Bit Range	$[15 : 15]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$\mathrm{TIO\_PLUS} == 1$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Circular buffer not in use 1 : Circular buffer enabled

PL_CYCLIC_BUFF	
	Details for selection of mode of operations see corresponding chapters "Instruction termination with cyclic buffer usage". **Note:** If **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** is set to 1, **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_S_EN** = **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** = 0 must be set, otherwise the behavior of the instruction execution is undefined.

PL_O_MODE	
Description	Select mode of operation for O resource channel [c].
Loop	–
Bit Range	[18 : 16]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : O resource operating as start buffer 1 : O resource operating as continuous buffer 2 : O resource operating as shift left 3 : O resource operating as shift right 4 : O resource operating as compare (greater than or equal) ; signal capture disabled 5 : O resource operating as compare (greater than or equal) ; signal capture enabled 6 : O resource operating as compare (equal); signal capture disabled 7 : O resource operating as capture; signal capture enabled
	Details for selection of mode of operations see *27.5.10 "TIO_PL Processing"* .

PL_FREEZE_O_EN	
Description	Enable O buffer freeze in channel [c].
Loop	–
Bit Range	[19 : 19]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync

PL_FREEZE_O_EN	
RW-Coding	0 : O buffer freeze disabled 1 : O buffer freeze enabled
	Details for selection of mode of operations see corresponding chapters "Instruction termination with instruction freeze" **Note:** If **TIO[i]_G[g]_CH[c]_CTRL.PL_FREEZE_O_EN** is set to 1, **TIO[i]_G[g]_CH[c]_CTRL.PL_CYCLIC_BUFF** = 0 must be set, otherwise the behavior of the instruction execution is undefined.

PL_ODIS	
Description	Disable change of TIO[i]_O.CH[x] bit x = [g] * 8 + [c]
Loop	-
Bit Range	[20 : 20]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : O resource change of **TIO[i]_O.CH[g*8+c]** enabled 1 : O resource change of **TIO[i]_O.CH[g*8+c]** disabled
	Details for selection of mode of operations see corresponding chapters.

PL_SEL_IN	
Description	Select input source for O resource channel [c].
Loop	-
Bit Range	[21 : 21]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Use **TIO[i]_O.CH[g*8+c]** as input source 1 : Use **TIO[i]_S.CH[g*8+c]** as input source

PL_SEL_IN	
	Details for selection of mode of operations see corresponding chapters.

PL_O_TRIG_OUT_EN	
Description	Enable usage of TRIG_OUT[c] in instruction of O resource channel [c].
Loop	–
Bit Range	$[22 : 22]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{TIO\_PLUS} == 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : *TRIG_OUT* [c:c] signal disabled 1 : *TRIG_OUT* [c:c] signal enabled
	Details for selection of mode of operations see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"* .

PL_S_TRIG_OUT_EN	
Description	Enable usage of TRIG_OUT[c] in instruction of S resource channel [c].
Loop	–
Bit Range	$[23 : 23]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{TIO\_PLUS} == 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : *TRIG_OUT* [c:c] signal disabled 1 : *TRIG_OUT* [c:c] signal enabled
	Details for selection of mode of operations see chapter *27.5.10.2 "Trigger Event Generation for S Resource and O Resource"* .

PL_TRIG_OUT_EN_S_RE	
Description	TIO[i] channel [c] PL_TRIG_OUT source: Rising edge S[c] enable
Loop	–

PL_TRIG_OUT_EN_S_RE	
Bit Range	[24 : 24]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

PL_TRIG_OUT_EN_S_FE	
Description	TIO[i] channel [c] PL_TRIG_OUT source: Falling edge S[c] enable
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

PL_TRIG_OUT_EN_O_RE	
Description	TIO[i] channel [c] PL_TRIG_OUT source: Rising edge O[c] enable
Loop	–
Bit Range	[26 : 26]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

PL_TRIG_OUT_EN_O_RE	
RW-Coding	0 : Source disabled 1 : Source enabled

PL_TRIG_OUT_EN_O_FE	
Description	TIO[i] channel [c] PL_TRIG_OUT source: Falling edge O[c] enable
Loop	-
Bit Range	[27 : 27]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

PL_TRIG_OUT_EN_PREV_-TRIG	
Description	TIO[i] channel [c] PL_TRIG_OUT source: TRIG_OUT[c]-1 enable
Loop	-
Bit Range	[28 : 28]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Source disabled 1 : Source enabled

PL_TRIG_OUT_EN_PL_EVT	
Description	TIO[i] channel [c] PL_TRIG_OUT source: PL_EVT[c] enable
Loop	-
Bit Range	[29 : 29]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-

PL_TRIG_OUT_EN_PL_EVT	
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Source disabled 1 : Source enabled

PL_TRIG_OUT_EN_PREV_-PL_TRIG	
Description	TIO[i] channel [c] PL_TRIG_OUT source: PL_TRIG_OUT[c]–1 enable
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Source disabled 1 : Source enabled

PL_TRIG_OUT_UPD_EN	
Description	Select TIO[i] basic update source of channel [c].
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW–Coding	0 : Use *PL_UPDATE* [c:c] as update source 1 : Use *PL_TRIG_OUT* [c:c] as update source

27.7.37 TIO[i]_G[g]_CH[c]_CTRL2

Description	TIO[i] group [g] channel [c] control register

Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_CTRL2
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3014$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].CTRL2

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_CTRL2
Address	$0x400 * g + 0x40 * c + 0x3014$
C-Name	

DUAL_CMP_EN	
Description	TIO[i] channel [c] Trigger Output source: Rising edge S[c] enable
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$\text{TIO\_PLUS} == 1$
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Dual compare disabled (Single compare) 1 : Dual compare enabled

DUAL_CMP_MST_EN	
Description	TIO[i] channel [c] Trigger Output source: Falling edge S[c] enable
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	$\text{TIO\_PLUS} == 1$
Initial value	0

DUAL_CMP_MST_EN	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Dual compare master disabled (Slave) 1 : Dual compare master enabled

27.7.38 TIO[i]_G[g]_CH[c]_IRQ_NOTIFY

Description	TIO[i] channel [c] interrupt notification register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_IRQ_NOTIFY
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3004$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].IRQ_NOTIFY

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_IRQ_NOTIFY
Address	$0x400 * g + 0x40 * c + 0x3004$
C-Name	

S_RE_IRQ	
Description	Interrupt request Rising edge S[c]
Loop	–
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

S_FE_IRQ	
Description	Interrupt request Falling edge S[c]
Loop	–
Bit Range	[1 : 1]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

O_RE_IRQ	
Description	Interrupt request Rising edge O[c]
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

O_FE_IRQ	
Description	Interrupt request Falling edge O[c]
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0

O_FE_IRQ	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

UPDATE_IRQ	
Description	Interrupt request for update of channel [c]
Loop	–
Bit Range	$[4:4]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No interrupt raised 1 : Interrupt was raised
W-Coding	0 : No action 1 : Clear interrupt

PL_EVT_IRQ	
Description	Interrupt request PL_EVT of channel [c]
Loop	–
Bit Range	$[5:5]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	oneToClear
Condition	$\mathrm{TIO\_PLUS} == 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : IRQ notify flag is cleared 1 : IRQ notify flag is set
W-Coding	0 : No action 1 : Clear IRQ notify flag

27.7.39 TIO[i]_G[g]_CH[c]_IRQ_EN

Description	TIO[i] channel [c] interrupt enable register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_IRQ_EN
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3008$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].IRQ_EN

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_IRQ_EN
Address	$0x400 * g + 0x40 * c + 0x3008$
C-Name	

S_RE_IRQ_EN	
Description	Interrupt enable request Rising edge S[c]
Loop	-
Bit Range	$[0 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

S_FE_IRQ_EN	
Description	Interrupt enable request Falling edge S[c]
Loop	-
Bit Range	$[1 : 1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0

S_FE_IRQ_EN	
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

O_RE_IRQ_EN	
Description	Interrupt enable request Rising edge O[c]
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

O_FE_IRQ_EN	
Description	Interrupt enable request Falling edge O[c]
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

UPDATE_IRQ_EN	
Description	Interrupt enable request for update of channel [c]
Loop	–
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–

UPDATE_IRQ_EN	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

PL_EVT_IRQ_EN	
Description	Interrupt enable request PL_EVT of channel [c]
Loop	–
Bit Range	$[5:5]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	$\mathrm{TIO\_PLUS} == 1$
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Disable interrupt 1 : Enable interrupt

27.7.40 TIO[i]_G[g]_CH[c]_IRQ_FORCINT

Description	TIO[i] channel [c] force interrupt register
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\mathrm{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\mathrm{TIO}[i]\mathrm{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_IRQ_FORCINT
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x300C$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].IRQ_FORCINT

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_IRQ_FORCINT
Address	$0x400 * g + 0x40 * c + 0x300C$
C-Name	

TRG_S_RE_IRQ	
Description	Trigger the bit TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.S_RE_IRQ by software.
Loop	–
Bit Range	$[0:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.S_RE_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** . If a write occurs in case the protection is active, the access response is always *TIO_AEI_STATUS* = 0b10.

TRG_S_FE_IRQ	
Description	Trigger the bit TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.S_FE_IRQ by software.
Loop	–
Bit Range	$[1:1]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.S_FE_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** . If a write occurs in case the protection is active, the access response is always *TIO_AEI_STATUS* = 0b10.

TRG_O_RE_IRQ	
Description	Trigger the bit TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.O_RE_IRQ by software.

TRG_O_RE_IRQ	
Loop	–
Bit Range	[2 : 2]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No status
W–Coding	0 : No action 1 : Force setting of **TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.O_RE_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** . If a write occurs in case the protection is active, the access response is always *TIO_AEI_STATUS* = 0b10.

TRG_O_FE_IRQ	
Description	Trigger the bit TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.O_FE_IRQ by software.
Loop	–
Bit Range	[3 : 3]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No status
W–Coding	0 : No action 1 : Force setting of **TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.O_FE_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** . If a write occurs in case the protection is active, the access response is always *TIO_AEI_STATUS* = 0b10.

TRG_UPDATE_IRQ	
Description	Trigger the bit TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.UPDATE_IRQ by software.
Loop	–

TRG_UPDATE_IRQ	
Bit Range	[4 : 4]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.UPDATE_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** . If a write occurs in case the protection is active, the access response is always *TIO_AEI_STATUS* = 0b10.

TRG_PL_EVT_IRQ	
Description	Trigger the bit TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.PL_EVT_IRQ by software.
Loop	–
Bit Range	[5 : 5]
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	clear
Condition	TIO_PLUS == 1
Initial value	0
Protect Enable Cond	RF_PROT == 1
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No status
W-Coding	0 : No action 1 : Force setting of **TIO[i]_G[g]_CH[c]_IRQ_NOTIFY.PL_EVT_IRQ**
	Note: This bit is cleared automatically after write. **Note:** This bit is write protected by bit **GTM_CTRL.RF_PROT** . If a write occurs in case the protection is active, the access response is always *TIO_AEI_STATUS* = 0b10.

27.7.41 TIO[i]_G[g]_CH[c]_IRQ_MODE

Description	TIO[i] channel [c] IRQ mode configuration register

Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_IRQ_MODE
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3010$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].IRQ_MODE

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_IRQ_MODE
Address	$0x400 * g + 0x40 * c + 0x3010$
C-Name	

IRQ_MODE	
Description	IRQ mode selection
Loop	-
Bit Range	$[1 : 0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	IRQ_MODE_RST_VAL
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Level mode 0b01 : Pulse mode 0b10 : Pulse-Notify mode 0b11 : Single-Pulse mode
	Note: The interrupt modes are described in section *3.12 "GTM-IP Interrupt Concept"* .

27.7.42 TIO[i]_G[g]_CH[c]_SINST

Description	TIO[i] channel [c] resource S instruction register (buffer operation)TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE=0b0-
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER

Clock Active Cond	TIO[i]_CLK_ENABLE == 1

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_SINST
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3020$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].SINST

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_SINST
Address	$0x400 * g + 0x40 * c + 0x3020$
C-Name	

OP	
Description	Operand
Loop	-
Bit Range	[23 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.9 "S Resource Start Buffer"* , *27.5.10.10 "S Resource Continue Buffer"* .

CMD	
Description	Command
Loop	-
Bit Range	[29 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync

CMD	
	Details of command definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.9 "S Resource Start Buffer"* , *27.5.10.10 "S Resource Continue Buffer"* .

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	$[30 : 30]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	$[31 : 31]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.43 TIO[i]_G[g]_CH[c]_SINST_COUNT

Description	TIO[i] channel [c] resource S instruction register (counter operation)TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE=0b1- and (TIO[i]_G[g]_OP_USAGE.MODE[c]=0b10- or TIO[i]_G[g]_OP_USAGE.MODE[c]=0b1-0)
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE >= 2 && (TIO[i]_G[g]_OP_USAGE.MODE[c] >= 4 || TIO[i]_G[g]_OP_USAGE.MODE[c] <= 6)

Register Reference	*27.7.42 "TIO[i]_G[g]_CH[c]_SINST"*

Interface: CPU
Interface: MCS[i]

OP	
Description	Operand
Loop	–
Bit Range	[23 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.8 "S Resource Counter Instructions"* .

CNT_COND	
Description	Command
Loop	–
Bit Range	[26 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Increment on update resolution if signal is low; decrement on update resolution if signal is high 1 : Increment on update resolution if signal is low 2 : Increment on update resolution if signal is high 3 : Increment on update resolution 5 : Increment on each falling edge 6 : Increment on each rising edge 7 : Increment on any edge

CNT_ITERATIONS	
Description	number of activations
Loop	–
Bit Range	[28 : 27]

CNT_ITERATIONS	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : No further activation 0b01 : One activation 0b10 : Two activations 0b11 : Endless operation

CMD_ACTIVE	
Description	counter active
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Counting disabled 1 : Counting enabled

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync

DATA_PUSH_EN	
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	-
Bit Range	$[31:31]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.44 TIO[i]_G[g]_CH[c]_SINST_COUNT12

Description	TIO[i] channel [c] resource S instruction register (counter operation)TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE=0b1- and !(TIO[i]_G[g]_OP_USAGE.MODE[c]=0b10- or TIO[i]_G[g]_OP_USAGE.MODE[c]=0b1-0)
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE >= 2 && !(TIO[i]_G[g]_OP_USAGE.MODE[c] >= 4 || TIO[i]_G[g]_OP_USAGE.MODE[c] <= 6)
Register Reference	*27.7.42 "TIO[i]_G[g]_CH[c]_SINST"*

Interface: CPU
Interface: MCS[i]

OPA	
Description	Operand A
Loop	-
Bit Range	$[11:0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync

OPA	
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.8 "S Resource Counter Instructions"* , *27.5.10.14 "Operand Usage"* .

OPB	
Description	Operand B
Loop	–
Bit Range	[23 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.8 "S Resource Counter Instructions"* , *27.5.10.14 "Operand Usage"* .

CNT_COND	
Description	Command
Loop	–
Bit Range	[26 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Increment on update resolution if signal is low; decrement on update resolution if signal is high 1 : Increment on update resolution if signal is low 2 : Increment on update resolution if signal is high 3 : Increment on update resolution 5 : Increment on each falling edge 6 : Increment on each rising edge 7 : Increment on any edge

CNT_ITERATIONS	
Description	number of activations
Loop	–

CNT_ITERATIONS	
Bit Range	[28 : 27]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : No further activation 0b01 : One activation 0b10 : Two activations 0b11 : Endless operation

CMD_ACTIVE	
Description	counter active
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Counting disabled 1 : Counting enabled

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

DATA_PUSH_EN	
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	-
Bit Range	$[31 : 31]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.45 TIO[i]_G[g]_CH[c]_SCMD

Description	TIO[i] channel [c] resource S command register (buffer operation)TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE=0b0-
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_SCMD
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3024$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].SCMD

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_SCMD
Address	$0x400 * g + 0x40 * c + 0x3024$
C-Name	

CMD	
Description	Command

CMD	
Loop	–
Bit Range	[29 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of command definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.9 "S Resource Start Buffer"* , *27.5.10.10 "S Resource Continue Buffer"* .

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

INSTR_PULL_EN	
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.46 TIO[i]_G[g]_CH[c]_SCMD_COUNT

Description	TIO[i] channel [c] resource S command register (counter operation)TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE=0b1-
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE == 2 || TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE == 3
Register Reference	*27.7.45 "TIO[i]_G[g]_CH[c]_SCMD"*

Interface: CPU
Interface: MCS[i]

CNT_COND	
Description	Command
Loop	-
Bit Range	$[26 : 24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Increment on update resolution if signal is low; decrement on update resolution if signal is high 1 : Increment on update resolution if signal is low 2 : Increment on update resolution if signal is high 3 : Increment on update resolution 5 : Increment on each falling edge 6 : Increment on each rising edge 7 : Increment on any edge

CNT_ITERATIONS	
Description	number of activations
Loop	-
Bit Range	$[28 : 27]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-

CNT_ITERATIONS	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : No further activation 0b01 : One activation 0b10 : Two activations 0b11 : Endless operation

CMD_ACTIVE	
Description	counter active
Loop	–
Bit Range	[29 : 29]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Counting disabled 1 : Counting enabled

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	-
Bit Range	$[31 : 31]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.47 TIO[i]_G[g]_CH[c]_SOP

Description	TIO[i] channel [c] resource S operand registerTIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE=0b0- or (TIO[i]_G[g]_OP_USAGE.MODE[c]=0b10- or TIO[i]_G[g]_OP_USAGE.MODE[c]=0b1-0)
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_SOP
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3028$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].SOP

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_SOP
Address	$0x400 * g + 0x40 * c + 0x3028$
C-Name	

OP	
Description	Operand
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-

OP	
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.9 "S Resource Start Buffer"* , *27.5.10.10 "S Resource Continue Buffer"* .

