GTM-IP

Application Note AN013
DPLL PMT generation

Date: 03.07.2012
(Released)

Robert Bosch GmbH
Automotive Electronics (AE)

LEGAL NOTICE

Revision History

Issue	Date	Remark
0.1	19.10.2011	Initial version
0.2	06.12.2011	Update for GTM-IP Specification v1.5.0
0.3	03.07.2012	Update for GTM-IP Specification v1.5.4

Tracking of major changes

Changes between revision 1.x and 1.y

NA

Conventions

The following conventions are used within this document.

ARIAL BOLD CAPITALS	Names of signals
Arial bold	Names of files and directories
`Courier bold`	Command line entries
`Courier`	Extracts of files

References

This document refers to the following documents.

Ref	Authors(s)	Title
1	AE/EIN2	GTM-IP Specification v1.5.4
2	AE/EIN2	AN011 DPLL Micro tick generation

Terms and Abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
GTM	Generic Timer Module

Table of Contents

1 Overview

This application note describes the DPLL PMT calculation feature. This feature offers the functionality to request a calculation service from the DPLL on behalf of a combination of the DPLL angle clock and the time domain. The system provides a position of the angle clock and a time in clock ticks of the TBU_TS0 clock and the DPLL calculates the position and time, this event will take place (Position-Minus-Time (PMT)).

1.1 Use case

In engine management systems it is important to know the position of the engine and the corresponding time to manage and control the engine. Since the valves have a physical delay, this delay has to be taken into account, when an action on the engine should be applied. For example if someone wants to initiate an injection of a pump at a specific engine position, he has to schedule the injection a little bit before this engine position to give the pump a little bit of time to open.

The DPLL is able to calculate the time and engine position out of the actual engine behaviour and predict this times for the future. This PMT calculation is done automatically for each new arriving tooth.

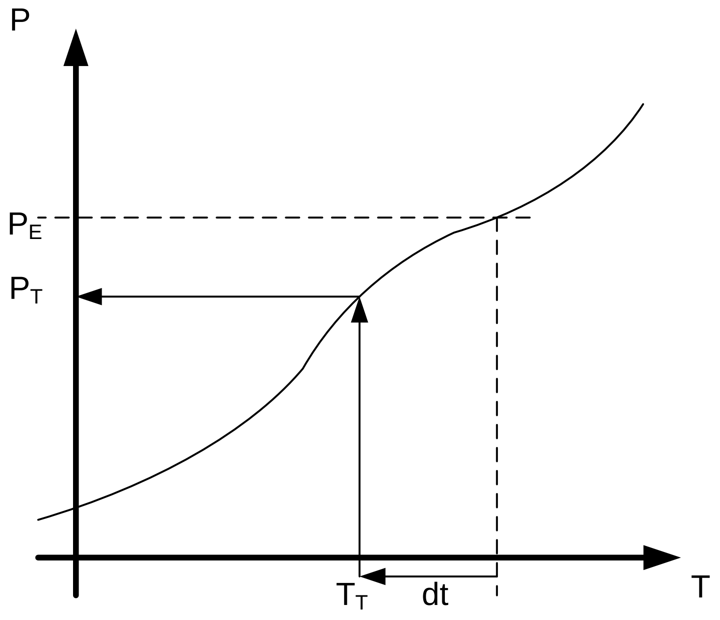

Figure 1.1: PMT calculation principle.

Figure 1.1 should clarify the PMT functionality. On the y-axis one can see the position of the engine while the x-axis represents the time and the swinging line represents the behaviour of the engine. At some time long before target time T_T, the user

requests the calculation of this target time and the target position P_T. This is done by specifying the engine position P_E and a delta time *dt* in clock ticks of the TBU time stamp clock of channel 0.
Please note, that these requests for PMT calculation can only issued via the ARU interface of the DPLL, thus only modules which can issue write requests to the ARU are able to do these requests. These are the TIM, PSM and MCS submodules.

1.2 System architecture

For this application note, the PMT calculations should be issued by the MCS and the PMT results should be shown by actions on the ATOM outputs. This section should describe the system architecture and submodules typically involved in the PMT calculation mechanism.