27.7.48 TIO[i]_G[g]_CH[c]_SOP12

Description	TIO[i] channel [c] resource S operand A/B registerTIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE=0b1– and !(TIO[i]_G[g]_OP_USAGE.MODE[c]=0b10– or TIO[i]_G[g]_OP_USAGE.MODE[c]=0b1–0)
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
View Condition	(TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE == 2 || TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE == 3) && !(TIO[i]_G[g]_OP_USAGE.MODE[c] <= 6 || TIO[i]_G[g]_OP_USAGE.MODE[c] >= 4)
Register Reference	*27.7.47 "TIO[i]_G[g]_CH[c]_SOP"*

Interface: CPU

Interface: MCS[i]

OPA	
Description	Operand A
Loop	–
Bit Range	[11 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.8 "S Resource Counter Instructions"* .

OPB	
Description	Operand B
Loop	–
Bit Range	[23 : 12]
Access Type	RW
Volatile	True

OPB	
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_S_MODE** ; see *27.5.10.8 "S Resource Counter Instructions"* .

27.7.49 TIO[i]_G[g]_CH[c]_OINST

Description	TIO[i] channel [c] resource O instruction register (buffer operation)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE =0b00–
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_OINST
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3030$
C–Name	GTM.CLS[i].TIO.G[g].CH[c].OINST

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_OINST
Address	$0x400 * g + 0x40 * c + 0x3030$
C–Name	

OP	
Description	Operand
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

OP	
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.11 "O Resource Start Buffer"* , *27.5.10.12 "O Resource Continue Buffer"* .

CMD	
Description	Command
Loop	–
Bit Range	[29 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of command definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.11 "O Resource Start Buffer"* , *27.5.10.12 "O Resource Continue Buffer"* .

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	[31 : 31]
Access Type	RW

INSTR_PULL_EN	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.50 TIO[i]_G[g]_CH[c]_OINST_CAP

Description	TIO[i] channel [c] resource O instruction register (capture operation)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE =0b111
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE == 7
Register Reference	*27.7.49 "TIO[i]_G[g]_CH[c]_OINST"*

Interface: CPU
Interface: MCS[i]

OP	
Description	Operand
Loop	–
Bit Range	[23 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.6 "O Resource Capture Instructions"* .

O_ACTION	
Description	action on trigger
Loop	–
Bit Range	[25 : 24]
Access Type	RW

O_ACTION	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : Set **TIO[i]_O** [g*8+c : g*8+c]=0 0b01 : Set **TIO[i]_O** [g*8+c : g*8+c]=1 0b10 : Hold **TIO[i]_O** [g*8+c : g*8+c] 0b11 : Invert **TIO[i]_O** [g*8+c : g*8+c]

SEL_TB	
Description	TB source selection
Loop	–
Bit Range	[27 : 26]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : *RS_TB0[g]* selected 0b01 : *RS_TB1[g]* selected 0b10 : *RS_TB2[g]* selected 0b11 : **TIO[i]_G[g]_CH[c]_SINST.OP** of channel c selected

CMD_ACTIVE_CC	
Description	capture function in use
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

CMD_ACTIVE_CC	
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : stopped 0b01 : prohibited 0b10 : capture 0b11 : prohibited

DATA_PUSH_EN	
Description	Enable data capture
Loop	-
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	-
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.51 TIO[i]_G[g]_CH[c]_OINST_SHIFT

Description	TIO[i] channel [c] resource O instruction register (shift operation)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE = 0b01-

Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE == 2 || TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE == 3
Register Reference	*27.7.49 "TIO[i]_G[g]_CH[c]_OINST"*

Interface: CPU
Interface: MCS[i]

OP	
Description	Operand
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.5 "O Resource Shift Instructions"* .

SHIFT_LEN	
Description	Number of bits to shift [1, ..., 31]
Loop	–
Bit Range	$[28 : 24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	A shift length of 0 is not allowed.

CMD_ACTIVE	
Description	shift operation active
Loop	–
Bit Range	$[29 : 29]$

CMD_ACTIVE	
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Shift operation enabled 1 : Shift operation disabled

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync

INSTR_PULL_EN	
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.52 TIO[i]_G[g]_CH[c]_OINST_COMP

Description	TIO[i] channel [c] resource O instruction register (compareaction operation)(TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b10- or TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b110) and TIO[i]_G[g]_OP_USAGE.MODE[c]=0b11-
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
View Condition	(TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE >= 4 && TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE <= 6) && TIO[i]_G[g]_OP_USAGE.MODE[c] >= 6
Register Reference	*27.7.49 "TIO[i]_G[g]_CH[c]_OINST"*

Interface: CPU
Interface: MCS[i]

OP	
Description	Operand
Loop	-
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.7 "O Resource Compare Instructions"* .

O_ACTION	
Description	action on compare
Loop	-
Bit Range	$[25 : 24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

O_ACTION	
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : Set **TIO[i]_O** [g*8+c : g*8+c]=0 0b01 : Set **TIO[i]_O** [g*8+c : g*8+c]=1 0b10 : Hold **TIO[i]_O** [g*8+c : g*8+c] 0b11 : Invert **TIO[i]_O** [g*8+c : g*8+c]

SEL_TB	
Description	TB source selection
Loop	–
Bit Range	[27 : 26]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : *RS_TB0[g]* selected 0b01 : *RS_TB1[g]* selected 0b10 : *RS_TB2[g]* selected 0b11 : **TIO[i]_G[g]_CH[c]_SINST.OP** of channel c selected

CMD_ACTIVE_CC	
Description	compare function in use
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : No compare operation active 0b01 : Dual compare operation active 0b10 : Single compare operation active 0b11 : Single compare operation active

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	$[30 : 30]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW–Coding	0 : Data capture disabled / serve first 1 : Data capture enabled / serve last

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	$[31 : 31]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW–Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.53 TIO[i]_G[g]_CH[c]_OINST_COMP12

Description	TIO[i] channel [c] resource O instruction register (compareaction operation)(TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b10– or TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b110) and !(TIO[i]_G[g]_OP_USAGE.MODE[c]=0b11–)
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
View Condition	(TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE >= 4 && TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE <= 6) && TIO[i]_G[g]_OP_USAGE.MODE[c] < 6
Register Reference	*27.7.49 "TIO[i]_G[g]_CH[c]_OINST"*

Interface: CPU

Interface: MCS[i]

OPA	
Description	Operand A
Loop	–
Bit Range	[11 : 0]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.8 "S Resource Counter Instructions"* , *27.5.10.14 "Operand Usage"* .

OPB	
Description	Operand B
Loop	–
Bit Range	[23 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.8 "S Resource Counter Instructions"* , *27.5.10.14 "Operand Usage"* .

O_ACTION	
Description	action on compare
Loop	–
Bit Range	[25 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

O_ACTION	
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : Set **TIO[i]_O** [g*8+c : g*8+c]=0 0b01 : Set **TIO[i]_O** [g*8+c : g*8+c]=1 0b10 : Hold **TIO[i]_O** [g*8+c : g*8+c] 0b11 : Invert **TIO[i]_O** [g*8+c : g*8+c]

SEL_TB	
Description	TB source selection
Loop	–
Bit Range	[27 : 26]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : *RS_TB0[g]* selected 0b01 : *RS_TB1[g]* selected 0b10 : *RS_TB2[g]* selected 0b11 : **TIO[i]_G[g]_CH[c]_SINST.OP** of channel c selected

CMD_ACTIVE_CC	
Description	compare function in use
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : No compare operation active 0b01 : Dual compare operation active 0b10 : Single compare operation active 0b11 : Single compare operation active

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.54 TIO[i]_G[g]_CH[c]_OCMD

Description	TIO[i] channel [c] resource O command register (buffer operation)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b00-
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_OCMD

Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3034$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].OCMD

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_OCMD
Address	$0x400 * g + 0x40 * c + 0x3034$
C-Name	

CMD	
Description	Command
Loop	-
Bit Range	$[29 : 24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of command definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.11 "O Resource Start Buffer"* , *27.5.10.12 "O Resource Continue Buffer"* .

DATA_PUSH_EN	
Description	Enable data capture
Loop	-
Bit Range	$[30 : 30]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	-

INSTR_PULL_EN	
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.55 TIO[i]_G[g]_CH[c]_OCMD_SHIFT

Description	TIO[i] channel [c] resource O command register (shift operation)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b01-
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE == 2 || TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE == 3
Register Reference	*27.7.54 "TIO[i]_G[g]_CH[c]_OCMD"*

Interface: CPU
Interface: MCS[i]

SHIFT_LEN	
Description	Number of bits to shift [1, ..., 31]
Loop	–
Bit Range	[28 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	A shift length of 0 is not allowed.

CMD_ACTIVE	
Description	shift operation active
Loop	–

CMD_ACTIVE	
Bit Range	[29 : 29]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Shift operation deactivated 1 : Shift operation activated

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync

INSTR_PULL_EN	
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.56 TIO[i]_G[g]_CH[c]_OCMD_COMP

Description	TIO[i] channel [c] resource O command register (compareaction operation)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b10- or TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b110
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE >= 4 && TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE <= 6
Register Reference	*27.7.54 "TIO[i]_G[g]_CH[c]_OCMD"*

Interface: CPU
Interface: MCS[i]

O_ACTION	
Description	action on compare
Loop	-
Bit Range	$[25 : 24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : Set **TIO[i]_O** [g*8+c : g*8+c]=0 0b01 : Set **TIO[i]_O** [g*8+c : g*8+c]=1 0b10 : Hold **TIO[i]_O** [g*8+c : g*8+c] 0b11 : Invert **TIO[i]_O** [g*8+c : g*8+c]

SEL_TB	
Description	TB source selection
Loop	-
Bit Range	$[27 : 26]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-

SEL_TB	
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : *RS_TB0[g]* selected 0b01 : *RS_TB1[g]* selected 0b10 : *RS_TB2[g]* selected 0b11 : **TIO[i]_G[g]_CH[c]_SINST.OP** of channel c selected

CMD_ACTIVE_CC	
Description	compare function in use
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : No compare operation active 0b01 : Dual compare operation active 0b10 : Single compare operation active 0b11 : Single compare operation active

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–

INSTR_PULL_EN	
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.57 TIO[i]_G[g]_CH[c]_OCMD_CAP

Description	TIO[i] channel [c] resource O command register (capture operation)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b111
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE == 7
Register Reference	*27.7.54 "TIO[i]_G[g]_CH[c]_OCMD"*

Interface: CPU
Interface: MCS[i]

O_ACTION	
Description	action on trigger
Loop	–
Bit Range	[25 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : Set **TIO[i]_O** [g*8+c : g*8+c]=0 0b01 : Set **TIO[i]_O** [g*8+c : g*8+c]=1 0b10 : Hold **TIO[i]_O** [g*8+c : g*8+c] 0b11 : Invert **TIO[i]_O** [g*8+c : g*8+c]

SEL_TB	
Description	TB source selection
Loop	–
Bit Range	[27 : 26]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : *RS_TB0[g]* selected 0b01 : *RS_TB1[g]* selected 0b10 : *RS_TB2[g]* selected 0b11 : **TIO[i]_G[g]_CH[c]_SINST.OP** of channel c selected

CMD_ACTIVE_CC	
Description	capture function in use
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : Stopped 0b01 : Prohibited 0b10 : Capture active 0b11 : Prohibited

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–

DATA_PUSH_EN	
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW–Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW–Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.58 TIO[i]_G[g]_CH[c]_OOP

Description	TIO[i] channel [c] resource O operand register!(TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b1–-) or (TIO[i]_G[g]_OP_USAGE.MODE[c]=0b11–)
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_OOP
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x3038$
C–Name	GTM.CLS[i].TIO.G[g].CH[c].OOP

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_OOP
Address	$0x400 * g + 0x40 * c + 0x3038$
C–Name	

OP	
Description	Operand
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on PL_O_MODE; see *27.5.10.6 "O Resource Capture Instructions"* , *27.5.10.7 "O Resource Compare Instructions"* .

27.7.59 TIO[i]_G[g]_CH[c]_OOP12

Description	TIO[i] channel [c] resource O operand A/B registerTIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b1-- and !(TIO[i]_G[g]_OP_USAGE.MODE[c]=0b11-)
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
View Condition	(TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE >= 4) && !(TIO[i]_G[g]_OP_USAGE.MODE[c] >= 6)
Register Reference	*27.7.58 "TIO[i]_G[g]_CH[c]_OOP"*

Interface: CPU
Interface: MCS[i]

OPA	
Description	Operand A
Loop	–
Bit Range	$[11 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.14 "Operand Usage"* .

OPB	
Description	Operand B
Loop	–
Bit Range	$[23 : 12]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.14 "Operand Usage"* .

27.7.60 TIO[i]_G[g]_CH[c]_OCAPTURE

Description	TIO[i] channel [c] resource O capture register (capture instruction)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b1--
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
View Condition	TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE >= 4
Register Reference	*27.7.58 "TIO[i]_G[g]_CH[c]_OOP"*

Interface: CPU
Interface: MCS[i]

OP	
Description	value of TB at capture
Loop	–
Bit Range	$[23 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.6 "O Resource Capture Instructions"* .

O_OUT	
Description	value of O at capture
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
W-Coding	0 : Signal level low 1 : Signal level high

S_OUT	
Description	value of S at capture
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
R-Coding	0 : Signal level low 1 : Signal level high

SEL_TB	
Description	TB source selection
Loop	–
Bit Range	[27 : 26]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–

SEL_TB	
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : *RS_TB0[g]* selected 0b01 : *RS_TB1[g]* selected 0b10 : *RS_TB2[g]* selected 0b11 : **TIO[i]_G[g]_CH[c]_SINST.OP** of channel c selected

CMD_ACTIVE_CC	
Description	compare/capture function in use
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : Stopped 0b01 : Prohibited 0b10 : Capture active 0b11 : Prohibited

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–

INSTR_PULL_EN	
Bit Range	$[31 : 31]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.61 TIO[i]_G[g]_CH[c]_OCAPTURE12

Description	TIO[i] channel [c] resource O capture register (capture instruction)TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE=0b1-- and !(TIO[i]_G[g]_OP_USAGE.MODE[c]=0b11-)
Loop	$i = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\mathrm{CTIO}[i] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
View Condition	(TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE >= 4) && !(TIO[i]_G[g]_OP_USAGE.MODE[c] >= 6)
Register Reference	*27.7.58 "TIO[i]_G[g]_CH[c]_OOP"*

Interface: CPU
Interface: MCS[i]

OPA	
Description	Operand A
Loop	–
Bit Range	$[11 : 0]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.6 "O Resource Capture Instructions"* , *27.5.10.14 "Operand Usage"* .

OPB	
Description	Operand B
Loop	–

OPB	
Bit Range	[23 : 12]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Details of operand definition is dependent on **TIO[i]_G[g]_CH[c]_CTRL.PL_O_MODE** ; see *27.5.10.6 "O Resource Capture Instructions"* , *27.5.10.14 "Operand Usage"* .

O_OUT	
Description	value of O at capture
Loop	–
Bit Range	[24 : 24]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
W-Coding	0 : Signal level low 1 : Signal level high

S_OUT	
Description	value of S at capture
Loop	–
Bit Range	[25 : 25]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync

S_OUT	
R-Coding	0 : Signal level low 1 : Signal level high

SEL_TB	
Description	TB source selection
Loop	–
Bit Range	[27 : 26]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : *RS_TB0[g]* selected 0b01 : *RS_TB1[g]* selected 0b10 : *RS_TB2[g]* selected 0b11 : **TIO[i]_G[g]_CH[c]_SINST.OP** of channel c selected

CMD_ACTIVE_CC	
Description	compare/capture function in use
Loop	–
Bit Range	[29 : 28]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0b00 : Stopped 0b01 : Prohibited 0b10 : Capture active 0b11 : Prohibited

DATA_PUSH_EN	
Description	Enable data capture
Loop	–
Bit Range	[30 : 30]
Access Type	RW

DATA_PUSH_EN	
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Data capture disabled 1 : Data capture enabled

INSTR_PULL_EN	
Description	Enable instruction fetch
Loop	–
Bit Range	[31 : 31]
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
RW-Coding	0 : Instruction fetch disabled 1 : Instruction fetch enabled

27.7.62 TIO[i]_G[g]_CH[c]_SHIFTCNT

Description	TIO[i] channel [c] resource shift count register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_CH[c]_SHIFTCNT
Address	$0x20000 * i + 0x400 * g + 0x40 * c + 0x303C$
C-Name	GTM.CLS[i].TIO.G[g].CH[c].SHIFTCNT

Interface: MCS[i]

Name	TIO[i]_G[g]_CH[c]_SHIFTCNT

Address	$0x400 * g + 0x40 * c + 0x303C$
C-Name	

CNT	
Description	Shift counter value
Loop	-
Bit Range	$[4:0]$
Access Type	R
Volatile	True
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	-
Reset group	HW_RESET: async GTM_RESET: async TIO[i]_G[g]_RESET_CH[c]: sync
	Represents the actual shift counter value; see *27.5.10.5 "O Resource Shift Instructions"* .