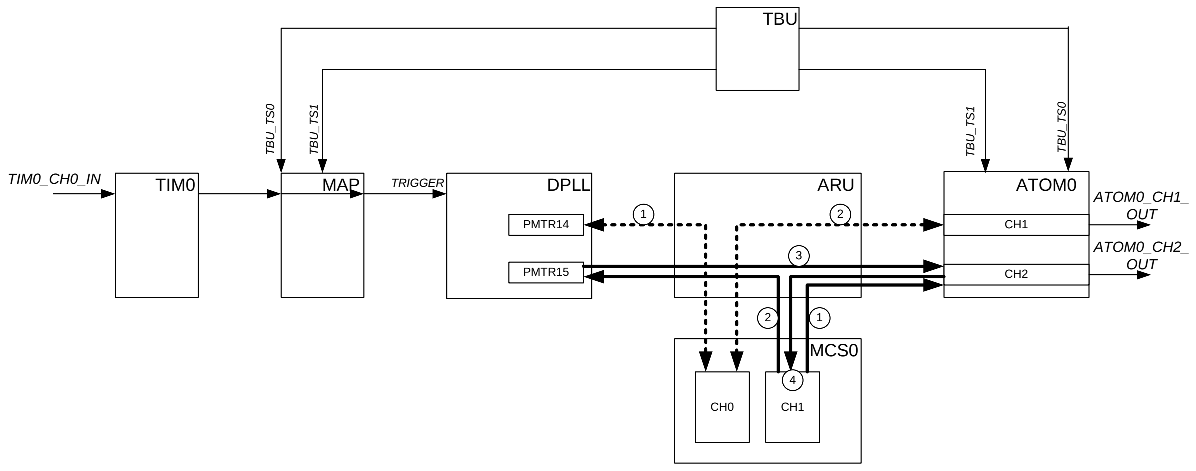

Figure 1.2: Application of the PMT functionality on the GTM device.

Figure 1.2 shows main submodules involved in the PMT functionality. For the DPLL to generate micro ticks a TIM input channel, the MAP submodule and the TBU is needed. For the PMT functionality a MCS channel, the ARU and the DPLL are needed. To visualize the result, an ATOM channel can be used which creates an edge at the output, when the PMT time is reached.
This application note shows two possibilities how to create such a PMT event at the ATOM outputs. Therefore, two MCS channels, two DPLL PMT channels, and two ATOM channels are used.
The first possibility is shown by the doted communication lines via the ARU. The MCS channel zero (0) first sends a PMT request to the DPLL channel 14 and receives the calculated action time at each incoming tooth from this DPLL channel. The MCS channel than can sends this action time stamps to the ATOM channel one (1) for edge creation.
With the second possibility, the MCS channel is not involved in sending the calculated action times to the ATOM channel. There, the MCS channel one (1) first

sets up the ATOM channel two (2) to receive subsequent match times from the DPLL channel 15. After setting up the ATOM, the MCS channels issues a PMT request with DPLL channel 15. The MCS channel than waits for the match event at ATOM channel two to occur. Further communication takes place between DPLL channel 15 and ATOM channel two (2) directly via ARU, where the ATOM channel receives at each incoming tooth new PMT action values.

For the PMT calculations and especially the output of these actions at the ATOM channels, micro ticks generated by the DPLL are needed. For the basic principle of micro tick generation please refer to the application note AN011.

1.3 Requirements

This application note only runs in combination with application note AN011, since micro tick generation has to be established for the PMT application. The following table shows the required environment:

Module	**Version**
GTM-RM	Starting from GTM-RM v1.4.4
GTM-IP	Starting from GTM-IP v1.4.2
AN011	v0.2
AN013	v0.2

2 Submodule setup

2.1 Overview

For the PMT calculation with the DPLL a TRIGGER or STATE input signal is needed on behalf of which micro ticks for the angle clock can be generated an the PMT results can be calculated. For the micro tick generation, the same startup up code and testbench environment as for application note AN011 “Micro tick generation with DPLL” is used.
Therefore, the code consists of two parts. The first part is identical to AN011 and the second part establishes the PMT application. Here first the DPLL micro tick part is described in brief. Starting from second 2.3 the PMT application is described.

2.2 DPLL Micro tick generation

Figure 2.1 shows the testbench setup for the DPLL micro tick generation. The TRIGGER input signal is generated as a PWM signal with ATOM0 Channel seven (7) where the PWM characteristic is served in a FIFO ring buffer mode.

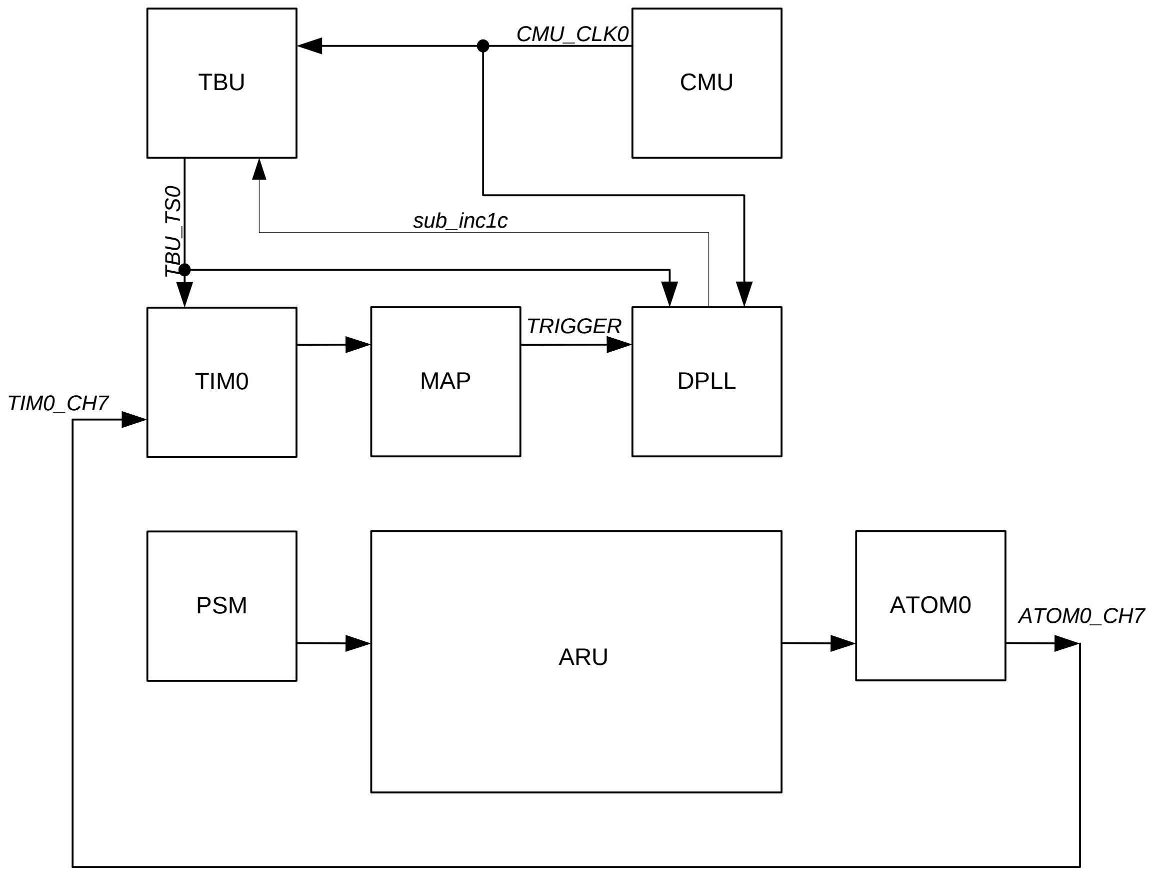

Figure 2.1: Micro tick generation with DPLL.

The DPLL than receives the ATOM PWM signal via TIM0 Channel seven (7) and input multiplexer for channel zero (0) on the TRIGGER input line.

2.3 ARU Communication plan

The GTM router is based on a addressing scheme, were the write addresses for the submodule channels are fixed and the read addresses can be configured by software to guarantee different routing possibilities for different application domains.
For this application note, different routes have to be setup between the DPLL, MCS0 and ATOM0 channels. These routes and their corresponding addresses are shown in the following table:

Submodule	Channel	Write address	Destination
DPLL	CH14	0x18d	MCS0 CH0
	CH15	0x18e	ATOM0 CH2
MCS0	CH0	0x077	DPLL CH14
	CH0	0x078	ATOM0 CH1
	CH1	0x079	DPLL CH15
	CH1	0x07A	ATOM0 CH2
ATOM0	CH1	0x120	MCS0 CH0
	CH2	0x121	MCS0 CH1

Table 2.1: ARU communication scheme.

2.4 MCS Controlled action generation

As mentioned above, there are two possibilities to program a PMT event for the GTM. One is a closely coupled communication between the MCS and the DPLL and no direct communication path between the DPLL and the corresponding ATOM channel. The resources for this PMT application are listed in the following table:

Submodule	Channel
DPLL	CH14
MCS0	CH0
ATOM0	CH1

Table 2.2: GTM Resources for closely coupled MCS – DPLL communication.

The operation principle is shown in Figure 2.2. As it can be seen from the figure, the communication pathes are drawn between the MCS and the DPLL and the MCS and the ATOM. In this application variant, the MCS channel is in close interaction with the DPLL, issues a PMT request first, sets up a match event at the ATOM and than waits for the PMT results from the DPLL. When the PMT result is close enough, the MCS issues a match request with the ATOM a last time and than waits for the match event to happen.

Before this sequence starts, the MCS channel has to wait for the DPLL to be synchronized. This synchronisation is determined by the CPU and signalled to the MCS via a TRIGGER event.