27.7.63 TIO[i]_G[g]_ISEL[q]_CTRL1

Description	TIO[i] input selection register 1
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; q = \{n : 0 <= n <= 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_ISEL0_CTRL1
Address	$0x20000 * i + 0x400 * g + 0x3200$
C-Name	GTM.CLS[i].TIO.G[g].ISEL0_CTRL1
Name	TIO[i]_G[g]_ISEL1_CTRL1
Address	$0x20000 * i + 0x400 * g + 0x3220$
C-Name	GTM.CLS[i].TIO.G[g].ISEL1_CTRL1

Interface: MCS[i]

Name	TIO[i]_G[g]_ISEL0_CTRL1
Address	$0x400 * g + 0x3200$
C-Name	
Name	TIO[i]_G[g]_ISEL1_CTRL1
Address	$0x400 * g + 0x3220$

C-Name	

LUT2_[k]	
Description	2 bit Lookup table function for channel c=[q]*4+[k]
Loop	$k = \{n : 0 <= n <= 3\}$
Bit Range	$[k * 4 + 3 : k * 4]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	12
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, k + 24, k + 24) == 0
Reset group	HW_RESET: async GTM_RESET: async
	2 bit logic function in use with index = *SYNC_OUT* [c:c] *2 + *O_OUT* [c-1:c-1]. Details see chapter *27.4.2 "TIO Input Selection"* **Note:** This bit field is write protected by bit **TIO[i]_G[g]_ISEL[q]_CTRL1.WRITE_EN[k]** = 0.

OUT_SEL[k]	
Description	select signal for ISEL_OUT[c] where c =[q]*4+[k]
Loop	$k = \{n : 0 <= n <= 3\}$
Bit Range	$[k + 16 : k + 16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	0
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, k + 24, k + 24) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Use 2 bit lookup table for channel c=[q]*4+[k] 1 : Use 3 bit lookup table for channel c=[q]*4+[k]
	Note: This bit field is write protected by bit **TIO[i]_G[g]_ISEL[q]_CTRL1.WRITE_EN[k]** = 0.

WRITE_EN[k]	
Description	enable writing of configuration bit fields LUT2_[k], OUT_SEL[k]
Loop	$k = \{n : 0 <= n <= 3\}$
Bit Range	$[k + 24 : k + 24]$
Access Type	RW

WRITE_EN[k]	
Volatile	True
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R–Coding	0 : No action
W–Coding	0 : No action 1 : Enable writing of bit field **TIO[i]_G[g]_ISEL[q]_CTRL1.LUT2_[k]** , **TIO[i]_G[g]_ISEL[q]_CTRL1.OUT_SEL[k]**
	Note: This bit is cleared automatically after write.

27.7.64 TIO[i]_G[g]_ISEL[q]_CTRL2

Description	TIO[i] input selection register 2
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; q = \{n : 0 <= n <= 1\}$
Condition	$\text{CTIO}[i] == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_ISEL0_CTRL2
Address	$0x20000 * i + 0x400 * g + 0x3204$
C–Name	GTM.CLS[i].TIO.G[g].ISEL0_CTRL2
Name	TIO[i]_G[g]_ISEL1_CTRL2
Address	$0x20000 * i + 0x400 * g + 0x3224$
C–Name	GTM.CLS[i].TIO.G[g].ISEL1_CTRL2

Interface: MCS[i]

Name	TIO[i]_G[g]_ISEL0_CTRL2
Address	$0x400 * g + 0x3204$
C–Name	
Name	TIO[i]_G[g]_ISEL1_CTRL2
Address	$0x400 * g + 0x3224$
C–Name	

LUT3	
Description	3 bit Lookup table function for quad=[q]; channels [q]*4+2 .. [q]*4
Loop	–

LUT3	
Bit Range	$[7:0]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
	3 bit logic function in use with index = *LUT3IN* [2:2] *4 + *LUT3IN* [1:1] *2 + *LUT3IN* [0:0]. Details see chapter *27.4.2 "TIO Input Selection"*

QOUT_SEL	
Description	select source for ISEL_QOUT[q] signal in quad = [q]
Loop	–
Bit Range	$[17:16]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0b00 : Use signal *LUT3OUT* 0b01 : Use signal *ISEL_QOUT_PREV[q]* 0b10 : Use signal *ISEL_QOUT_NEXT[q]* 0b11 : Use signal *LUT2OUT* [[q]*4+3:[q]*4+3]

LUT3IN_SEL[k]	
Description	select source for LUT3_IN[k] signal in quad = [q]
Loop	$k = \{n : 0 <= n <= 2\}$
Bit Range	$[k+20 : k+20]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	–
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async

LUT3IN_SEL[k]	
RW-Coding	0b0 : Use signal *LUT2OUT* [q*4+k:q*4+k] 0b1 : Use signal *LUT2OUT_PREV* [k:k]

27.7.65 TIO[i]_G[g]_OP_USAGE

Description	TIO[i] operand usage selection register
Loop	$i = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}$
Condition	$\text{CTIO}[i] == 1 \&\& \text{TIO\_PLUS} == 1$
Storage Type	REGISTER
Clock Active Cond	$\text{TIO}[i]\text{\_CLK\_ENABLE} == 1$

Interface: CPU

Name	TIO[i]_G[g]_OP_USAGE
Address	$0x20000 * i + 0x400 * g + 0x3240$
C-Name	GTM.CLS[i].TIO.G[g].OP_USAGE

Interface: MCS[i]

Name	TIO[i]_G[g]_OP_USAGE
Address	$0x400 * g + 0x3240$
C-Name	

MODE[c]	
Description	operand usage of S/ O resource
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c * 3 + 2 : c * 3]$
Access Type	RW
Volatile	False
Multithread	False
ModifiedWriteValue	-
Condition	-
Initial value	6
Protect Enable Cond	bitrange(CLS[i]_AEI_ARB_WDATA, c + 24, c + 24) == 0
Reset group	HW_RESET: async GTM_RESET: async
RW-Coding	0 : Count **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA** and **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB** ; compare **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA** 1 : Count **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA** and **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB** ; compare **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB** 2 : Count **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA** ; compare **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA** 3 : Count **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB** ; compare **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB** 4 : Count **TIO[i]_G[g]_CH[c]_SINST.OP** ; compare **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPA** 5 : Count **TIO[i]_G[g]_CH[c]_SINST.OP** ; compare **TIO[i]_G[g]_CH[c]_OINST_COMP12.OPB** 6 : Count **TIO[i]_G[g]_CH[c]_SINST.OP** ; compare **TIO[i]_G[g]_CH[c]_OINST_COMP.OP** 7 : Count **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPA** and **TIO[i]_G[g]_CH[c]_SINST_COUNT12.OPB** ; compare **TIO[i]_G[g]_CH[c]_OINST_COMP.OP**

MODE[c]	
	Note: This bit field is write protected by bit **TIO[i]_G[g]_OP_USAGE.WRITE_EN[c]** =0

WRITE_EN[c]	
Description	enable writing of configuration bit fields TIO[i]_G[g]_OP_USAGE.MODE[c]
Loop	$c = \{n : 0 <= n <= 7\}$
Bit Range	$[c+24 : c+24]$
Access Type	RW
Volatile	True
Multithread	False
ModifiedWriteValue	clear
Condition	–
Initial value	0
Protect Enable Cond	–
Reset group	HW_RESET: async GTM_RESET: async
R-Coding	0 : No action
W-Coding	0 : No action 1 : Enable writing of bit field **TIO[i]_G[g]_OP_USAGE.MODE[c]**
	Note: This bit is cleared automatically after write.

27.8 TIO Port Description

27.8.1 TIO_AEI Interface

Loop	–
Condition	–
Format	–

TIO_AEI_STATUS	
Description	AEI status
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	1 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

27.8.2 TIO CCM Interface

Loop	–
Condition	–
Format	–

CCM_CLKEN	
Description	Cluster cmu clock resolution driven by signal CCM[i]_CLK_RES[7:0]
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TB0	
Description	Cluster tbu timebase 0 driven by CCM[i]_TBU_TS0 of instance i
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TB1	
Description	Cluster tbu timebase 1 driven by CCM[i]_TBU_TS1 of instance i
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

CCM_TB2	
Description	Cluster tbu timebase 2 driven by CCM[i]_TBU_TS2 of instance i
Loop	–
Condition	–
Logical Name	–

CCM_TB2	
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	23 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

27.8.3 TIO Signal Interface

Loop	$g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}$
Condition	–
Format	–

TIO_G[g]_IN	
Description	TIO input signals for group [g]; port of instance i will be driven by GTM toplevel port GTM_TIO[i]_G[g]_IN
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	isAsyncInput
Operational Reset	GTM_RESET
Reset Value	–

TIO_G[g]_OUT	
Description	TIO O output signals for group [g]; will connect to the corresponding DTM modules
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIO_G[g]_S_OUT	
Description	TIO S output signals for group [g]; will connect to the corresponding DTM modules
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT

TIO_G[g]_S_OUT	
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

27.8.4 TIO Interrupt Interface

Loop	$g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}$
Condition	–
Format	–

TIO_G[g]_IRQ	
Description	TIO interrupt signals (configurable mode pulse/level) for group [g]; port of instance i will connect to GTM toplevel port GTM_TIO[i]_G[g]_IRQ
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	OUT
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

TIO_G[g]_IRQ_CLR	
Description	TIO interrupt clear for group [g]; port of instance i will be driven by GTM toplevel port GTM_TIO[i]_G[g]_IRQ_CLR
Loop	–
Condition	–
Logical Name	–
Port Type	STD_LOGIC_VECTOR
Port Direction	IN
Port Range	7 DOWNTO 0
Operational Clock	CLS[i]_CLK
Special Function	–
Operational Reset	GTM_RESET
Reset Value	–

27.9 TIO Signal Description

27.9.1 TIO Input Selection Unit Signals

Loop	–
Condition	–
Format	–

LUT2OUT	
Description	Signal vector of input selection LUT2 channels c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

LUT3IN	
Description	Internal signal vector of inputs k∈{0, 1,..., 2} to LUT3 instance
Loop	–
Condition	–
Signal Type	INT
Assignment	–

LUT3OUT	
Description	Internal signal of input selection LUT3 instance.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ISEL_QOUT[q]	
Description	Output of input selection unit [q] in channel group g to next/ previous input selection unit.
Loop	$q = \{n : 0 <= n <= 1\}$
Condition	–
Signal Type	ARRAY[NTIO_CH8*2]
Assignment	–

ISEL_QOUT_PREV[q]	
Description	Input signal from previous selection unit to selection unit [q] in group g.
Loop	$q = \{n : 0 <= n <= 1\}$
Condition	–
Signal Type	ARRAY[NTIO_CH8*2]
Assignment	–

ISEL_QOUT_NEXT[q]	
Description	Input signal from next selection unit to selection unit [q] in group g.
Loop	$q = \{n : 0 <= n <= 1\}$
Condition	–
Signal Type	ARRAY[NTIO_CH8*2]
Assignment	–

ISEL_LUT2OUT[q]	
Description	Output signal vector of input selection unit LUT2 signals of channel 0 .. 2 in selection unit [q] in channel group g.
Loop	$q = \{n : 0 <= n <= 1\}$

ISEL_LUT2OUT[q]	
Condition	–
Signal Type	ARRAY[NTIO_CH8*2]
Assignment	–

LUT2OUT_PREV	
Description	Internal signal vector of previous input selection unit LUT2 signals of channel 0 .. 2.
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ISEL_LUT2OUT_PREV[q]	
Description	Input signal vector of previous input selection unit LUT2 signals of channel 0 .. 2 in selection unit [q] in channel group g.
Loop	$q = \{n : 0 <= n <= 1\}$
Condition	–
Signal Type	ARRAY[NTIO_CH8*2]
Assignment	–

27.9.2 TIO Channel Processing Unit Signals

Loop	–
Condition	–
Format	–

SYNC_OUT	
Description	Input signal vector from synchronization stage to input selection unit. c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ISEL_OUT	
Description	Output signal vector of channels of input selection unit c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_OUT	
Description	Output signal vector driven by S registers of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_RE	
Description	Output signal vector driven by S registers rising edge detection of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_FE	
Description	Output signal vector driven by S registers falling edge detection of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CFG_REG_DO_FFS	
Description	S register input signal write vector driven by configuration interface of channel c∈{0, 1,..., 7};x = g*8+c
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CFG_REG_WSTB_FFS	
Description	S register input signal write enable vector driven by configuration interface of channel c∈{0, 1,..., 7};x = g*8+c
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CFG_REG_DI_FFS	
Description	S register out signal vector read by configuration interface of channel c∈{0, 1,..., 7};x = g*8+c
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_OUT	
Description	Output signal vector driven by O registers of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_RE	
Description	Output signal vector driven by O registers rising edge detection of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_FE	
Description	Output signal vector driven by O registers falling edge detection of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CFG_REG_DO_FFO	
Description	O register input signal write vector driven by configuration interface of channel c∈{0, 1,..., 7};x = g*8+c
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CFG_REG_WSTB_FFO	
Description	O register input signal write enable vector driven by configuration interface of channel c∈{0, 1,..., 7};x = g*8+c
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CFG_REG_DI_FFO	
Description	O register out signal vector read by configuration interface of channel c∈{0, 1,..., 7};x = g*8+c
Loop	–
Condition	–
Signal Type	INT
Assignment	–

27.9.3 TIO Channel Group Signals

Loop	–
Condition	–
Format	–

TRIG_OUT	
Description	Output signal vector driven by TRIG_OUT control unit of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TRIG_OUT_PREV	
Description	Input signal vector driven by TRIG_OUT control unit of previous channel c∈{0, 1,..., 6}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

PL_TRIG_OUT	
Description	Output signal vector driven by PL_TRIG_OUT control unit of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

PL_TRIG_OUT_PREV	
Description	Input signal vector driven by PL_TRIG_OUT control unit of previous channel c∈{0, 1,..., 6}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TIO_IRQ	
Description	TIO interrupt output signal vector of IRQ control unit of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

27.9.4 TIO Resolution Selection Signals

Loop	–
Condition	–
Format	–

UPDATE	
Description	Output signal vector driven by resolution selection unit of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

PL_UPDATE	
Description	Output signal vector driven by resolution selection unit of channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

27.9.5 TIO Resource Selection Signals

Loop	$g = \{n : 0 <= n <= \mathrm{NTIO\_CH8} - 1\}$
Condition	–
Format	–

RS_TB0[g]	
Description	Selected timebase 0 of group [g]; signal vector
Loop	–
Condition	–
Signal Type	ARRAY[NTIO_CH8]
Assignment	–

RS_TB1[g]	
Description	Selected timebase 1 of group [g]; signal vector
Loop	–
Condition	–
Signal Type	ARRAY[NTIO_CH8]
Assignment	–

RS_TB1_PREV[g]	
Description	Wired to CCM_TB1 when g = 0. Wired to RS_TB1[g-1] when g != 0; signal vector
Loop	–
Condition	–
Signal Type	ARRAY[NTIO_CH8]
Assignment	–

RS_TB2[g]	
Description	Selected timebase 2 of group [g]; signal vector
Loop	–
Condition	–
Signal Type	ARRAY[NTIO_CH8]
Assignment	–

RS_TB2_PREV[g]	
Description	Wired to CCM_TB2 when g = 0. Wired to RS_TB2[g-1] when g != 0; signal vector
Loop	–
Condition	–
Signal Type	ARRAY[NTIO_CH8]
Assignment	–

SOP_CH1[g]	
Description	SOP of channel 1 from channel group [g].
Loop	–
Condition	–
Signal Type	ARRAY[NTIO_CH8]
Assignment	–

SOP_CH5[g]	
Description	SOP of channel 5 from channel group [g].
Loop	–
Condition	–
Signal Type	ARRAY[NTIO_CH8]
Assignment	–

RS_CLKEN[g]	
Description	Selected clock resolutions for group [g]; signal vector the bit [c:c] represents the clock resolution c, c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	ARRAY[NTIO_CH8]
Assignment	–

27.9.6 TIO Plus Signals

Loop	–
Condition	–
Format	–

S_TRIG_OUT	
Description	Output signal vector driven by instruction TRIG_OUT control unit of TIO plus channel; bit [c:c] represents the channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_TRIG_OUT	
Description	Output signal vector driven by instruction TRIG_OUT control unit of TIO plus channel; bit [c:c] represents the channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CYCLIC_INIT_TRIG	
Description	Output signal vector driven by instruction TRIG_OUT control unit of TIO plus channel; bit [c:c] represents the channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

PL_O_VAL	
Description	Output signal write vector driven by O resource of TIO plus channel; bit [c:c] represents the channel c∈{0, 1, ..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

PL_O_LOAD	
Description	Output signal write enable vector driven by O resource of TIO plus channel; bit [c:c] represents the channel c∈{0, 1,..., 7}
Loop	–
Condition	–

PL_O_LOAD	
Signal Type	INT
Assignment	–

CMP_EVT	
Description	Output signal vector driven by O resource of TIO plus channel; bit [c:c] represents the channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMP_EVT_PREV	
Description	Input signal vector driven by O resource of TIO plus previous channel ; bit [c:c] represents the channel c∈{0, 1, ..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

PL_EVT	
Description	Output signal vector driven by O resource of TIO plus channel; bit [c:c] represents the channel c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMD_ACTIVE_CC[c]	
Description	Output signal vector driven by O resource of TIO plus channel [c]
Loop	$c = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]
Assignment	–

CMD_ACTIVE_CC_PREV[-c]	
Description	Input signal vector driven by O resource of TIO plus channel [c]-1
Loop	$c = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]
Assignment	–

O_ACTION[c]	
Description	Output signal vector driven by O resource of TIO plus channel [c]
Loop	$c = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]

O_ACTION[c]	
Assignment	–

O_ACTION_PREV[c]	
Description	Input signal vector driven by O resource of TIO plus channel [c]-1
Loop	$c = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]
Assignment	–

PL_ODIS_PREV	
Description	Input signal vector driven by O resource of TIO plus previous channel; bit [c:c] represents the channel c∈{0, 1, ..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

FUPD	
Description	Input signal vector driven by TIO[i]_FUPD.CH[x]; bit [x:x] represents the channel x∈ g*8 +c; g∈{0,...,NTIO_CH8-1}; c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