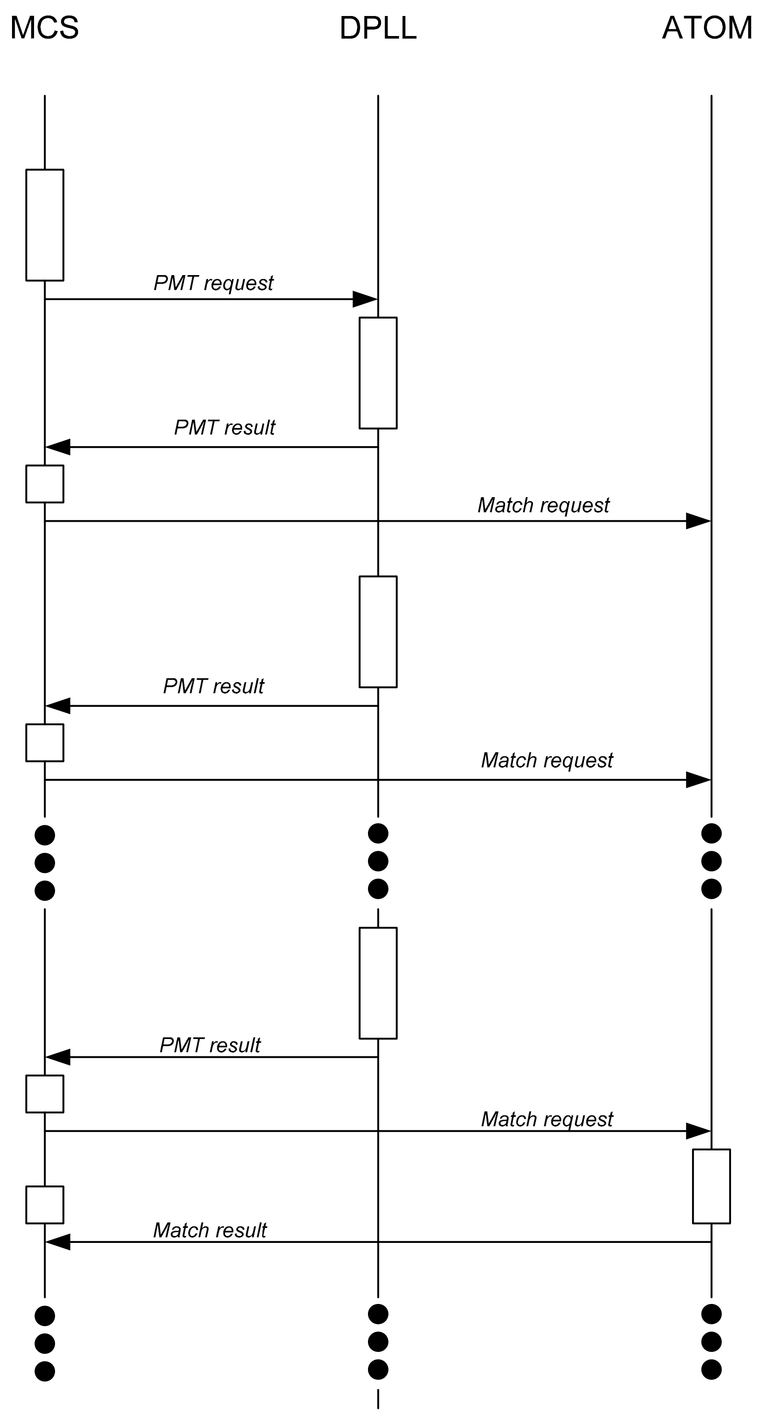

Figure 2.2: PMT functionality with closely coupled MCS – DPLL path.

For this application to work, the MCS, DPLL and ATOM channel have to be configured.

2.4.1 DPLL Configuration

For the action calculation, the DPLL needs to know the source where the PMT requests and values for calculation come from. This source has to be configured in the **DPLL_ID_PMTR_x** register, where *x* is the channel for the calculation. In total, the DPLL offers 24 such calculation channels. Since these ID registers are write protected when the DPLL is enabled, the read addresses have to be configured before the DPLL is initialized. For this application note and according to Table 2.1 the DPLL PMT CH14 gets it requests from the MCS0 CH0 and their from the address 0x77 and the DPLL PMT CH15 gets its data from MCS0 CH1 write address 0x79.

The action calculation can be enabled with the action enable bits in the **DPLL_CTRL_2**, **3**, **4** registers. For this application note for channels 14 and 15, the **DPLL_CTRL_3** register has to be used. This is done after the DPLL has locked on the TRIGGER input signal, inside of the GL1I ISR.

2.4.2 ATOM0 CH1 Configuration

The ATOM0 CH1 is intended to do a compare match for the calculated PMT position and time. The channel receives this data via ARU from the MCS0 CH0. Therefore, two registers have to be initialized for this application. First, the ARU read address has to be configured. According to Table 2.1 this is the MCS0 CH0 write address 0x78. This has to be programmed into the register **ATOM0_CH1_RDADDR**. Than, the channel is initialized in SOMC mode with ARU enabled.

The ATOM channel and its output are enabled together with the second ATOM channel used for the PMT application demonstration. After the match event, the ATOM0 CH1 raises an interrupt to the CPU which causes the MCS CH0 to cancel any subsequent ARU transfers.

2.4.3 MCS0 CH0 Implementation

The MCS0 CH0 has a closely coupled connection to the DPLL and the ATOM CH1. Code fragment 2.1 shows the MCS implementation for MCS0 CH0.

```
tsk_x0: lit24 400    ; operand 1: Time     0x190
tsk_x1: lit24 110592 ; operand 2: Position 0x1B000

;*************************************************************
;*   task 0: closely-coupled mcs task for action calculation
;*   tsk_x0  Time value for action
;*   tsk_x1  Position value for action
;*************************************************************

tsk0_init:
      mrd    R0   tsk_x0   ; load PMT values to register R0 and R1
      mrd    R1   tsk_x1
      movl   ACB  $3       ; SF then toggle
      movl   R2   $1       ; setup trigger bit to wait for
      wurm   R2 CTRG $1    ; wait on trigger register match
      awr    R0 R1 $0      ; issue PMT request on WR_ADDR 0x77 --> DPLL CH14
```

```
   tsk0_prq:
18       ard    R4 R5 $18D         ; read DPLL CH14 result (0x18D)
19       jbs    STA CAT tsk0_done   ; if ARU transfer canceled by CPU stopp ch.
20       awr    R4 R5 $1           ; write DPLL calculated result to ATOM
21       jmp    tsk0_prq           ; one more time...
22 tsk0_done:
23       andl  STA ~EN_MSK         ; disable task
```

Code 2.1: MCS0 CH0 implementation.

In lines 01 and 02 the PMT request values for the DPLL are stored. The action time should be at position stamp 0x1B000 minus 0x190 ticks of the time stamp clock TBU_TS0. Then, the task starts with first loading the PMT values into the MCS registers R0, and R1 and the ARU control bits 0x3, which is the command for the ATOM to do a toggle at the first match is detected on either time or position.
Starting from line 14 a trigger synchronization point is prepared. A trigger bit position 0 is chosen in register R2 (line 14), and a wait until register match WURM on the MCS0 Trigger register CTRG is set up with masking of bit 0. By this construction, the MCS0 CH0 is stopped until the CPU triggers the channel by setting the trigger bit 0.
With line 16 the PMT application is done. Line 16 first sends the PMT request at the MCS0 write address offset 0. This is the write address 0x77 for MCS0. Destination is the DPLL channel 14 (see also Table 2.1).
The MCS then receives the loops. First, the PMT request is done in line 16. The MCS0 channel waits in line 17 for the calculated result. This result is received for every tooth. When the ARU read request was not canceled by the CPU by writing to the corresponding CAT bit 0 in the **MCS0_RST** register, the channel writes the received compare values in register R4, R5 as is the ATOM channel (line 20), jumps back to label `tsk0_prq` and waits for an actualized PMT result from the DPLL.
When the ARU read command is canceled by the CPU, the MCS0 CH0 jumps to label tsk0_done and disables himself by clearing the channel enable bit in line 23. Cancelation of the ARU read request is done by the CPU, when the ATOM0 CH1 matches on the configured compare values and has raised an interrupt.

2.5 DPLL Controlled action generation

The second possibility to establish a PMT calculation and action event at the ATOM output is a loosely communication scheme between MCS and DPLL and a more closely communication directly between DPLL and ATOM channel. The resources for this PMT application are listed in the following table:

Submodule	Channel
DPLL	CH15
MCS0	CH1
ATOM0	CH2

Table 2.3: GTM Resources for loosely coupled MCS – DPLL communication.

The communication scheme is shown in Figure 2.3. The first two parts (1), (2) of the MCS program are intended to reprogram the ATOM channels read address to listen to the DPLL CH15 write address and to setup the PMT request at the DPLL CH15. After that, the MCS CH1 issues a read request on the ATOM CH2 match event (3) and blocks here until the match event occurs.
The DPLL calculates at each tooth new PMT compare values and sends these values directly to the ATOM channel (1).
The ATOM channel changes his read address (1) to listen to the DPLL and at one point in time matches on the compare values (2) and sends the result to the MCS.