27.9.7 TIO Plus Processsing S / O Resource Signals

Loop	–
Condition	–
Format	–

O_INSTR_PULL	
Description	Output signal driven by PL_S control unit of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_INSTR_PULL_NEXT	
Description	Input signal driven by PL_O control unit of next channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_DATA_PUSH	
Description	Output signal driven by PL_O control unit of channel

O_DATA_PUSH	
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_DATA_PUSH_PREV	
Description	Input signal driven by PL_O control unit of previous channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_BUFF_FILLED	
Description	Output signal driven by PL_O control unit of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_BUFF_FILLED	
Description	Output signal driven by PL_S control unit of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

O_BUFF_FILLED_PREV	
Description	Input signal driven by PL_O control unit of previous channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

OCMD	
Description	Output signal vector driven by command of O resource of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SCMD	
Description	Output signal vector driven by command of S resource of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

OCMD_PREV	
Description	Input signal vector driven by command of O resource of previous channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SCMD_PREV	
Description	Input signal vector driven by command of S resource of previous channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

OOP	
Description	Output signal vector driven by operand of O resource of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

OOP_PREV	
Description	Input signal vector driven by operand of O resource of previous channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SOP_PREV	
Description	Input signal vector driven by operand of S resource of previous channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

SOP	
Description	Output signal vector driven by operand of S resource of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_DATA_PUSH	
Description	Output signal of S resource of channel; used in O resource of local channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_DATA_PUSH_PREV	
Description	Input signal of S resource of local channel; receiving input from signal O_DATA_PUSH of O resource of previous channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_INSTR_PULL	
Description	Output signal of S resource of channel; driving output signal INSTR_PULL of local channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_INSTR_PULL_NEXT	
Description	Input signal of S resource of local channel; receiving input from signal O_INSTR_PULL of O resource of local channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

S_BUFFER_EMPTY	
Description	Local signal of S resource of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

TB	
Description	Local signal vector driven by the time base multiplexer in the O resource of a channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

ITERA_CNT[c]	
Description	Local signal vector of S resource of channel [c] in use while count instruction is active
Loop	$c = \{n : 0 <= n <= 7\}$
Condition	–
Signal Type	ARRAY[8]
Assignment	–

SHIFT_CNT	
Description	Local signal vector of O resource of channel in use while shift instruction is active
Loop	–
Condition	–
Signal Type	INT

SHIFT_CNT	
Assignment	–

O_BUFFER_EMPTY	
Description	Local signal of O resource of channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMD_ACTIVE	
Description	Local signal of S / O resource of channel: indicating an instruction is active
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMD_ACTIVE_COUNT	
Description	Local signal vector of S resource of channel [c] in a channel group. Every bit reflects the respective TIO[i]_G[-g]_CH[c]_SINST_COUNT.CMD_ACTIVE where c∈{0, 1,..., 7}
Loop	–
Condition	–
Signal Type	INT
Assignment	–

INSTR_END	
Description	Local signal of S / O resource of channel: event indicating the termination of a instruction
Loop	–
Condition	–
Signal Type	INT
Assignment	–

INSTR_PULL	
Description	Local signal of S / O resource of channel: event indicating the reload of an instruction
Loop	–
Condition	–
Signal Type	INT
Assignment	–

DATA_PUSH	
Description	Local signal of S / O resource of channel: event indicating that data shall be stored in capture buffer of next S / O resource
Loop	–
Condition	–
Signal Type	INT
Assignment	–

DUALCMP_END	
Description	Local signal of O resource of channel: event indicating the dual compare instruction terminates

DUALCMP_END	
Loop	–
Condition	–
Signal Type	INT
Assignment	–

DUALCMP_END_NEXT	
Description	Input signal of O resource of previous channel: event from the master channel indicating the dual compare instruction terminates
Loop	–
Condition	–
Signal Type	INT
Assignment	–

FSM_DUALCMP_STATE	
Description	Internal signal of O resource; stores the actual state of an active dual compare instruction
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMP_EVT_EN	
Description	Local signal in O resource of TIO plus channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

CMP_EVT_PREV_EN	
Description	Local signal in O resource of TIO plus channel
Loop	–
Condition	–
Signal Type	INT
Assignment	–

27.9.8 TIO Channel Enable

Loop	$j = \{n : 0 <= n <= \mathrm{NCCM} - 1\}; x = \{n : 0 <= n <= \mathrm{NTIO\_CH8} * 8 - 1\}$
Condition	–
Format	–

TIO[j]_CH_EN[x]	
Description	TIO[j] channel [x] enabled signal
Loop	–
Condition	$\mathrm{CTIO}[j] == 1 \&\& \mathrm{TIO\_PLUS} == 1$
Signal Type	ARRAY[NCCM][NTIO_CH8 *8]

TIO[j]_CH_EN[x]	
Assignment	(TIO[j]_PLUS_AVAILABLE * TIO[j]_ENDIS.CH[x])

27.9.9 TIO Reset

Loop	$j = \{n : 0 <= n <= \text{NCCM} - 1\}; g = \{n : 0 <= n <= \text{NTIO\_CH8} - 1\}; c = \{n : 0 <= n <= 7\}$
Condition	–
Format	–

TIO[j]_G[g]_RESET_CH[c]	
Description	TIO[j] group [g] channel [c] reset active signal
Loop	$x = \{n : g * 8 + c <= n <= g * 8 + c\}$
Condition	$\text{CTIO}[j] == 1 \&\& \text{TIO\_PLUS} == 1$
Signal Type	ARRAY[NCCM][NTIO_CH8 *8]
Assignment	(TIO[j]_PLUS_AVAILABLE * TIO[j]_PL_SWRST.CH[x]) == 1

27.9.10 TIO Plus Available

Loop	$j = \{n : 0 <= n <= \text{NCCM} - 1\}$
Condition	–
Format	–

TIO[j]_PLUS_AVAILABLE	
Description	TIO[j] plus functionality available
Loop	–
Condition	–
Signal Type	ARRAY[NCCM]
Assignment	CTIO[j] == 1 && TIO_PLUS == 1

28 Appendix A

28.1 Acronym Definitions

Table 83 Acronym Definition

Abbreviation	Definition / Meaning
ADC	Analog to Digital Converter
AEI	Automotive Electronics Interface (Bosch proprietary bus protocol)
AFD	AEI to FIFO Data Interface
ARU	Advanced Routing Unit
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
ATOM	ARU–connected Timer Output Module
BRC	Broadcast Module
CDC	Clock Domain Crossing
CMP	Output Compare Unit
CMU	Clock Management Unit
DCM	Data Consistency Mode; available within module BRC
DfT	Design for Testability
DMA	Direct Memory Access
DPLL	Digital Phase Locked Loop (Module)
DTM	Dead Time Module
EC	Equivalence Check
F2A	FIFO to ARU Unit
FIFO	First In First Out Module
FLT	Filter Unit in TIM Channel
GTM	Generic Timer Module
GTM_AEI	Bus interface bridge (module which converts multiple AEI bus protocol variants to GTM internal accesses)
GTM_RM	Generic Timer Module Reference Model
ICM	Interrupt Concentrator Module
ICPA	Inter cluster pulse adapter
IP	Intellectual Property
IRC	Interrupt Controller
LBIST	Logic Build In Self Test
MAP	TIM0 Input Mapping Module
MCFG	Memory Configuration
MCS	Multi Channel Sequencer
MON	Monitor Unit
MTM	Maximum Throughput Mode; available within module BRC
PCM	Pulse Count Modulation
PSM	Parameter Storage Module
PWM	Pulse Width Modulation
SPE	Sensor Pattern Evaluation
TBU	Time Base Unit
TDU	Timeout detection unit in TIM channel
TIM	Timer Input Module
TIM_CH	Channel unit in TIM module
TIO	Timer Input Output Module
TOM	Timer Output Module

28.2 GTM Device Configuration Variables

Table 84 GTM Device Configuration Variables, Define Available Module Resources

Device configuration variable	Type	Array index definition	Description
ATOM_HIGH_RES	ARRAY[NATOM] of INT	[i]	Number of available High Resolution channels in ATOM instance of cluster i.
AXIM	INT		Number of AXI master slots in AXI master instance.
CDTM	ARRAY[NCDTM] of LIST	[i]	The i–th array member defines the list of DTM instances available within the cluster i.
CFG_CLOCK_RATE	INT		Configurable clock rate available.
CTIO	ARRAY[NCCM] of INT	[i]	0: no TIO instance available in cluster i. 1: TIO instance implemented in cluster i.
FAST_CLK_CLS	ARRAY[NCCM] of INT	[i]	0: Cluster i only supports clock frequency divided by 2. 1: Cluster i supports with clock frequency divider 1 / 2.
INT_CLK_EN_GEN	INT		GTM internal clock enable signals in use.
MCS_ERM	ARRAY[NMCS] of INT	[i]	0: MSB of MCS RAM1 address not used in MCS module of cluster i. 1: MSB of MCS RAM1 address used in MCS module of cluster i.
MCS_MAW	ARRAY[NMCS] of INT	[i]	Memory address width of RAM 0 in MCS module of cluster i.
MCS_MEM_SIZE	ARRAY[NMCS] of INT	[i]	Initial address space (bytewise addressing) of MCS RAM in MCS of cluster i.
NADC	INT		Number of ADC interfaces.
NARU	INT		Number of ARU modules.
NATOM	INT		Number of ATOM modules.
NAXIM	INT		Number of clusters with AXI master support.
NBRC	INT		Number of BRC modules.
NCCM	INT		Number of CCM modules.
NCDTM	INT		Number of elements in array CDTM.
NCMP	INT		Number of CMP modules.
NDPLL	INT		Number of DPLL modules.
NMAP	INT		Number of MAP modules.
NMCFG	INT		Number of MCFG modules.
NMCS	INT		Number of MCS modules.
NMON	INT		Number of MON modules.
NPSM	INT		Number of PSM modules.
NSPE	INT		Number of SPE modules.
NTIM	INT		Number of TIM modules.
NTIO	INT		Number of TIO modules.
NTIO_CH8	INT		Number of 8 TIO channel resources. Value range 1 to 3
NTOM	INT		Number of TOM modules.
TIO_PLUS	INT		Select TIO basic/plus implementation.

Device configuration variable	Type	Array index definition	Description
TOM_HIGH_RES	ARRAY[NTOM] of INT	[i]	Number of available High Resolution channels in TOM instance of cluster i.
USE_AXIS	INT		GTM bus configuration interface implements an AXI slave instead of an AEI slave.
USE_TBU_CH3	INT		TBU channel 3 is implemented

Table 85 GTM Device Configuration Variables, Defined by SOC Integration

Device configuration variable	Type	Description
AEI_ADDR_PIPELINE_STAGE	INT	AEI address pipeline stage implemented
AEI_RDATA_PIPELINE_STAGE	INT	AEI read data pipeline stage implemented
ATOM_OUT_RST	INT	ATOM_OUT reset level.
ATOM_TRIG_CHAIN	INT	Define ATOM external trigger chain configuration.
ATOM_TRIG_INTCHAIN	INT	Define ATOM internal trigger chain configuration.
AXIM_ID_WIDTH	INT	AXIM ID width
AXIM_PRIV_ACC	INT	AXIM private access is allowed.
AXIM_SEC_ACC	INT	AXIM secure access is allowed.
AXIM_POSTED_WRITE	INT	AXIM posted write access is allowed.
AXIM_DATA_WIDTH	INT	AXI master data bus width.
AXIS_DATA_WIDTH	INT	AXI slave data bus width.
AXIS_DEBUGGER_ID0_ENABLE	INT	AXI Debugger ID comparator enable 0.
AXIS_DEBUGGER_ID0_MASK	INT	AXI Debugger ID compare mask 0.
AXIS_DEBUGGER_ID0_VALUE	INT	AXI Debugger ID compare value 0.
AXIS_DEBUGGER_ID1_ENABLE	INT	AXI Debugger ID comparator enable 1.
AXIS_DEBUGGER_ID1_MASK	INT	AXI Debugger ID compare mask 1.
AXIS_DEBUGGER_ID1_VALUE	INT	AXI Debugger ID compare value 1.
AXIS_ID_WIDTH	INT	AXIS ID width
BRIDGE_BUFF_DPT	INT	Buffer depth of the GTM AEI bridge implementation.
GRSTEN	INT	Global GTM reset register enabled.
IRQ_MODE_RST_VAL	INT	Initial value of IRQ mode.
RESET_ACTIVE	INT	Active level of asynchronous reset
SYNC_INPUT_REG	INT	Additional pipelined stage implemented. All accesses in synchronous.
TIO_OUT_RST	INT	TIO_OUT reset level.
TOM_OUT_RST	INT	TOM_OUT reset level.
TOM_TRIG_CHAIN	INT	Define TOM external trigger chain configuration.
TOM_TRIG_INTCHAIN	INT	Define TOM internal trigger chain configuration.

Table 86 GTM Device Configuration Variables, Fixed or Derived Values

Device configuration variable	Type	Array index definition	Description
ATOM_OUT_N_WITH_DTM	ARRAY[NCDTM][8] of INT	[i][x]	0: Channel x of signal ATOM_OUT_N in cluster i has no DTM implemented. 1: Channel x of signal ATOM_OUT_N in cluster i has a DTM implemented.
CADDR_END_RST_VAL	INT		Reset value of ARU_CADDR END register.

Device configuration variable	Type	Array index definition	Description
DPLL_RR2_MEM_SIZE	INT		Address space (bytewise addressing) of DPLL RAM2.
GTM_REV_DEVICE_CODE	INT		Device encoding digit.
GTM_REV_REL_BASE	INT		GTM release base.
GTM_REV_REL_ITER	INT		GTM release iteration.
GTM_REV_VENDOR_CODE	INT		Vendor encoding digit.
GTM_REV_VER_MAJOR	INT		Define major version number of GTM-IP specification.
GTM_REV_VER_MINOR	INT		Define minor version number of GTM-IP specification.
NARP	INT		Number of Address Range Protectors per CCM.
NHBP	INT		Number of Hardware Break Points.
NOAC	INT		Number of DPLL Actions Modules.
TIO_OUT_N_WITH_DTM	ARRAY[NCDTM][24] of INT	[i][x]	0: Channel x of signal TIO_OUT_N in cluster i has no DTM implemented. 1: Channel x of signal TIO_OUT_N in cluster i has a DTM implemented.
TOM_OUT_N_WITH_DTM	ARRAY[NCDTM][16] of INT	[i][x]	0: Channel x of signal TOM_OUT_N in cluster i has no DTM implemented. 1: Channel x of signal TOM_OUT_N in cluster i has a DTM implemented.

28.3 Conventions

28.3.1 Signals and Ports

Signals are abstract nets with unique names that are used for interconnecting components within a module or across modules.

In GTM-IP, there are three types defined for signals:
1. INT: An integer type that can be viewed as a one-dimensional array of type bit. The word length to represent the integer is left unspecified but can be inferred from the signal description or the expression in the 'signal assignment'. Signals that have the signal assignment defined are 1-bit wide that can be interpreted as Boolean. That is, the value '1' can be interpreted as true and the value '0' can be interpreted as false.
2. ARRAY[N]: A one-dimensional array of INTs of size N.
3. ARRAY[N][M]: A two-dimensional array of INTs of size N*M.

Signals do not necessarily correspond to signals/variables on the RTL level nor to nets in the generated netlist. They are rather abstract means describing interconnectivity of components in a module on a higher level. Every signal has a 'signal description' that provides info about it. The signal 'assignment', when present, provides a more rigorous definition of the signal by means of an expression which may include bit fields, ports, device configuration variables, other signals, and so on.

Since they are of integer type, signals can be used in mathematical functions such as min, max, mod, ceil, floor, and abs.

For a signal *SIG* , the notation *SIG* [v:w] can be used in prose text and figures to refer to a bit vector with range v down to w in the signal.

For a signal *SIG* , the notation bitrange(SIG, v, w) can be used to extract a bit vector from the signal. The extracted bit vector is assumed unsigned.

Ports are abstract nets with unique names that are used for interconnecting modules. They do not necessarily correspond to physical ports on the module level.

A port's special function attribute can have one of the following values:
1. isClock: used to designate clocks provided to the GTM-IP.
2. isReset: used to designate resets provided to the GTM-IP.
3. isClockEnable: used to designate clock enables provided to the GTM-IP.
4. '-': used to designate other ports with no special function.

The following types can be used in the 'port type' attribute:
1. STD_LOGIC: represents a single logical value (0b or 1b).

2. STD_LOGIC_VECTOR: represents a one-dimensional array of STD_LOGIC. The 'Port range' specifies the range of the array (and from which the size can be calculated).

The 'Port direction' attribute indicates the direction of the port. When 'IN', it indicates the data communicated on the port flows into the module. When 'OUT', it indicates that the data on the port flows out of the module.

When specified, the 'Operational clock' attributes indicate which clock is the port synchronous to (with which clock it is sampled).

Syntax-wise, ports and signals are both italicized and capitalized.

28.3.2 Loops and Conditions

A loop specifies a set and a variable which is defined as an element of the set. Here are some examples:
loop: i ∈ {0, 1, ..., 5}; OR equivalently:
loop: i ∈ {n: 0 <= n <= 5}; OR ...

Instead of defining closely related entities manually and one by one, loops provide a mechanism to define them conveniently and concisely. Having a loop in an entity is equivalent to the following. The entity, without the loop, is replicated X times (X being the cardinality of the set). Each copy is associated with a unique element from the set to further customize the attributes and prose text under that copy. The loop variable enclosed in square brackets [i] shall be substituted with the value of the element associated with the copy. An entity in this context may refer to modules (loops defined at the beginning of every chapter), signals, ports, registers, bit fields, among others. The scope of a loop defined in an entity propagates to the sub-entities declared within. For example, a bit field name may use a loop index declared in the hosting register.