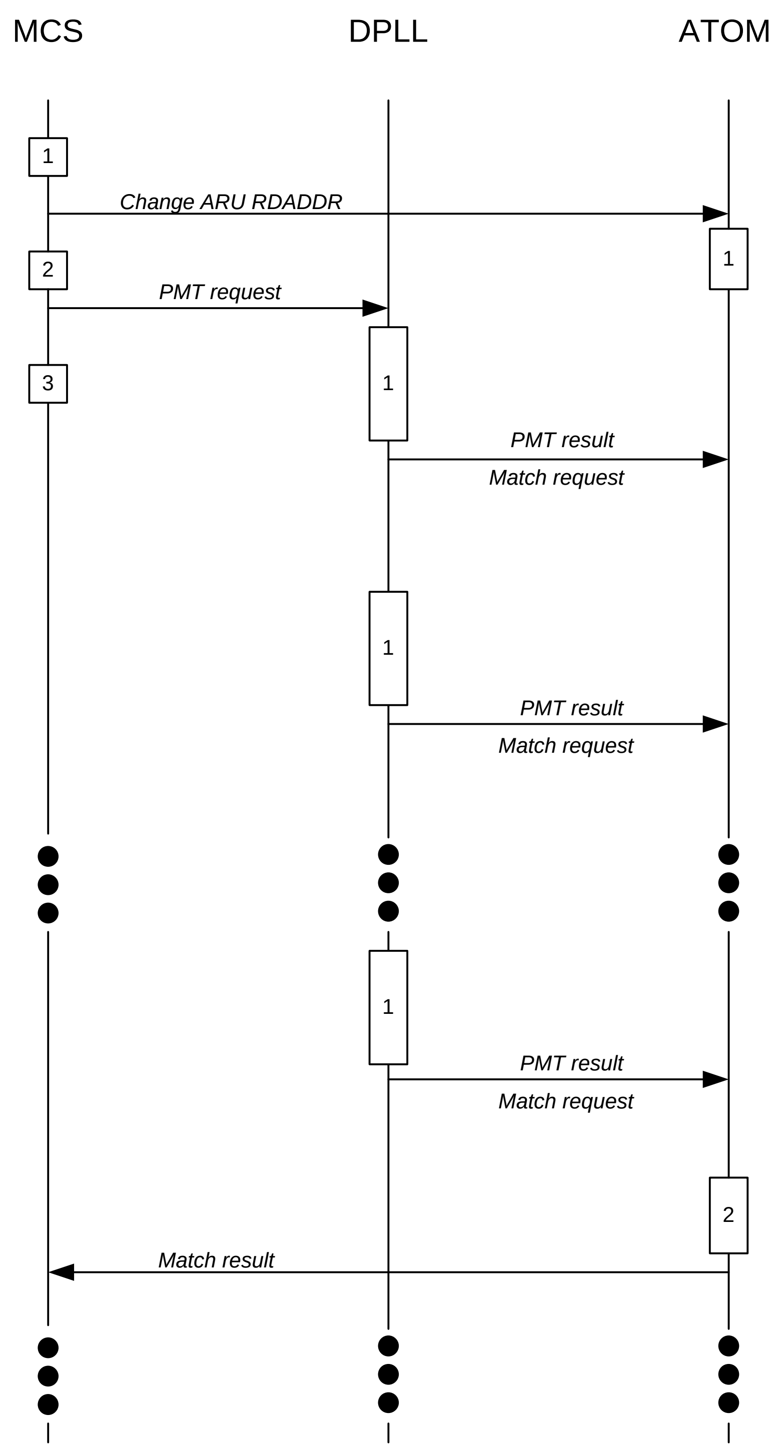

Figure 2.3: PMT functionality with loosely coupled MCS – DPLL path.

2.5.1 DPLL Configuration

In the loosely coupled PMT application there is no difference for the DPLL setup. The read address has to be defined for CH15. This is the write address of the MCS0 CH2.

The DPLL CH15 is enabled together with CH14 and calculates new PMT results, when the channel receives the PMT request from the MCS.

2.5.2 ATOM0 CH2 Configuration

For the ATOM0 CH2, there is a difference in configuring the ARU read addresses. Each ATOM channel has the possibility to read from two different addresses. The ATOM channels ARU read address register is shown in Register 2.5.2.1.

2.5.2.1 Register ATOM[i]_CH[x]_RDADDR (x: 0...7)

Address Offset:	**see Appendix B**																**Initial Value: 0x01FE_01FE**															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit	Reserved							RDADDR1									Reserved							RDADDR0								
Mode	R							RPw									R							RPw								
Initial Value	0x00							0x1FE									0x00							0x1FE								

Bit 8:0	**RDADDR0**: ARU Read address 0.
	Note: This read address is used by the ATOM channel to receive data from ARU immediately after the channel and ARU access is enabled (see ATOM[i]_CH[x]_CTRL register for details).
	Note: This bitfield is only writeable if channel is disabled.
Bit 15:9	**Reserved**: Read as zero, should be written as zero.
	Note: Read as zero, should be written as zero.
Bit 24:16	**RDADDR1**: ARU Read address 1.
	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 RDADDR0 after one ARU data package was received on RDADDR1.
	Note: This read address is only applicable in SOMC mode.
	Note: This bitfield is only writeable if channel is disabled.
Bit 31:25	**Reserved**: Read as zero, should be written as zero.
	Note: Read as zero, should be written as zero.

The main read address is defined by bitfield RDADDR0. This read address is used by the channel when it is enabled. The read address can be changed by a source at the ARU by sending the value ‘111’ in the ACB control bits 4 downto 2. This will cause the ATOM channel to listen to RDADDR1 further on, until a compare match event occurs. Then, the channel switches back to RDADDR0.
Please note, that this feature is only available in ATOM SOMC mode.

As it can be seen from Table 2.1, the ATOM0 CH2 occurs in the destination column two times. One source is the DPLL CH15 and the second one is the MCS0 CH1. Therefore, there are two addresses to configure, where the main read address should be the MCS channel. The corresponding code can be seen in the following code fragment.

```
ATOM0_CH2_RDADDR     = 0x018E007A;  // get data from DPLL CH15 and MCS0 CH1
ATOM0_CH2_CTRL       = 0x9;         // ATOM0 CH2 in SOMC, ARU enabled

ATOM0_AGC_OUTEN_STAT = 0x0028;      // enable CH1 and CH2 output
ATOM0_AGC_ENDIS_STAT = 0x0028;      // enable CH1 and CH2
```

Code 2.2: ATOM0 CH2 configuration.