It is possible to have more than one set in a loop. For example:
loop: i ∈ {0, 1, ..., 5}; j ∈ {0, 1, 2}
This is equivalent to having 6*3=18 copies of the entity (without the loop). Each of the copies is associated with a unique i and j from their respective sets. Any occurrence of an [i] and [j] in the copy shall be substituted with the corresponding values associated with it.

A condition is an expression that evaluates to a Boolean value. Conditions are used in conjunction with loops to determine if an entity (port, signal, register, bit field) is implemented as detailed shortly.

If the loop and the condition are left empty for a particular entity (in this context: port, signal, register), the entity is implemented. An entity is also implemented when the associated condition evaluates to true and the indices used in forming its name are elements of the set specified by the loop. Otherwise, the entity is not implemented. Device configuration variables (see *28.2 "GTM Device Configuration Variables"*) can be used to specify loops and conditions.

An attempt to access a non-implemented register returns an unsupported address (0b11) on *AEI_STATUS* .

A bit field shall only be implemented when it is explicitly defined and when the hosting register is implemented. In addition to that, the condition associated with the bit field has to evaluate to true (or the condition is left empty) and the indices used in forming its name are elements of the set specified (or the loop is left empty).

28.4 Specific Definitions

Table 87 Specific Definition

Definition	Description
Pulse width	It is a definition for pulse-width modulation (PWM) signal. In a PWM period, pulse width is the time when a signal or system is active (in GTM specification on or off is possible). The signal level in the pulse width can be specified as 1 or 0, e.g. configured by **ATOM[i]_CH[x]_CTRL.SL** in ATOM module.
Duty cycle	It is a definition for pulse-width modulation (PWM) signal. The term duty cycle describes the proportion of pulse time to the total PWM period. In GTM specification, duty cycle can be specified as the ratio of "on" time or "off" time of a PWM signal.

28.5 Register / Bit Field Definitions

Table 88 Register Attributes

Attribute	Description
Clock active condition (clock_active_cond)	A register can be accessed (for writing or reading) only when clock_active_cond evaluates to true. The attribute is expressed as a function possibly involving multiple bit fields and signals.

Attribute	Description
View condition (view_cond)	A physical register may have multiple views (multi_view) each with its own function. The bit fields within each view can be defined differently. A view_cond describes the applicability of a multiview. A multiview is only applicable when its view condition (view_cond) evaluates to true. The expression in view_cond may involve device configuration variables, ports, bit fields, registers, and so forth.

Table 89 Register Bit Field Attributes

Attribute	Description
Reserved	A bit field shall only be implemented when it is explicitly defined and when the hosting register is implemented. In addition to that, the condition associated with the bit field has to evaluate to true (or left empty) and the indices used in forming its name are elements of the set specified by the loop (when not left empty). A bit field shall be identified as reserved when not implemented or when explicitly noted. When reserved, a write access to the bit field shall have no side effect and a read access shall always return the value 0. A non–implemented register shall automatically have all the bit fields within identified as reserved.
Not used	A bit field shall be identified as not used when the name of the bit field is NOT_USED. When not used, the bit field operates as a mere storage element with no other associated functionality. It is allowed for a non used bit field to have an unconventional write behavior such as oneToSet... When considering mutiviews for a particular register, a bit field that is not explicitly defined under a particular view might be physically implemented in the register. Such a bit field shall be implicitly identified as not used in that view – It shall inherit the write behavior, volatility, protect enable condition, and the reset groups from the physically implemented bit field in the register.

Table 90 Register Bit Field Type Attributes

Attribute	Description
Accessibility ('R', 'RW') – Modified Write Value ('clear', 'oneToSet', 'oneToClear', 'oneToToggle', 'modify')	This attribute specifies the accessibility of the data in the bit field. One of the following applies: 'R': A read access shall return the data stored in the bit field. A write access shall leave the bit field unchanged and shall return 'No error' on *AEI_STATUS* (i.e., *AEI_STATUS* = 0b00). 'RW': A read access shall return the value stored in the bit field. A write access affects the contents of the bit field (can also be further specified by modified_write_value). Modified write value (modified_write_value) describes the manipulation of data written to a bit field. If not specified, the value written to the field is the value stored in the field. Otherwise, one of the following values applies: Clear: Any write access clears the bit field (sets all the bits within zero) regardless of the value written. oneToSet: Each write data bit of a one shall set (to one) the corresponding bit in the bit field in a bitwise fashion. A write data bit of a zero shall leave the corresponding bit in the bit field unchanged. oneToClear: Each write data bit of a one shall clear (to zero) the corresponding bit in the bit field in a bitwise fashion. A write data bit of a zero shall leave the corresponding bit in the bit field unchanged. oneToToggle: Each write data bit of a one shall invert the corresponding bit in the bit field in a bitwise fashion. A write data bit of a zero shall leave the corresponding bit in the bit field unchanged. Modify: If the manipulation of the data written cannot be described by any of the above, modify applies. Multithread (defined shortly in the table) is an example of a modified write with the value set to '-modify'.

Attribute	Description
Volatility ('v')	Volatile when true indicates in the case of a write followed by a read, or in the case of two consecutive reads, that the read value of the second transaction can be inconsistent with the first write or read. In general, a bit field which can acquire new values due to internal operations in GTM will have the volatile attribute set to true. A bit field which can acquire new values only by writes (when unprotected or when protect_en_cond evaluates as false), reads, MCS programming sequences, or resets will have the volatile attribute set to false. An unconventional write behavior (e.g., oneToClear) for a bit field, although not guaranteed to return the written value, does not qualify the volatile attribute to true.
Multithread ('m')	A group of bit fields in a particular register shall have their multithread attribute set to true if a subset of their W–Coding (wcoding) items (ci) shall result in leaving the bit field value unmodified for the sake of allowing a thread to modify a particular bit field (or more) and leave the rest in the group, possibly controlled by other threads, unmodified without the need for an atomic read–modify–write operation for the register. The width of bit fields qualified as multithread is 2 bits and the manipulation of the data written has the value 'modify' (see modified write value above). When the write data is 0b01, the stored data is 0b00. When the write data is 0b10, the stored data is 0b11. When the write data is 0b00 or 0b11, the stored data is left unchanged. Reading out a bit field qualified as multithread shall always return 0b00 or 0b11.

Table 91 Register Bit Field Coding Items, Conditions, and Reset Groups

Term	Description
Coding condition (coding_cond)	A coding condition (coding_cond) determines which coding items (ci) are applicable for an access type (read/write). The coding condition along with the coding items are used to define the coding attribute. If the semantic of the coding items of a write access and the corresponding one of a read access is identical, only RW–Coding (coding) attribute might be specified. If the semantic is different and the access type is 'RW', R–Coding (rcoding) attribute and W–Coding (wcoding) attribute might be both specified. If the access type is 'R', only rcoding might be specified. Any coding possibility not listed as a coding item in an R–Coding shall be assumed 'not applicable'. Such a coding possibility may never be read back. Any coding possibility designated as 'prohibited' or not listed as a coding item in RW–Coding (or W–Coding) shall be qualified as prohibited. Such a coding possibility, although can be written and read back, shall result in an undefined circuit behavior and it is forbidden to be programmed by an application.
Reset group (reset_group)	A reset_group contains one or more items each of which comprises a reset condition (reset_cond) and a Boolean attribute named async. The reset_cond is specified in the assignment attribute of a signal. The reset_cond may involve device configuration variables, ports, bit fields, registers, and so forth. When one (or more) of the reset_cond's evaluates to true, the bit field is initialized with the specified initial value (init_val). The initialization is performed synchronous to the clock edge if async is false. The initialization is performed immediately (does not require a clock) when async is true.
Protect enable condition (protect_en_cond)	The attribute provides a mechanism to protect a bit field from an unintended write access. A write access to a bit field is enabled when protect_en_cond evaluates to false. A write access to a bit field does not alter the bit field when protect_en_cond evaluates to true. The attribute is expressed as a function possibly involving multiple bit fields, signals, and *CLS[j]_AEI_ARB_WDATA* [k:k] ($k \in \{0, 1, \ldots, 31\}$). When unspecified, protect_en_cond is assumed false.

Table 92 Register Reset Value

Reset value	Description
0	Logic value is 0 after reset
1	Logic value is 1 after reset

28.6 ARU Write Address Definition

Table 93 ARU Write Address Overview

Name	Address	MCS Write Index
ARU_WRADDR	0x000	
F2A0_WRADDR0	0x001	
F2A0_WRADDR1	0x002	
F2A0_WRADDR2	0x003	
F2A0_WRADDR3	0x004	
F2A0_WRADDR4	0x005	
F2A0_WRADDR5	0x006	
F2A0_WRADDR6	0x007	
F2A0_WRADDR7	0x008	
F2A1_WRADDR0	0x009	
F2A1_WRADDR1	0x00A	
F2A1_WRADDR2	0x00B	
F2A1_WRADDR3	0x00C	
F2A1_WRADDR4	0x00D	
F2A1_WRADDR5	0x00E	
F2A1_WRADDR6	0x00F	
F2A1_WRADDR7	0x010	
F2A2_WRADDR0	0x011	
F2A2_WRADDR1	0x012	
F2A2_WRADDR2	0x013	
F2A2_WRADDR3	0x014	
F2A2_WRADDR4	0x015	
F2A2_WRADDR5	0x016	
F2A2_WRADDR6	0x017	
F2A2_WRADDR7	0x018	
BRC_WRADDR0	0x019	
BRC_WRADDR1	0x01A	
BRC_WRADDR2	0x01B	
BRC_WRADDR3	0x01C	
BRC_WRADDR4	0x01D	
BRC_WRADDR5	0x01E	
BRC_WRADDR6	0x01F	
BRC_WRADDR7	0x020	
BRC_WRADDR8	0x021	
BRC_WRADDR9	0x022	
BRC_WRADDR10	0x023	
BRC_WRADDR11	0x024	
BRC_WRADDR12	0x025	
BRC_WRADDR13	0x026	
BRC_WRADDR14	0x027	
BRC_WRADDR15	0x028	
BRC_WRADDR16	0x029	
BRC_WRADDR17	0x02A	
BRC_WRADDR18	0x02B	
BRC_WRADDR19	0x02C	
BRC_WRADDR20	0x02D	

Name	Address	MCS Write Index
BRC_WRADDR21	0x02E	
DPLL_WRADDR0	0x02F	
DPLL_WRADDR1	0x030	
DPLL_WRADDR2	0x031	
DPLL_WRADDR3	0x032	
DPLL_WRADDR4	0x033	
DPLL_WRADDR5	0x034	
DPLL_WRADDR6	0x035	
DPLL_WRADDR7	0x036	
DPLL_WRADDR8	0x037	
DPLL_WRADDR9	0x038	
DPLL_WRADDR10	0x039	
DPLL_WRADDR11	0x03A	
DPLL_WRADDR12	0x03B	
DPLL_WRADDR13	0x03C	
DPLL_WRADDR14	0x03D	
DPLL_WRADDR15	0x03E	
DPLL_WRADDR16	0x03F	
DPLL_WRADDR17	0x040	
DPLL_WRADDR18	0x041	
DPLL_WRADDR19	0x042	
DPLL_WRADDR20	0x043	
DPLL_WRADDR21	0x044	
DPLL_WRADDR22	0x045	
DPLL_WRADDR23	0x046	
DPLL_WRADDR24	0x047	
DPLL_WRADDR25	0x048	
DPLL_WRADDR26	0x049	
DPLL_WRADDR27	0x04A	
DPLL_WRADDR28	0x04B	
DPLL_WRADDR29	0x04C	
DPLL_WRADDR30	0x04D	
DPLL_WRADDR31	0x04E	
MCS0_WRADDR0	0x04F	0
MCS0_WRADDR1	0x050	1
MCS0_WRADDR2	0x051	2
MCS0_WRADDR3	0x052	3
MCS0_WRADDR4	0x053	4
MCS0_WRADDR5	0x054	5
MCS0_WRADDR6	0x055	6
MCS0_WRADDR7	0x056	7
MCS0_WRADDR8	0x057	8
MCS0_WRADDR9	0x058	9
MCS0_WRADDR10	0x059	10
MCS0_WRADDR11	0x05A	11
MCS0_WRADDR12	0x05B	12
MCS0_WRADDR13	0x05C	13
MCS0_WRADDR14	0x05D	14
MCS0_WRADDR15	0x05E	15

Name	Address	MCS Write Index
MCS0_WRADDR16	0x05F	16
MCS0_WRADDR17	0x060	17
MCS0_WRADDR18	0x061	18
MCS0_WRADDR19	0x062	19
MCS0_WRADDR20	0x063	20
MCS0_WRADDR21	0x064	21
MCS0_WRADDR22	0x065	22
MCS0_WRADDR23	0x066	23
MCS1_WRADDR0	0x067	0
MCS1_WRADDR1	0x068	1
MCS1_WRADDR2	0x069	2
MCS1_WRADDR3	0x06A	3
MCS1_WRADDR4	0x06B	4
MCS1_WRADDR5	0x06C	5
MCS1_WRADDR6	0x06D	6
MCS1_WRADDR7	0x06E	7
MCS1_WRADDR8	0x06F	8
MCS1_WRADDR9	0x070	9
MCS1_WRADDR10	0x071	10
MCS1_WRADDR11	0x072	11
MCS1_WRADDR12	0x073	12
MCS1_WRADDR13	0x074	13
MCS1_WRADDR14	0x075	14
MCS1_WRADDR15	0x076	15
MCS1_WRADDR16	0x077	16
MCS1_WRADDR17	0x078	17
MCS1_WRADDR18	0x079	18
MCS1_WRADDR19	0x07A	19
MCS1_WRADDR20	0x07B	20
MCS1_WRADDR21	0x07C	21
MCS1_WRADDR22	0x07D	22
MCS1_WRADDR23	0x07E	23
MCS2_WRADDR0	0x07F	0
MCS2_WRADDR1	0x080	1
MCS2_WRADDR2	0x081	2
MCS2_WRADDR3	0x082	3
MCS2_WRADDR4	0x083	4
MCS2_WRADDR5	0x084	5
MCS2_WRADDR6	0x085	6
MCS2_WRADDR7	0x086	7
MCS2_WRADDR8	0x087	8
MCS2_WRADDR9	0x088	9
MCS2_WRADDR10	0x089	10
MCS2_WRADDR11	0x08A	11
MCS2_WRADDR12	0x08B	12
MCS2_WRADDR13	0x08C	13
MCS2_WRADDR14	0x08D	14
MCS2_WRADDR15	0x08E	15
MCS2_WRADDR16	0x08F	16

Name	Address	MCS Write Index
MCS2_WRADDR17	0x090	17
MCS2_WRADDR18	0x091	18
MCS2_WRADDR19	0x092	19
MCS2_WRADDR20	0x093	20
MCS2_WRADDR21	0x094	21
MCS2_WRADDR22	0x095	22
MCS2_WRADDR23	0x096	23
MCS3_WRADDR0	0x097	0
MCS3_WRADDR1	0x098	1
MCS3_WRADDR2	0x099	2
MCS3_WRADDR3	0x09A	3
MCS3_WRADDR4	0x09B	4
MCS3_WRADDR5	0x09C	5
MCS3_WRADDR6	0x09D	6
MCS3_WRADDR7	0x09E	7
MCS3_WRADDR8	0x09F	8
MCS3_WRADDR9	0x0A0	9
MCS3_WRADDR10	0x0A1	10
MCS3_WRADDR11	0x0A2	11
MCS3_WRADDR12	0x0A3	12
MCS3_WRADDR13	0x0A4	13
MCS3_WRADDR14	0x0A5	14
MCS3_WRADDR15	0x0A6	15
MCS3_WRADDR16	0x0A7	16
MCS3_WRADDR17	0x0A8	17
MCS3_WRADDR18	0x0A9	18
MCS3_WRADDR19	0x0AA	19
MCS3_WRADDR20	0x0AB	20
MCS3_WRADDR21	0x0AC	21
MCS3_WRADDR22	0x0AD	22
MCS3_WRADDR23	0x0AE	23
MCS4_WRADDR0	0x0AF	0
MCS4_WRADDR1	0x0B0	1
MCS4_WRADDR2	0x0B1	2
MCS4_WRADDR3	0x0B2	3
MCS4_WRADDR4	0x0B3	4
MCS4_WRADDR5	0x0B4	5
MCS4_WRADDR6	0x0B5	6
MCS4_WRADDR7	0x0B6	7
MCS4_WRADDR8	0x0B7	8
MCS4_WRADDR9	0x0B8	9
MCS4_WRADDR10	0x0B9	10
MCS4_WRADDR11	0x0BA	11
MCS4_WRADDR12	0x0BB	12
MCS4_WRADDR13	0x0BC	13
MCS4_WRADDR14	0x0BD	14
MCS4_WRADDR15	0x0BE	15
MCS4_WRADDR16	0x0BF	16
MCS4_WRADDR17	0x0C0	17