2.5.3 MCS0 CH1 Implementation

The MCS0 CH1 implementation differs also from the closely coupled solution. The code can be seen in Code fragment 2.3.

```
01  tsk_x0: lit24 400    ; operand 1: Time     0x190
02  tsk_x1: lit24 110592 ; operand 2: Position 0x1B000
03
04  ;************************************************************
05  ;*   task 1: loosely-coupled mcs task for action calculation
06  ;*   tsk_x0  Time value for action
07  ;*   tsk_x1  Position value for action
08  ;************************************************************
09
10  tsk1_init:
11        mrd    R0    tsk_x0     ; load PMT values to register R0 and R1
12        mrd    R1    tsk_x1
13        movl   ACB   $1F        ; ATOM - ACB: change read address to DPLL
14        awr    R0 R1 $3         ; write request to ATOM channel
15        movl   R2    $1         ; setup trigger bit to wait for
16        wurm   R2 CTRG $1       ; wait on trigger register match
17        movl   ACB  $3          ; ATOM - ACB: SF then toggle
18        awr    R0 R1 $2         ; issue PMT req. on WR_ADDR 0x79 --> DPLL CH15
19        ard    R4 R5 $121       ; read ATOM CH2 result after match (0x121)
20        orl    STA IRQ_MSK      ; raise IRQ, when result received from ATOM
21  tsk1_done:
22        andl   STA ~EN_MSK      ; disable task
```

Code 2.3: MCS0 CH1 code for loosely coupled MCS – DPLL communication.

The MCS0 CH1 task also initializes the registers R0 and R1 with the PMT request values in line 11 and 12 but then configures the ACB register with all ‘1’. This ACB command is send in line 14 to the ATOM0 CH2 and causes this channel to listen further on to the DPLL PMT results.

With line 15 and 16 the wait until register match sequence is programmed to wait for the CPU trigger to continue with the work.
Lines 17 and 18 are used for the PMT request to the DPLL. Please note, that the ACB bit field has to be changed to the action which should be executed by the ATOM channel on a compare match event. This action command is send together with the PMT request data in registers R0 and R1 with the command in line 18. The DPLL will send this programmed ACB bits together with the PMT result, and because the ATOM channel is supposed to listen to the DPLL further on will get the compare match values together with these ACB bits.
The ARU read command in line 19 then causes the MCS0 CH1 to sleep until the match event at the ATOM occurs and the match times are provided by the ATOM channel. The MCS0 CH1 then raises an interrupt in line 20 and disables himself.
The interrupt is used by the application note to check the action results. This check is described in section 3.2.

3 Implementation

3.1 Overview

3.1.1 AN013 main()

The main application code is shown in code fragment 4.1.

```
...

cout << "Configure DPLL" << endl;
cout << "========================================================" << endl;
// setup DPLL PMTR read addresses before DPLL enable
cout << "Setup DPLL PMTR 14, 15 first" << endl;
DPLL_ID_PMTR_14       = 0x077;           // receive PMT value from MCS0 CH0
DPLL_ID_PMTR_15       = 0x079;           // receive PMT value from MCS0 CH1
init_dpll();

// start the MCS ch0, ch1 programs
cout << "Initialize MCS0 RAM..." << sizeof(mcs0_mem) << endl;
p = &MCS0_MEM;
for (tmp = 0; tmp<sizeof(mcs0_mem); tmp++) {
  p[tmp] = mcs0_mem[tmp];
}
MCS0_CH0_CTRL   = 0x1;
MCS0_CH1_IRQ_EN = 0x1;   // enable channel 1 IRQ
MCS0_CH1_CTRL   = 0x1;

// setup ATOM channels 1 and 2
cout << "Setup ATOM CH1,2..." << endl;
ATOM0_CH1_RDADDR  = 0x078;          // get data from MCS0 CH0
ATOM0_CH1_IRQ_EN  = 0x3;            // enable CCU0/1 IRQ
ATOM0_CH1_CTRL    = 0x9;            // ATOM0 ch1 in SOMC, ARU enabled
ATOM0_CH2_RDADDR  = 0x018E007A;     // get data from DPLL CH15 and MCS0 CH1
ATOM0_CH2_CTRL    = 0x9;            // ATOM0 CH2 in SOMC, ARU enabled

ATOM0_AGC_OUTEN_STAT    = 0x0028;   // enable CH1 and CH2 output
ATOM0_AGC_ENDIS_STAT    = 0x0028;   // enable CH1 and CH2

// setup and enable DPLL action calculation for channels 14 and 15
cout << "Enable DPLL PMTR 14, 15..." << endl;
DPLL_CTRL_3       = 0x00C0C000;     // enable actions for CH14 and CH15

// initialize get of lock interrupt
cout << "Enable DPLL GL1I interrupt..." << endl;
tmp = DPLL_IRQ_EN;
tmp |=0x00002000;
DPLL_IRQ_EN = tmp;
```

Code 4.1: PMT Application configuration scenario.