Name	Address	MCS Write Index
MCS4_WRADDR18	0x0C1	18
MCS4_WRADDR19	0x0C2	19
MCS4_WRADDR20	0x0C3	20
MCS4_WRADDR21	0x0C4	21
MCS4_WRADDR22	0x0C5	22
MCS4_WRADDR23	0x0C6	23
MCS5_WRADDR0	0x0C7	0
MCS5_WRADDR1	0x0C8	1
MCS5_WRADDR2	0x0C9	2
MCS5_WRADDR3	0x0CA	3
MCS5_WRADDR4	0x0CB	4
MCS5_WRADDR5	0x0CC	5
MCS5_WRADDR6	0x0CD	6
MCS5_WRADDR7	0x0CE	7
MCS5_WRADDR8	0x0CF	8
MCS5_WRADDR9	0x0D0	9
MCS5_WRADDR10	0x0D1	10
MCS5_WRADDR11	0x0D2	11
MCS5_WRADDR12	0x0D3	12
MCS5_WRADDR13	0x0D4	13
MCS5_WRADDR14	0x0D5	14
MCS5_WRADDR15	0x0D6	15
MCS5_WRADDR16	0x0D7	16
MCS5_WRADDR17	0x0D8	17
MCS5_WRADDR18	0x0D9	18
MCS5_WRADDR19	0x0DA	19
MCS5_WRADDR20	0x0DB	20
MCS5_WRADDR21	0x0DC	21
MCS5_WRADDR22	0x0DD	22
MCS5_WRADDR23	0x0DE	23
MCS6_WRADDR0	0x0DF	0
MCS6_WRADDR1	0x0E0	1
MCS6_WRADDR2	0x0E1	2
MCS6_WRADDR3	0x0E2	3
MCS6_WRADDR4	0x0E3	4
MCS6_WRADDR5	0x0E4	5
MCS6_WRADDR6	0x0E5	6
MCS6_WRADDR7	0x0E6	7
MCS6_WRADDR8	0x0E7	8
MCS6_WRADDR9	0x0E8	9
MCS6_WRADDR10	0x0E9	10
MCS6_WRADDR11	0x0EA	11
MCS6_WRADDR12	0x0EB	12
MCS6_WRADDR13	0x0EC	13
MCS6_WRADDR14	0x0ED	14
MCS6_WRADDR15	0x0EE	15
MCS6_WRADDR16	0x0EF	16
MCS6_WRADDR17	0x0F0	17
MCS6_WRADDR18	0x0F1	18

Name	Address	MCS Write Index
MCS6_WRADDR19	0x0F2	19
MCS6_WRADDR20	0x0F3	20
MCS6_WRADDR21	0x0F4	21
MCS6_WRADDR22	0x0F5	22
MCS6_WRADDR23	0x0F6	23
MCS7_WRADDR0	0x0F7	0
MCS7_WRADDR1	0x0F8	1
MCS7_WRADDR2	0x0F9	2
MCS7_WRADDR3	0x0FA	3
MCS7_WRADDR4	0x0FB	4
MCS7_WRADDR5	0x0FC	5
MCS7_WRADDR6	0x0FD	6
MCS7_WRADDR7	0x0FE	7
MCS7_WRADDR8	0x0FF	8
MCS7_WRADDR9	0x100	9
MCS7_WRADDR10	0x101	10
MCS7_WRADDR11	0x102	11
MCS7_WRADDR12	0x103	12
MCS7_WRADDR13	0x104	13
MCS7_WRADDR14	0x105	14
MCS7_WRADDR15	0x106	15
MCS7_WRADDR16	0x107	16
MCS7_WRADDR17	0x108	17
MCS7_WRADDR18	0x109	18
MCS7_WRADDR19	0x10A	19
MCS7_WRADDR20	0x10B	20
MCS7_WRADDR21	0x10C	21
MCS7_WRADDR22	0x10D	22
MCS7_WRADDR23	0x10E	23
MCS8_WRADDR0	0x10F	0
MCS8_WRADDR1	0x110	1
MCS8_WRADDR2	0x111	2
MCS8_WRADDR3	0x112	3
MCS8_WRADDR4	0x113	4
MCS8_WRADDR5	0x114	5
MCS8_WRADDR6	0x115	6
MCS8_WRADDR7	0x116	7
MCS8_WRADDR8	0x117	8
MCS8_WRADDR9	0x118	9
MCS8_WRADDR10	0x119	10
MCS8_WRADDR11	0x11A	11
MCS8_WRADDR12	0x11B	12
MCS8_WRADDR13	0x11C	13
MCS8_WRADDR14	0x11D	14
MCS8_WRADDR15	0x11E	15
MCS8_WRADDR16	0x11F	16
MCS8_WRADDR17	0x120	17
MCS8_WRADDR18	0x121	18
MCS8_WRADDR19	0x122	19

Name	Address	MCS Write Index
MCS8_WRADDR20	0x123	20
MCS8_WRADDR21	0x124	21
MCS8_WRADDR22	0x125	22
MCS8_WRADDR23	0x126	23
MCS9_WRADDR0	0x127	0
MCS9_WRADDR1	0x128	1
MCS9_WRADDR2	0x129	2
MCS9_WRADDR3	0x12A	3
MCS9_WRADDR4	0x12B	4
MCS9_WRADDR5	0x12C	5
MCS9_WRADDR6	0x12D	6
MCS9_WRADDR7	0x12E	7
MCS9_WRADDR8	0x12F	8
MCS9_WRADDR9	0x130	9
MCS9_WRADDR10	0x131	10
MCS9_WRADDR11	0x132	11
MCS9_WRADDR12	0x133	12
MCS9_WRADDR13	0x134	13
MCS9_WRADDR14	0x135	14
MCS9_WRADDR15	0x136	15
MCS9_WRADDR16	0x137	16
MCS9_WRADDR17	0x138	17
MCS9_WRADDR18	0x139	18
MCS9_WRADDR19	0x13A	19
MCS9_WRADDR20	0x13B	20
MCS9_WRADDR21	0x13C	21
MCS9_WRADDR22	0x13D	22
MCS9_WRADDR23	0x13E	23
UNUSED_13F	0x13F	
UNUSED_140	0x140	
UNUSED_141	0x141	
UNUSED_142	0x142	
UNUSED_143	0x143	
UNUSED_144	0x144	
UNUSED_145	0x145	
UNUSED_146	0x146	
UNUSED_147	0x147	
UNUSED_148	0x148	
UNUSED_149	0x149	
UNUSED_14A	0x14A	
UNUSED_14B	0x14B	
UNUSED_14C	0x14C	
UNUSED_14D	0x14D	
UNUSED_14E	0x14E	
UNUSED_14F	0x14F	
UNUSED_150	0x150	
UNUSED_151	0x151	
UNUSED_152	0x152	
UNUSED_153	0x153	

Name	Address	MCS Write Index
UNUSED_154	0x154	
UNUSED_155	0x155	
UNUSED_156	0x156	
ATOM0_WRADDR0	0x157	
ATOM0_WRADDR1	0x158	
ATOM0_WRADDR2	0x159	
ATOM0_WRADDR3	0x15A	
ATOM0_WRADDR4	0x15B	
ATOM0_WRADDR5	0x15C	
ATOM0_WRADDR6	0x15D	
ATOM0_WRADDR7	0x15E	
ATOM1_WRADDR0	0x15F	
ATOM1_WRADDR1	0x160	
ATOM1_WRADDR2	0x161	
ATOM1_WRADDR3	0x162	
ATOM1_WRADDR4	0x163	
ATOM1_WRADDR5	0x164	
ATOM1_WRADDR6	0x165	
ATOM1_WRADDR7	0x166	
ATOM2_WRADDR0	0x167	
ATOM2_WRADDR1	0x168	
ATOM2_WRADDR2	0x169	
ATOM2_WRADDR3	0x16A	
ATOM2_WRADDR4	0x16B	
ATOM2_WRADDR5	0x16C	
ATOM2_WRADDR6	0x16D	
ATOM2_WRADDR7	0x16E	
ATOM3_WRADDR0	0x16F	
ATOM3_WRADDR1	0x170	
ATOM3_WRADDR2	0x171	
ATOM3_WRADDR3	0x172	
ATOM3_WRADDR4	0x173	
ATOM3_WRADDR5	0x174	
ATOM3_WRADDR6	0x175	
ATOM3_WRADDR7	0x176	
ATOM4_WRADDR0	0x177	
ATOM4_WRADDR1	0x178	
ATOM4_WRADDR2	0x179	
ATOM4_WRADDR3	0x17A	
ATOM4_WRADDR4	0x17B	
ATOM4_WRADDR5	0x17C	
ATOM4_WRADDR6	0x17D	
ATOM4_WRADDR7	0x17E	
ATOM5_WRADDR0	0x17F	
ATOM5_WRADDR1	0x180	
ATOM5_WRADDR2	0x181	
ATOM5_WRADDR3	0x182	
ATOM5_WRADDR4	0x183	
ATOM5_WRADDR5	0x184	

Name	Address	MCS Write Index
ATOM5_WRADDR6	0x185	
ATOM5_WRADDR7	0x186	
ATOM6_WRADDR0	0x187	
ATOM6_WRADDR1	0x188	
ATOM6_WRADDR2	0x189	
ATOM6_WRADDR3	0x18A	
ATOM6_WRADDR4	0x18B	
ATOM6_WRADDR5	0x18C	
ATOM6_WRADDR6	0x18D	
ATOM6_WRADDR7	0x18E	
ATOM7_WRADDR0	0x18F	
ATOM7_WRADDR1	0x190	
ATOM7_WRADDR2	0x191	
ATOM7_WRADDR3	0x192	
ATOM7_WRADDR4	0x193	
ATOM7_WRADDR5	0x194	
ATOM7_WRADDR6	0x195	
ATOM7_WRADDR7	0x196	
ATOM8_WRADDR0	0x197	
ATOM8_WRADDR1	0x198	
ATOM8_WRADDR2	0x199	
ATOM8_WRADDR3	0x19A	
ATOM8_WRADDR4	0x19B	
ATOM8_WRADDR5	0x19C	
ATOM8_WRADDR6	0x19D	
ATOM8_WRADDR7	0x19E	
ATOM9_WRADDR0	0x19F	
ATOM9_WRADDR1	0x1A0	
ATOM9_WRADDR2	0x1A1	
ATOM9_WRADDR3	0x1A2	
ATOM9_WRADDR4	0x1A3	
ATOM9_WRADDR5	0x1A4	
ATOM9_WRADDR6	0x1A5	
ATOM9_WRADDR7	0x1A6	
ATOM10_WRADDR0	0x1A7	
ATOM10_WRADDR1	0x1A8	
ATOM10_WRADDR2	0x1A9	
ATOM10_WRADDR3	0x1AA	
ATOM10_WRADDR4	0x1AB	
ATOM10_WRADDR5	0x1AC	
ATOM10_WRADDR6	0x1AD	
ATOM10_WRADDR7	0x1AE	
ATOM11_WRADDR0	0x1AF	
ATOM11_WRADDR1	0x1B0	
ATOM11_WRADDR2	0x1B1	
ATOM11_WRADDR3	0x1B2	
ATOM11_WRADDR4	0x1B3	
ATOM11_WRADDR5	0x1B4	
ATOM11_WRADDR6	0x1B5	

Name	Address	MCS Write Index
ATOM11_WRADDR7	0x1B6	
TIM0_WRADDR0	0x1B7	
TIM0_WRADDR1	0x1B8	
TIM0_WRADDR2	0x1B9	
TIM0_WRADDR3	0x1BA	
TIM0_WRADDR4	0x1BB	
TIM0_WRADDR5	0x1BC	
TIM0_WRADDR6	0x1BD	
TIM0_WRADDR7	0x1BE	
TIM1_WRADDR0	0x1BF	
TIM1_WRADDR1	0x1C0	
TIM1_WRADDR2	0x1C1	
TIM1_WRADDR3	0x1C2	
TIM1_WRADDR4	0x1C3	
TIM1_WRADDR5	0x1C4	
TIM1_WRADDR6	0x1C5	
TIM1_WRADDR7	0x1C6	
TIM2_WRADDR0	0x1C7	
TIM2_WRADDR1	0x1C8	
TIM2_WRADDR2	0x1C9	
TIM2_WRADDR3	0x1CA	
TIM2_WRADDR4	0x1CB	
TIM2_WRADDR5	0x1CC	
TIM2_WRADDR6	0x1CD	
TIM2_WRADDR7	0x1CE	
TIM3_WRADDR0	0x1CF	
TIM3_WRADDR1	0x1D0	
TIM3_WRADDR2	0x1D1	
TIM3_WRADDR3	0x1D2	
TIM3_WRADDR4	0x1D3	
TIM3_WRADDR5	0x1D4	
TIM3_WRADDR6	0x1D5	
TIM3_WRADDR7	0x1D6	
TIM4_WRADDR0	0x1D7	
TIM4_WRADDR1	0x1D8	
TIM4_WRADDR2	0x1D9	
TIM4_WRADDR3	0x1DA	
TIM4_WRADDR4	0x1DB	
TIM4_WRADDR5	0x1DC	
TIM4_WRADDR6	0x1DD	
TIM4_WRADDR7	0x1DE	
TIM5_WRADDR0	0x1DF	
TIM5_WRADDR1	0x1E0	
TIM5_WRADDR2	0x1E1	
TIM5_WRADDR3	0x1E2	
TIM5_WRADDR4	0x1E3	
TIM5_WRADDR5	0x1E4	
TIM5_WRADDR6	0x1E5	
TIM5_WRADDR7	0x1E6	

Name	Address	MCS Write Index
TIM6_WRADDR0	0x1E7	
TIM6_WRADDR1	0x1E8	
TIM6_WRADDR2	0x1E9	
TIM6_WRADDR3	0x1EA	
TIM6_WRADDR4	0x1EB	
TIM6_WRADDR5	0x1EC	
TIM6_WRADDR6	0x1ED	
TIM6_WRADDR7	0x1EE	
TIM7_WRADDR0	0x1EF	
TIM7_WRADDR1	0x1F0	
TIM7_WRADDR2	0x1F1	
TIM7_WRADDR3	0x1F2	
TIM7_WRADDR4	0x1F3	
TIM7_WRADDR5	0x1F4	
TIM7_WRADDR6	0x1F5	
TIM7_WRADDR7	0x1F6	
UNUSED_1F7	0x1F7	
UNUSED_1F8	0x1F8	
UNUSED_1F9	0x1F9	
UNUSED_1FA	0x1FA	
UNUSED_1FB	0x1FB	
UNUSED_1FC	0x1FC	
UNUSED_1FD	0x1FD	
ARU_EMPTY_ADDR	0x1FE	
ARU_FULL_ADDR	0x1FF	

28.7 ARU Master ID Definition

Table 94 ARU Master ID

ARU master ID (dec)	ARU interface ARU_0	ARU interface ARU_1
0	reserved	reserved
1	ARU_0	ARU_1
2	BRC channel 0	DPLL action 0
3	PSM0 channel 0	PSM1 channel 0
4	BRC channel 1	DPLL action 1
5	PSM0 channel 1	PSM1 channel 1
6	BRC channel 2	DPLL action 2
7	PSM0 channel 2	PSM1 channel 2
8	BRC channel 3	DPLL action 3
9	PSM0 channel 3	PSM1 channel 3
10	BRC channel 4	DPLL action 4
11	PSM0 channel 4	PSM1 channel 4
12	BRC channel 5	DPLL action 5
13	PSM0 channel 5	PSM1 channel 5
14	BRC channel 6	DPLL action 6
15	PSM0 channel 6	PSM1 channel 6
16	BRC channel 7	DPLL action 7
17	PSM0 channel 7	PSM1 channel 7
18	BRC channel 8	DPLL action 8

ARU master ID (dec)	ARU interface ARU_0	ARU interface ARU_1
19	MCS0 channel 0	MCS1 channel 0
20	BRC channel 9	DPLL action 9
21	MCS0 channel 1	MCS1 channel 1
22	BRC channel 10	DPLL action 10
23	MCS0 channel 2	MCS1 channel 2
24	BRC channel 11	DPLL action 11
25	MCS0 channel 3	MCS1 channel 3
26	ATOM0 channel 0	DPLL action 12
27	MCS0 channel 4	MCS1 channel 4
28	ATOM0 channel 1	DPLL action 13
29	MCS0 channel 5	MCS1 channel 5
30	ATOM0 channel 2	DPLL action 14
31	MCS0 channel 6	MCS1 channel 6
32	ATOM0 channel 3	DPLL action 15
33	MCS0 channel 7	MCS1 channel 7
34	ATOM0 channel 4	DPLL action 16
35	MCS2 channel 0	ATOM1 channel 0
36	ATOM0 channel 5	DPLL action 17
37	MCS2 channel 1	ATOM1 channel 1
38	ATOM0 channel 6	DPLL action 18
39	MCS2 channel 2	ATOM1 channel 2
40	ATOM0 channel 7	DPLL action 19
41	MCS2 channel 3	ATOM1 channel 3
42	ATOM2 channel 0	DPLL action 20
43	MCS2 channel 4	ATOM1 channel 4
44	ATOM2 channel 1	DPLL action 21
45	MCS2 channel 5	ATOM1 channel 5
46	ATOM2 channel 2	DPLL action 22
47	MCS2 channel 6	ATOM1 channel 6
48	ATOM2 channel 3	DPLL action 23
49	MCS2 channel 7	ATOM1 channel 7
50	ATOM2 channel 4	DPLL action 24
51	MCS4 channel 0	MCS3 channel 0
52	ATOM2 channel 5	DPLL action 25
53	MCS4 channel 1	MCS3 channel 1
54	ATOM2 channel 6	DPLL action 26
55	MCS4 channel 2	MCS3 channel 2
56	ATOM2 channel 7	DPLL action 27
57	MCS4 channel 3	MCS3 channel 3
58	ATOM4 channel 0	DPLL action 28
59	MCS4 channel 4	MCS3 channel 4
60	ATOM4 channel 1	DPLL action 29
61	MCS4 channel 5	MCS3 channel 5
62	ATOM4 channel 2	DPLL action 30
63	MCS4 channel 6	MCS3 channel 6
64	ATOM4 channel 3	DPLL action 31
65	MCS4 channel 7	MCS3 channel 7
66	ATOM4 channel 4	ATOM3 channel 0
67	MCS6 channel 0	MCS5 channel 0

ARU master ID (dec)	ARU interface ARU_0	ARU interface ARU_1
68	ATOM4 channel 5	ATOM3 channel 1
69	MCS6 channel 1	MCS5 channel 1
70	ATOM4 channel 6	ATOM3 channel 2
71	MCS6 channel 2	MCS5 channel 2
72	ATOM4 channel 7	ATOM3 channel 3
73	MCS6 channel 3	MCS5 channel 3
74	ATOM6 channel 0	ATOM3 channel 4
75	MCS6 channel 4	MCS5 channel 4
76	ATOM6 channel 1	ATOM3 channel 5
77	MCS6 channel 5	MCS5 channel 5
78	ATOM6 channel 2	ATOM3 channel 6
79	MCS6 channel 6	MCS5 channel 6
80	ATOM6 channel 3	ATOM3 channel 7
81	MCS6 channel 7	MCS5 channel 7
82	ATOM6 channel 4	ATOM5 channel 0
83	ATOM8 channel 0	ATOM7 channel 0
84	ATOM6 channel 5	ATOM5 channel 1
85	ATOM8 channel 1	ATOM7 channel 1
86	ATOM6 channel 6	ATOM5 channel 2
87	ATOM8 channel 2	ATOM7 channel 2
88	ATOM6 channel 7	ATOM5 channel 3
89	ATOM8 channel 3	ATOM7 channel 3
90	MCS8 channel 0	ATOM5 channel 4
91	ATOM8 channel 4	ATOM7 channel 4
92	MCS8 channel 1	ATOM5 channel 5
93	ATOM8 channel 5	ATOM7 channel 5
94	MCS8 channel 2	ATOM5 channel 6
95	ATOM8 channel 6	ATOM7 channel 6
96	MCS8 channel 3	ATOM5 channel 7
97	ATOM8 channel 7	ATOM7 channel 7
98	MCS8 channel 4	MCS7 channel 0
99	ATOM10 channel 0	ATOM9 channel 0
100	MCS8 channel 5	MCS7 channel 1
101	ATOM10 channel 1	ATOM9 channel 1
102	MCS8 channel 6	MCS7 channel 2
103	ATOM10 channel 2	ATOM9 channel 2
104	MCS8 channel 7	MCS7 channel 3
105	ATOM10 channel 3	ATOM9 channel 3
106	PSM2 channel 0	MCS7 channel 4
107	ATOM10 channel 4	ATOM9 channel 4
108	PSM2 channel 1	MCS7 channel 5
109	ATOM10 channel 5	ATOM9 channel 5
110	PSM2 channel 2	MCS7 channel 6
111	ATOM10 channel 6	ATOM9 channel 6
112	PSM2 channel 3	MCS9 channel 0
113	ATOM11 channel 0	MCS7 channel 7
114	ATOM10 channel 7	MCS9 channel 1
115	ATOM11 channel 1	ATOM9 channel 7
116	PSM2 channel 4	MCS9 channel 2

ARU master ID (dec)	ARU interface ARU_0	ARU interface ARU_1
117	ATOM11 channel 2	unused
118	PSM2 channel 5	MCS9 channel 3
119	ATOM11 channel 3	unused
120	PSM2 channel 6	MCS9 channel 4
121	ATOM11 channel 4	unused
122	PSM2 channel 7	MCS9 channel 5
123	ATOM11 channel 5	unused
124	unused	MCS9 channel 6
125	ATOM11 channel 6	unused
126	unused	MCS9 channel 7
127	ATOM11 channel 7	unused

28.8 GTM Application Constraints

Table 95 Application Constraints for GTM-IP 4100

Nr	Module	Description	Required value	Effect, when not considered
1	DPLL	Increment duration (time between two valid inputs of the DPLL: *TRIGGER* / *STATE*)	> 23.4 µs	The calculation of at least 3 actions per increment in addition to a proper micro-tick generation (*SUB_INC1* / *SUB_INC2*) could not be ensured for a system clock frequency of 100 MHz
2	DPLL	**TBU_CH0_BASE** difference for two successive valid input events of the DPLL	>256 (for the chosen input resolution)	Overflow in the calculation of the reciprocal values; the increment prediction and all other computations of the DPLL would become wrong
3	DPLL	*CMU_CLK_RES* [0:0] frequency	<= half the frequency of cluster 0 clock	The fast pulse correction in automatic end mode could be wrong
4	MCS	Worst Case Execution Time of an N-cycle instruction.	<= 9*N of correspondent cluster clock periods	MCS program execution may be too long for the application's requirements.
5	MCS	If MCS scheduling is configured in Round Robin mode, the MCS features deterministic program execution (the program of MCS-channel x does not modify execution time of MCS-channel y). This determinism no longer exists, if an MCS-channel of MCS instance 0 is accessing the DPLL RAM with its bus master interface.		No channel independent execution time.
6	TOM	If channel is triggered by preceding channel via *TOM_CH_TRIGOUT* [x-1:x--1] signal, the selected *SEL_FXCMU_CLKEN* / *CCM[i]_FXCLK_RES* of both channels has to be the same	Identical *SEL_FXCMU_CLKEN* / *CCM[i]_FXCLK_RES*	The trigger of preceding channel may be lost

Nr	Module	Description	Required value	Effect, when not considered
7	ATOM	If channel is triggered by preceding channel via *ATOM_CH_TRIGOUT* [x--1:x−1] signal, the selected *SEL_CMU_CLKEN* / *CCM[i]_CLK_RES* of both channels has to be the same	Identical *SEL_CMU_CLKEN* / *CCM[i]_CLK_RES*	The trigger of preceding channel may be lost
8	TIM	If a TIM channel x uses ARU transfer with enabled TDU, the minimal time TIN between two subsequent measurement cycles has to be greater than the maximum time TSAMPLE between two subsequent ARU read request events on TIM channel x.	T_{IN} > T_{SAMPLE} (FIFO: T_{SAMPLE} = worst case ARU round trip time, MCS: T_{SAMPLE} = max. time between two (N)ARD(I))	The ARU destination of TIM channel x cannot distinguish between a measurement cycle overflow and a timeout with subsequent valid measurement cycle.
9	ARU	The ARU round trip time in pure Round Robin mode is device specific.	(**ARU_CADDR_END** +1) times slow clock periods with **ARU_CADDR_END** = 126 (default value)	If dynamic mode is activated with **ARU_[g]_DYN_ROUTE_HIGH.DYN_CLK_WAIT** > 0 the ARU round trip time becomes longer and the value is undefined.

28.9 GTM Internal Functional Dependencies

Table 96 Dependency of ATOM Functions

ATOM[i] depends on	Valid index	Description	Type
GTM inputs	i < NATOM	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA
Module ARU	i < NATOM * NARU	ARU 53−bit data; ARU cluster disable	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use
Module ATOM[0]	i == 0 && NATOM > 1	*ATOM_TRIGOUT_DEL* , *ATOM_HRES_OUT_DEL* signals	Cluster internal
Module ATOM[i−1]	i > 1 && i < min(NATOM+1)	*ATOM_TRIGOUT* signals	Cluster external uses ICPA
Module ATOM[i−1]	i > 1 && i < min(NATOM+1)	*ATOM_HRES_OUT* signals	Cluster external, no ICPA in use
Module CCM[i]	i < NATOM	*CCM[i]_CLK_RES* , *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , *CCM[i]_TBU_TS2* signals	Cluster internal
Module MCS[i]	i < min(NATOM, NMCS)	IRQ clear signals	Cluster internal
Module MCS[i−1]	i < min(NATOM, NMCS+1)	IRQ clear signals	Cluster external uses ICPA
Module TIM[i]	i < min(NTIM, NATOM)	*TIM_EXT_CAPTURE* signals	Cluster internal

Table 97 Dependency of ARCH Functions

ARCH depends on	Valid index	Description	Type
GTM inputs	i == 0	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA

Table 98 Dependency of ARU Functions

ARU depends on	Valid index	Description	Type
Module ARCH	i == 0	*CLS_DIS* [NCCM−1:0] signals	Cluster internal
GTM inputs	i == 0	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA
Module ATOM[i]	i < NATOM * NARU	ARU 53−bit data;	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use

ARU depends on	Valid index	Description	Type
Module BRC	NBRC * NARU == 1	ARU 53-bit data;	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use
Module CMU	NARU == 1	ARU_CADDR reset signal	Cluster internal
Module DPLL	NBRC * NARU == 1	ARU 53-bit data;	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use
Module PSM[i]	i < NPSM * NARU	ARU 53-bit data;	Cluster external, no ICPA in use; pulse synchronization done internal to ATOM module based on selected *CCM[i]_CLK_RES*
Module MCS[i]	i < NMCS * NARU	ARU 53-bit data;	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use
Module TIM[i]	i < NTIM * NARU	ARU 53-bit data;	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use

Table 99 Dependency of AXIM Functions

AXIM[i] depends on	Valid index	Description	Type
Module MCS[i]	i < min(NAXIM, NMCS)	IRQ clear signals	Cluster internal

Table 100 Dependency of BRC Functions

BRC depends on	Valid index	Description	Type
GTM inputs	NBRC == 1	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA
Module ARU	NBRC * NARU == 1	ARU 53-bit data; ARU cluster disable	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use

Table 101 Dependency of CCM Functions

CCM[i] depends on	Valid index	Description	Type
Module CMU	i < NCCM	*CMU_CLK_RES* , *CMU_FXCLK_RES* signals	Cluster external uses ICPA
Module DPLL	i == 0 && NDPLL == 1	*DIR1* , *DIR2* signals	Cluster internal
Module TBU	i < NCCM	*TBU_TS0_BASE* , *TBU_TS1_BASE* , *TBU_TS2_BASE* , *TBU_TS3_BASE* signals	Cluster internal
Module TBU	i < NCCM	*TBU_UP0* , *TBU_UP1* , *TBU_UP2* signals	Cluster internal
Module CDTM[i]_DTM[d]	i < NCCM && d ∈ {0, .. 11}	*DTM_OUT0* , *DTM_OUT1* signals	Cluster internal
Module TIM[i]	i < NTIM	*TIM_EXT_CAPTURE* signals	Cluster internal

Table 102 Dependency of CMP Functions

CMP depends on	Valid index	Description	Type
GTM inputs	NCMP == 1	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA
Module CDTM[i]_DTM[d]	NCMP == 1 && i < min(NATOM, 3) && d ∈ {4, 5}	*DTM_OUT0* signals	Cluster external, no ICPA in use; signals typically are not single cluster clock pulses

CMP depends on	Valid index	Description	Type
Module CDTM[i]_DTM[d]	NCMP == 1 && ((NTOM>1 && d ∈ {0, .. 3}) || (NTOM>2 && d ∈ {0, .. 1}))	*DTM_OUT0* signals	Cluster external, no ICPA in use; signals typically are not single cluster clock pulses

Table 103 Dependency of DPLL Functions

DPLL depends on	Valid index	Description	Type
GTM inputs	NDPLL == 1	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA
Module ARU	NDPLL * NARU == 1	ARU 53-bit data; ARU cluster disable	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use
Module CCM[0]	NDPLL > 0	*CCM[0]_CLK_RES* [0:0], *CCM[0]_TBU_TS0* , *CCM[0]_TBU_TS1* , *CCM[0]_TBU_TS2* signals	Cluster internal
Module MAP	NMAP > 0 && NDPLL > 0	*TRIGGER* , *STATE* , *T_DIR* , *S_DIR* , *T_VALID* , *S_VALID* signals	Cluster internal
Module MCS[0]	NDPLL > 0 && NMCS > 0	IRQ clear signals	Cluster internal
Module TBU	NDPLL > 0	*LOW_RES* , *TS_CLK* signals	Cluster internal

Table 104 Dependency of DTM Functions

CDTM[i]_DTM[d] depends on	Valid index	Description	Type
Module ATOM[i]	i < NATOM && d > 3 && d < 6 && d ∈ CDTM[i]	*ATOM_OUT* , *ATOM_OUT_T* , *ATOM_OUT_HRES[x]* , *ATOM_OUT_T_HRES[x]* signals; x ∈ { 0, .. 7}	Cluster internal
Module CCM[i]	i < NCCM && d > 3 && d ∈ CDTM[i]	*CCM[i]_CLK_RES* signals	Cluster internal
Module CCM[i]	i < NCCM && d < 4 && d ∈ CDTM[i]	*CCM[i]_FXCLK_RES* , *CCM[i]_CLK_RES* [8:8] signals	Cluster internal
Module CDTM[i]_DTM[d-1]	d ∈ {1, 2, 3, 5, 7 .. 11} && i < NCCM && d ∈ CDTM[i]	*DTM_PSU_OUT* , *DTM_IN0_FEDGE* , *DTM_IN0_REDGE* signals	Cluster internal
Module CDTM[i]_DTM[d-1]	d ∈ {1, 3, 5, 7, 9, 11} && i < NCCM && d ∈ CDTM[i]	*DTM_COUT3_O* signals	Cluster internal
Module CDTM[i-1]_DTM[e]	i > 1 && e ∈ {3, 5} && d ∈ {0, 4} && i < NCCM && d ∈ CDTM[i]	*DTM_PSU_OUT* , *DTM_IN0_FEDGE* , *DTM_IN0_REDGE* signals	Cluster external uses ICPA
Module TIM[i]	i < NTIM && d < 6 && d ∈ CDTM[i]	*F_OUT* [x:x] signals; x ∈ { 0, .. 1}	Cluster internal
Module TIO[i]	i < NCCM && CTIO[i] == 1 && d > 6 && d ∈ CDTM[i]	*TIO_G[g]_IN* , *TIO_G[g]_S_OUT* signals ; g ∈ { 0, .. 2}	Cluster internal
Module TOM[i]	i < NTOM && d < 4 && d ∈ CDTM[i]	*TOM_OUT* , *TOM_OUT_T* , *TOM_OUT_HRES[x]* , *TOM_OUT_T_HRES[x]* signals; x ∈ { 0, .. 15}	Cluster internal

Table 105 Dependency of ICM Functions

ICM depends on	Valid index	Description	Type
Module ARCH	i == 0	EIRQ, IRQ signals	Cluster internal
Module ARU	i == 0 && NARU > 0	IRQ signals	Cluster internal
Module ATOM[i]	i < NATOM	IRQ signals	Cluster internal for i==0; else cluster external
Module BRC	i == 0 && NBRC > 0	EIRQ, IRQ signals	Cluster internal
Module CMP	i == 0 && NCMP > 0	EIRQ, IRQ signals	Cluster external
Module DPLL	i == 0 && NDPLL > 0	EIRQ, IRQ signals	Cluster internal
Module FIFO[i]	i < NPSM	EIRQ, IRQ signals	Cluster internal for i==0; else cluster external
Module MCS[i]	i < NMCS	EIRQ, IRQ signals	Cluster internal for i==0; else cluster external

ICM depends on	Valid index	Description	Type
Module SPE[i]	i < NSPE	EIRQ, IRQ signals	Cluster internal for i==0; else cluster external
Module TIM[i]	i < NTIM	EIRQ, IRQ signals	Cluster internal for i==0; else cluster external
Module TIO[i]	i < NCCM && CTIO[i] == 1	IRQ signals	Cluster internal for i==0; else cluster external
Module TOM[i]	i < NTOM	IRQ signals	Cluster internal for i==0; else cluster external

Table 106 Dependency of MAP Functions

MAP depends on	Valid index	Description	Type
Module SPE[i]	NMAP > 0 && i < min(NSPE, 2)	*SPE[i]_NIPD* signals	Cluster external uses ICPA
Module SPE[i]	NMAP > 0 && i < min(NSPE, 2)	*SPE[i]_NIPD_NUM* signals	Cluster external, no ICPA in use; signals are valid with *SPE[i]_NIPD* pulse
Module TIM	NMAP > 0 && NTIM > 0	*TIM_MAP_DATA[x]* [48:0] signals	Cluster internal

Table 107 Dependency of MON Functions

MON depends on	Valid index	Description	Type
Module CCM[1]	NCMP == 1 && NCCM > 1	*CCM[1]_CLK_RES* , *CCM[1]_FXCLK_RES* signals	Cluster internal
Module CMP	NCMP == 1 && NMON == 1	*CMP_ERR* signal	Cluster internal
Module MCS[i]	i < NMCS * NMON	*MCS_CH_MCA* signals	Cluster external uses ICPA
Module MCS[i]	i < NMCS * NMON	*MCS_ERR* signals	Cluster external, no ICPA in use; signals typically are not single cluster clock pulses

Table 108 Dependency of MCS Functions

MCS[i] depends on	Valid index	Description	Type
GTM inputs	i < NMCS	Error IRQ clear, IRQ clear, IRQ set signals	Cluster external uses ICPA
Module AXIM[i]	i < min(NAXIM, NMCS)	IRQ signals	Cluster internal
Module ARU	i < NMCS * NARU	ARU 53-bit data; ARU cluster disable	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use
Module ATOM[i]	i < min(NATOM, NMCS)	IRQ signals	Cluster internal
Module ATOM[i+1]	i < min(NATOM+1, NMCS)	IRQ signals	Cluster external uses ICPA
Module CCM[i]	i < NMCS	*CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , *CCM[i]_TBU_TS2* signals	Cluster internal
Module DPLL	i < min(NPDLL, NMCS)	IRQ signals, DPLL state signals	Cluster internal
Module MCS[j]	i==0; j>0 && j<NMCS	IRQ signals	Cluster external uses ICPA
Module MCS[0]	i>0 && i<NMCS	IRQ clear signals	Cluster external uses ICPA
Module MCS[i+1]	i < NMCS-1	IRQ signals	Cluster external uses ICPA
Module MCS[i-1]	i < NMCS	IRQ clear signals	Cluster external uses ICPA
Module TIM[i]	i < min(NTIM, NMCS)	*TIM_EXT_CAPTURE* , *F_OUT* , IRQ signals	Cluster internal
Module TIM[i+1]	i < min(NTIM+1, NMCS)	*F_OUT* signals	Cluster external, no ICPA in use; signals typically are not single cluster clock pulses
Module TIM[i+1]	i < min(NTIM+1, NMCS)	*TIM_EXT_CAPTURE* , IRQ signals	Cluster external uses ICPA
Module TIO[i]	CTIO[i]==1 && i < NMCS	*TIO_G[g]_S_OUT* , IRQ signals	Cluster internal
Module TOM[i]	i < min(NATOM, NMCS)	IRQ signals	Cluster internal