The PMT Application note is an add-on to the micro tick application note AN011. Therefore, functions already implemented for AN011 are reused. **Please note, that you have to install AN011 in order to get this application note AN013 to run.**
The AN011 part is represented in line 01. The AN013 application note starts from line 03.
The DPLL ARU read addresses have to be defined before the DPLL is enabled. Therefore, this is done in lines 07 and 08 while in line 09 the init_dpll() function of AN011 is reused to start the micro tick generation.
After that, the MCS program has to be loaded. This is done with the lines 14 to 16. For sake of simplicity, the MCS code is copied from the generated C-File to the top of the application note.
The MCS channels are enabled and start their execution. For the MCS0 CH1 the interrupt is enabled. This is done to get the end of the application note signaled by this channel. The MCS channels start to initialize the system and then go into their WURM instructions to wait for the CPU trigger. This CPU trigger is generated by the ISR associated with the GL1I interrupt of the DPLL. This interrupt has to be enabled. This is shown in lines 37 to 40. The ISR is described with code fragment 4.2.
The ATOM channels 1 and 2 are setup through lines 23 to 30 and the DPLL actions are enabled with line 34.

3.1.2 AN013 Interrupt Service Routines

The dpll_an013_isr() augments the Interrupt Service Routines (ISR) from application note AN011. As stated above, the GL1I interrupt (ISR No. 113) is used to start the PMT application. This is done by line 15, where the both MCS channels waiting in a WURM instruction are triggered by writing to the MCS global trigger register **MCS0_STRG**.
The ATOM0 CH1 interrupt is used within ISR901 to stop the MCS0 CH0 by canceling the ARU read request, this channel does to the DPLL. Otherwise, the MCS0 CH0 would not detect that the action time is reached and would block in the ARU read command forever.
The ISR 401 part is described in more detail in section 3.2.

```
int dpll_an013_isr(int number) {

  unsigned int ach1ccu0  = 0; // match time of ATOM0 CH1 CCU0
  unsigned int ach1ccu1  = 0; // match time of ATOM0 CH1 CCU1
  unsigned int mch1r4    = 0; // match time of MCS0 CH1 R4
                              // --> received from ATOM0 CH2 CCU0
  unsigned int mch1r5    = 0; // match time of MCS0 CH1 R5
                              // --> received from ATOM0 CH2 CCU1

  cout << "-----> ISR call of interrupt " << dec << number << endl;

  switch (number) {
      case 113:
        cout << "DPLL GL1I interrupt" << endl;
        MCS0_STRG = 0x1; // TRIGGER MCS channel programs to start
        break;
      case 401:
        cout << "MCS0 CH1 IRQ" << endl;
        MCS0_CH1_IRQ_NOTIFY = 0x1;  // clear IRQ
        // check ATOM CH0, CH1 MATCH times
        ach1ccu0 = ATOM0_CH1_SR0;
        mch1r4   = MCS0_CH1_R4;
        if ((ach1ccu0 != mch1r4) || (ach1ccu0==0) || (mch1r4==0)) {
              cout << "ERROR: Wrong TBU_TS0 match time for PMT!" << endl;
              break;
        }
        ach1ccu1 = ATOM0_CH1_SR1;
        mch1r5   = MCS0_CH1_R5;
        if ((ach1ccu1 != mch1r5) || (ach1ccu1==0) || (mch1r5==0)) {
              cout << "ERROR: Wrong TBU_TS1 match time for PMT!" << endl;
              break;
        }
        cout << "AN013 PMT checks successful." << endl;
        break;
      case 901:
        cout << "ATOM0 CH1 interrupt" << endl;
        // disable NOTIFY bit
        ATOM0_CH1_IRQ_NOTIFY = 0x3;
        // Cancel MCS0 CH0 ARU transfer --> will stop MCS0 CH0
        MCS0_RST = 0x100;
      default:
        break;
  }

  return 0;
}
```

Code 4.2: Interrupt Service Routines for PMT application note.

3.2 Automatic check of application

The application note AN013 checks automatically after the action time was reached, if the two DPLL channels have calculated the same result and if the ATOM channels created the actions at the same points in time. This is done in ISR 401 which is shown in code fragment 4.2.
IN SOMC mode, the ATOM channels store the match times in the channels shadow registers SR0 and SR1. These two registers can either be read by CPU or, when the ARU interface of the channel is enabled also by a destination at the ARU.
For AN013, the MCS0 CH0 only reads values from the DPLL and writes these values to the ATOM channel. There, the shadow registers are not read. Therefore, the two match times have to be read by CPU. This is done in lines 21 and 27.
The MCS0 CH1 reads the shadow registers of the ATOM0 CH2 through the ARU. By this construction, the MCS channel blocks until the ATOMs compare match event. The result is stored in the registers R4 and R5. Please see code fragment 2.3 line 19 for this ARU read command. After that, an interrupt is raised by the channel. Please see code fragment 2.3 line 20 for this interrupt generation. This interrupt causes ISR 401 to be called. The two match times are read from the MCS0 CH1 registers R4 and R5 in the code fragment 4.2 lines 22 and 28. These values are compared to the values obtained from the shadow registers R0 and R1 of ATOM0 CH1. They have to be the same and should not be equal to zero (0).