MCS[i] depends on	Valid index	Description	Type
Module TOM[i]	i < min(NTOM+1, NMCS)	IRQ signals	Cluster external uses ICPA

Table 109 Dependency of FIFO Functions

FIFO[i] depends on	Valid index	Description	Type
GTM inputs	i < NPSM	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA
Module ARU	i < NPSM * NARU	ARU 53-bit data; ARU cluster disable	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use

Table 110 Dependency of SPE Functions

SPE[i] depends on	Valid index	Description	Type
GTM inputs	i < NSPE	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA
Module TIM[0]	i == 0 && NTIM > 0	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 0, .. 2}	Cluster internal
Module TIM[1]	i == 0 && NTIM > 1	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 3, .. 5}	Cluster external; uses ICPA for *TIM_MAP_DVAL*
Module TIM[0]	i == 1 && NTIM > 0	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 0, .. 2}	Cluster external; uses ICPA for *TIM_MAP_DVAL*
Module TIM[1]	i == 1 && NTIM > 1	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 3, .. 5}	Cluster internal
Module TIM[2]	i == 2 && NTIM > 2	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 0, .. 2}	Cluster internal
Module TIM[3]	i == 2 && NTIM > 3	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 3, .. 5}	Cluster external; uses ICPA for *TIM_MAP_DVAL*
Module TIM[2]	i == 3 && NTIM > 2	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 0, .. 2}	Cluster external; uses ICPA for *TIM_MAP_DVAL*
Module TIM[3]	i == 3 && NTIM > 3	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 3, .. 5}	Cluster internal
Module TIM[4]	i == 4 && NTIM > 4	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 0, .. 2}	Cluster internal
Module TIM[5]	i == 4 && NTIM > 5	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 3, .. 5}	Cluster external; uses ICPA for *TIM_MAP_DVAL*
Module TIM[4]	i == 5 && NTIM > 4	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 0, .. 2}	Cluster external; uses ICPA for *TIM_MAP_DVAL*
Module TIM[5]	i == 5 && NTIM > 5	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 3, .. 5}	Cluster internal
Module TIM[6]	i == 6 && NTIM > 6	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 0, .. 2}	Cluster internal
Module TIM[7]	i == 6 && NTIM > 7	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 3, .. 5}	Cluster external; uses ICPA for *TIM_MAP_DVAL*
Module TIM[6]	i == 7 && NTIM > 6	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 0, .. 2}	Cluster external; uses ICPA for *TIM_MAP_DVAL*
Module TIM[7]	i == 7 && NTIM > 7	*TIM_MAP_DATA[x]* [48:48], *TIM_MAP_DVAL* signals; x ∈ { 3, .. 5}	Cluster internal

Table 111 Dependency of TBU functions

TBU depends on	Valid index	Description	Type
Module CCM[0]	i == 0	*CCM[0]_CLK_RES* signals	Cluster internal
Module DPLL	NDPLL == 1	*SUB_INC1C* , *SUB_INC2C* signals	Cluster internal
Module MAP	NMAP > 0	*TRIGGER* , *STATE* , *T_DIR* , *S_DIR* signals	Cluster internal

Table 112 Dependency of TIM functions

TIM[i] depends on	Valid index	Description	Type
GTM inputs	i < NTIM	Error IRQ clear, IRQ clear signals	Cluster external uses ICPA
Module ARU	i < NTIM * NARU	ARU 53-bit data; ARU cluster disable	Cluster internal, cluster external: ARU protocol ensures correct operation in case of different cluster clock frequencies are in use
Module CCM[i]	i < NTIM	*CCM[i]_CLK_RES* , *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , *CCM[i]_TBU_TS2* signals	Cluster internal
Module CDTM[i]_DTM[d]	i < NTIM && d ∈ {0, .. 5}	*DTM_OUT0* , *DTM_OUT1* signals	Cluster internal
Module TIM[i+1]	i < NTIM-1	*TIMCH0_EXT_CAP_SRC_OUT* signals	Cluster external uses ICPA

Table 113 Dependency of TIO Functions

TIO[i] depends on	Valid index	Description	Type
GTM inputs	i ∈ CTIO[i]	IRQ clear signals	Cluster external uses ICPA
Module CCM[i]	i ∈ CTIO[i]	*CCM[i]_CLK_RES* , *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , *CCM[i]_TBU_TS2* signals	Cluster internal
Module MCS[i]	i ∈ CTIO[i] && i < NMCS	IRQ clear signals	Cluster internal

Table 114 Dependency of TOM Functions

TOM[i] depends on	Valid index	Description	Type
GTM inputs	i < NTOM	IRQ clear signals	Cluster external uses ICPA
Module CCM[i]	i < NTOM	*CCM[i]_FXCLK_RES* , *CCM[i]_TBU_TS0* , *CCM[i]_TBU_TS1* , *CCM[i]_TBU_TS2* signals	Cluster internal
Module MCS[i]	i < min(NTOM, NMCS)	IRQ clear signals	Cluster internal
Module MCS[i-1]	i < min(NTOM, NMCS+1)	IRQ clear signals	Cluster external uses ICPA
Module SPE[i]	i < min(NTOM, NSPE)	*SPE[i]_OUT* , *SPE[i]_NIPD* , signals	Cluster internal
Module TIM[i]	i < min(NTIM, NTOM)	*TIM_EXT_CAPTURE* signals	Cluster internal
Module TOM[0]	i == 0 && NTOM > 1	*TOM_TRIGOUT_DEL* , *TOM_HRES_OUT_DEL* signals	Cluster internal
Module TOM[i-1]	i > 1 && i < min(NTOM+1)	*TOM_TRIGOUT* signals	Cluster external uses ICPA
Module TOM[i-1]	i > 1 && i < min(NTOM+1)	*TOM_HRES_OUT* signals	Cluster external, no ICPA in use

28.10 Compatibility Notes / Functional Changes Versus GTM v.3.1.5 Release

Find next a listing of all functional changes which have been introduced with GTM v4.1.x release where backward compatibility to GTM v3.1.5 is not existing. Software developers have to introduce changes to their application code to ensure correct operation of the same application on GTM 4.1.x devices.

28.10.1 GTM Architecture

- Address map changed, base address of all modules changed. Each module inside a cluster has now a defined offset to the cluster start address. As long as applications use C-HAL or symbolic names no corrective actions are needed.

- Legacy register from GTM generation 1 and GTM generation 2 removed.
- GTM_TIM[i]_AUX_IN_SRC replaced by **CCM[i]_TIM_AUX_IN_SRC**
- GTM_EXT_CAP_EN_[i] replaced by **CCM[i]_EXT_CAP_EN**
- GTM_TOM0[i]_OUT replaced by **CCM[i]_TOM_OUT**

- GTM_ATOM[i]_OUT replaced by **CCM[i]_ATOM_OUT**
- **GTM_AEI_STA_XPT.W1R0** changed to bit 24.

- Redefinition of AEI_STATUS behavior in case of AEI configuration access to undefined, protected, disabled register. *3.5.1 "GTM-IP Generic Bus Interface (AEI)"*

- Change of module address map for MCS AEI master interface. Offsets are now identical to CPU AEI interface. As long as MCS applications use C-HAL or symbolic names no corrective actions are needed.

28.10.2 ICM

Removed the following legacy and redundant ICM registers: **ICM_IRQG_6** , **ICM_IRQG_7** , **ICM_IRQG_8** , **ICM_IRQG_9** , **ICM_IRQG_10** , **ICM_IRQG_1** .

28.10.3 ATOM

- **ATOM[i]_CH[x]_CTRL.CLK_SRC** , **ATOM[i]_CH[x]_CTRL.SL** shadow register functionality (see chapter **ATOM[i]_CH[x]_CTRL_SR**): To ensure backward compatibility, new shadow register must be written with same values as operational register **ATOM[i]_CH[x]_CTRL_SR** = **ATOM[i]_CH[x]_CTRL** .
- SOMP using ARU update: *15.3.3.2 "ARU controlled update"* . ACB bits allow to specify the **ATOM[i]_CH[x]_CTRL.CLK_SRC** and **ATOM[i]_CH[x]_CTRL.SL** for each ARU transfer. SOMP applications using ARU update mechanism have to specify the ACB bits for each ARU transfer: **ATOM[i]_CH[x]_CTRL.ACB** [4:1] = **ATOM[i]_CH[x]_CTRL.CLK_SRC** , **ATOM[i]_CH[x]_CTRL.ACB** [0:0]= **ATOM[i]_CH[x]_CTRL.SL** .

- Bit field ECLK_SRC of **ATOM[i]_CH[x]_CTRL** removed and **ATOM[i]_CH[x]_CTRL.CLK_SRC** bit field expanded by one bit and coding of bit field **ATOM[i]_CH[x]_CTRL.CLK_SRC** changed: **ATOM[i]_CH[x]_CTRL_SOMP** , **ATOM[i]_CH[x]_CTRL_SOMS** , **ATOM[i]_CH[x]_CTRL_SOMI**

- Behavior of retriggering a one-shot cycle in SOMP mode changed: *15.3.3.6 "One-shot Counting Up Mode"*

28.10.4 TOM

- **TOM[i]_CH[x]_CTRL.CLK_SRC** , **TOM[i]_CH[x]_CTRL.SL** shadow register functionality (see **TOM[i]_CH[x]_CTRL_SR**): To ensure backward compatibility, new shadow register must be written with same values as operational register **TOM[i]_CH[x]_CTRL_SR** = **TOM[i]_CH[x]_CTRL** .

- Bit field ECLK_SRC of **TOM[i]_CH[x]_CTRL** removed and **TOM[i]_CH[x]_CTRL.CLK_SRC** bit field expanded by one bit and coding of bit field **TOM[i]_CH[x]_CTRL.CLK_SRC** changed.

- Behavior of retriggering a one-shot cycle changed: *14.6 "One-Shot Counting Up Mode"*

28.10.5 MCS

Added new MCS internal Special Function Registers **TIOI** and **TIOSL** .

- The bit field **MCS[i]_CTRL_STAT.XOREG** is replaced by **MCS[i]_CTRL_STAT.EN_HVD** .

- Enable write protection to **RS[y]** registers after reset: **MCS[i]_REG_PROT**

- Removed stack pointer counter bit field SP_CNT of register **STA** and register **MCS[i]_CH[x]_CTRL** .

- Implement safe indirect memory addressing by evaluation of MSBs at adress offset and base register: *17.9.6 "MRDI Instruction"* , *17.9.8 "MWRI Instruction"* , *17.9.7 "MRDIO Instruction"* , and *17.9.9 "MWRIO Instruction"* .

- Removed write protection on register R6 while running ARU instructions with indirect ARU addressing.

- The SFRs **STA** , **ACB** , **CTRG** , **STRG** , **TBU_TS0** , **TBU_TS1** , **TBU_TS2** , and **MHB** can no longer be used as base address registers for the memory access instructions.

28.10.6 ARU

- ARU read address map changed. New ARU address assignment is listed in *94 "ARU Master ID"* .

28.10.7 CCM

- Remove bit field ARU_CONNECT_CONFIG in register **CCM[i]_HW_CONF** due to discontinued device configuration parameter.

28.11 New Features / Functional Enhancements

Find next a listing of all functional enhancements which have been introduced with GTM v4.1.x release where backward compatibility to GTM v3.1.5 is not affected.

28.11.1 GTM Architecture

- ARU Cluster Isolation: Disable ARU transfers for distinct clusters. *3.9.3 "ARU Cluster Isolation"*
- GTM_CTL: extended functionality of **GTM_CTRL.TO_MODE** ; internal AEI accesses can be retrieved or aborted. See chapter **GTM_CTRL**
- AXIS: CPU configuration interface, which operates on AXI protocol. Alternatively, it can be used to the AEI CPU configuration interface. *3.5.2 "AXI Slave"*

28.11.2 ATOM

- High resolution support: *3.8 "High Resolution PWM Support"* , *15.3.3.10 "High resolution PWM support (HRES mode)"*
- Retrigger oneshot mode: *15.3.3.6 "One-shot Counting Up Mode"* , **ATOM[i]_CH[x]_STAT.OSM_RTF**
- Shadow register for **ATOM[i]_CH[x]_CTRL.SL** and **ATOM[i]_CH[x]_CTRL.CLK_SRC** : **ATOM[i]_CH[x]_CTRL_SR**
- Functionality of ACB bits if ARU routing is in use with SOMP mode enhanced. ACB bits allow to specify the **ATOM[i]_CH[x]_CTRL.CLK_SRC** and **ATOM[i]_CH[x]_CTRL.SL** for each ARU transfer. *15.3.3.2 "ARU controlled update"*
- ATOM OUT_T controlled by ENDIS (enhancement): Value of *ATOM_OUT_T* is inverse signal level (**ATOM[i]_CH[x]_CTRL.SL**) when channel is disabled (ENDIS=0): **ATOM[i]_AGC_OUTEN_CTRL.OUTEN_CTRL[k]**

28.11.3 TOM

- High resolution support: *3.8 "High Resolution PWM Support"* , *14.12 "High Resolution PWM Support (HRES Mode)"*
- Retrigger oneshot mode: *14.6 "One-Shot Counting Up Mode"* , **TOM[i]_CH[x]_STAT.OSM_RTF**
- Shadow register for **TOM[i]_CH[x]_CTRL.SL** and CLK_SRC: **TOM[i]_CH[x]_CTRL_SR**
- TOM OUT_T controlled by ENDIS (enhancement): Value of *TOM_OUT_T* is inverse signal level (**TOM[i]_CH[x]_CTRL.SL**) when channel is disabled (ENDIS=0): **TOM[i]_TGC[g]_OUTEN_CTRL.OUTEN_CTRL[c]**

28.11.4 DPLL

- DPLL direction control is providing the option to actively switch into either forward or backward direction with inactive input signal of *TRIGGER* input. For this purpose, the bit field **DPLL_THMI.THMI** has to be configured in a dedicated manner.

- Generation of active or inactive input signals for *TRIGGER* and *STATE* inputs by writing on newly defined register **DPLL_SW_TRIG** . See *21.11.63 "DPLL_SW_TRIG"* .
- Improved check of nominal value of angle clock by introducing registers **DPLL_MP_T** and **DPLL_MP_S** which are providing the number of missing pulses of *SUB_INC1* or *SUB_INC2* pulse generators at the point of time when a new active input signal edge on *TRIGGER* / *STATE* inputs arrive.
- The transition of *CCM[0]_TBU_TS1* between different sources of pulse generation (*SUB_INC1C* , *SUB_INC2C*) is now possible immediately by configuration of the register **DPLL_CTRL_12** .

28.11.5 DTM

- Dead time shadow register including **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** and **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** : *16.2.3 "Dead Time Shadow Register"* , **CDTM[i]_DTM[d]_CH[x]_DTV_SR**
- Expand bitwidth of **CDTM[i]_DTM[d]_CH[x]_DTV.RELRISE** / **CDTM[i]_DTM[d]_CH[x]_DTV.RELFALL** to 13-bit.
- AUXIN enhancements for GTM Gen4 (individual shut off): *16.7 "DTM Individual Channel Shut Off"*
- High resolution support: *16.8 "High Resolution PWM Support"*
- Allow usage of DTM modules after a TIO module.

28.11.6 MCS

- Memory accesses can now make usage of a modified Harvard Architecture: *17.4.1 "Modified Harvard Architecture"*
- Added new debugging capabilities with hardware break points: *17.8 "Debugging"*
- Added new shared interrupts that can either be set by MCS-channels or from external peripherals: **MSINT** , **MCS[i]_SINT_IRQ_NOTIFY** , **MCS[i]_SINT_IRQ_EN** , **MCS[i]_SINT_IRQ_FORCINT** , and **MCS[i]_SINT_IRQ_MODE** .

28.11.7 TIM

- Added new coding for **TIM[i]_CH[x]_ECTRL.TDU_STOP** =0b110. See chapter **TIM[i]_CH[x]_ECTRL**

28.11.8 TIO

- Added new module to GTM architecture, allows to operate as input or output. *27 "Timer Input Output (TIO)"*

28.11.9 CCM

- Added register to read actual value of output signal after DTM behind TIO module. **CCM[i]_TIO_G0_OUT** , **CCM[i]_TIO_G2_OUT**
- Added register **CCM[i]_HW_CONF2** to monitor additional device configuration parameter.
- Added bit fields **CCM[i]_HW_CONF.RESET_ACTIVE** , **CCM[i]_HW_CONF.AEI_ADDR_PIPELINE_STAGE** , **CCM[i]_HW_CONF.AEI_RDATA_PIPELINE_STAGE** to monitor additional device configuration parameter.

28.11.10 AXIM

- Added new module to GTM architecture, allows a MCS module to act as bus master external to the GTM. *4 "AXI Master"*

28.12 Appendix A Signals

28.12.1 Appendix A Signals

Loop	–
Condition	–
Format	–

SIG	
Description	Dummy signal
Loop	–
Condition	–
Signal Type	INT
Assignment	–

BOSCH

A List of Tables

B List of Figures

C Version Information