GTM-IP

Application Note AN011 DPLL Micro tick generation

Date: 03.07.2012
(Released)

Robert Bosch GmbH
Automotive Electronics (AE)

LEGAL NOTICE

© Copyright 2012 by Robert Bosch GmbH and its licensors. All rights reserved.

“Bosch” is a registered trademark of Robert Bosch GmbH.

The content of this document is subject to continuous developments and improvements. All particulars and its use contained in this document are given by BOSCH in good faith.

NO WARRANTIES: TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON, EITHER EXPRESSLY OR IMPLICITLY, WARRANTS ANY ASPECT OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING ANY OUTPUT OR RESULTS OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO UNLESS AGREED TO IN WRITING. THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS BEING PROVIDED "AS IS", WITHOUT ANY WARRANTY OF ANY TYPE OR NATURE, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY THAT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IS FREE FROM DEFECTS.

ASSUMPTION OF RISK: THE RISK OF ANY AND ALL LOSS, DAMAGE, OR UNSATISFACTORY PERFORMANCE OF THIS SPECIFICATION (RESPECTIVELY THE PRODUCTS MAKING USE OF IT IN PART OR AS A WHOLE), SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO RESTS WITH YOU AS THE USER. TO THE MAXIMUM EXTENT PERMITTED BY LAW, NEITHER THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, NOR ANY PERSON EITHER EXPRESSLY OR IMPLICITLY, MAKES ANY REPRESENTATION OR WARRANTY REGARDING THE APPROPRIATENESS OF THE USE, OUTPUT, OR RESULTS OF THE USE OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, BEING CURRENT OR OTHERWISE. NOR DO THEY HAVE ANY OBLIGATION TO CORRECT ERRORS, MAKE CHANGES, SUPPORT THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, DISTRIBUTE UPDATES, OR PROVIDE NOTIFICATION OF ANY ERROR OR DEFECT, KNOWN OR UNKNOWN. IF YOU RELY UPON THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, YOU DO SO AT YOUR OWN RISK, AND YOU ASSUME THE RESPONSIBILITY FOR THE RESULTS. SHOULD THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL LOSSES, INCLUDING, BUT NOT LIMITED TO, ANY NECESSARY SERVICING, REPAIR OR CORRECTION OF ANY PROPERTY INVOLVED TO THE MAXIMUM EXTEND PERMITTED BY LAW.

DISCLAIMER: IN NO EVENT, UNLESS REQUIRED BY LAW OR AGREED TO IN WRITING, SHALL THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS OR ANY PERSON BE LIABLE FOR ANY LOSS, EXPENSE OR DAMAGE, OF ANY TYPE OR NATURE ARISING OUT OF THE USE OF, OR INABILITY TO USE THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, INCLUDING, BUT NOT LIMITED TO, CLAIMS, SUITS OR CAUSES OF ACTION INVOLVING ALLEGED INFRINGEMENT OF COPYRIGHTS, PATENTS, TRADEMARKS, TRADE SECRETS, OR UNFAIR COMPETITION.

INDEMNIFICATION: TO THE MAXIMUM EXTEND PERMITTED BY LAW YOU AGREE TO INDEMNIFY AND HOLD HARMLESS THE INTELLECTUAL PROPERTY OWNERS, COPYRIGHT HOLDERS AND CONTRIBUTORS, AND EMPLOYEES, AND ANY PERSON FROM AND AGAINST ALL CLAIMS, LIABILITIES, LOSSES, CAUSES OF ACTION, DAMAGES, JUDGMENTS, AND EXPENSES, INCLUDING THE REASONABLE COST OF ATTORNEYS' FEES AND COURT COSTS, FOR INJURIES OR DAMAGES TO THE PERSON OR PROPERTY OF THIRD PARTIES, INCLUDING, WITHOUT LIMITATIONS, CONSEQUENTIAL, DIRECT AND INDIRECT DAMAGES AND ANY ECONOMIC LOSSES, THAT ARISE OUT OF OR IN CONNECTION WITH YOUR USE, MODIFICATION, OR DISTRIBUTION OF THIS SPECIFICATION, SOFTWARE RELATED THERETO, CODE AND/OR PROGRAM RELATED THERETO, ITS OUTPUT, OR ANY ACCOMPANYING DOCUMENTATION.

GOVERNING LAW: THE RELATIONSHIP BETWEEN YOU AND ROBERT BOSCH GMBH SHALL BE GOVERNED SOLELY BY THE LAWS OF THE FEDERAL REPUBLIC OF GERMANY. THE STIPULATIONS OF INTERNATIONAL CONVENTIONS REGARDING THE INTERNATIONAL SALE OF GOODS SHALL NOT BE APPLICABLE. THE EXCLUSIVE LEGAL VENUE SHALL BE DUESSELDORF, GERMANY.

MANDATORY LAW SHALL BE UNAFFECTED BY THE FOREGOING PARAGRAPHS.

INTELLECTUAL PROPERTY OWNERS/COPYRIGHT OWNERS/CONTRIBUTORS: ROBERT BOSCH GMBH, ROBERT BOSCH PLATZ 1, 70839 GERLINGEN, GERMANY AND ITS LICENSORS.

Revision History

Issue	Date	Remark
0.1	26.08.2011	Initial version
0.2	16.09.2011	Fixes
0.3	06.12.2011	Update for GTM-IP Specification v1.5.0
0.4	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

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 basic principles and usage of the GTM-IP submodule DPLL. Due to the complexity of the DPLL and its various configuration possibilities this application note can only be a starting point for further notes.
This application note starts with an introduction to the principles and explains the various submodules which are involved in the Micro Tick generation for an external input signal.

1.1 System architecture

This section should describe the system architecture of the submodules involved in the micro tick generation. To get in depth information for the other involved submodules please refer to the GTM-IP specification. **Figure 1.1** shows the GTM submodules involved in the micro tick generation.

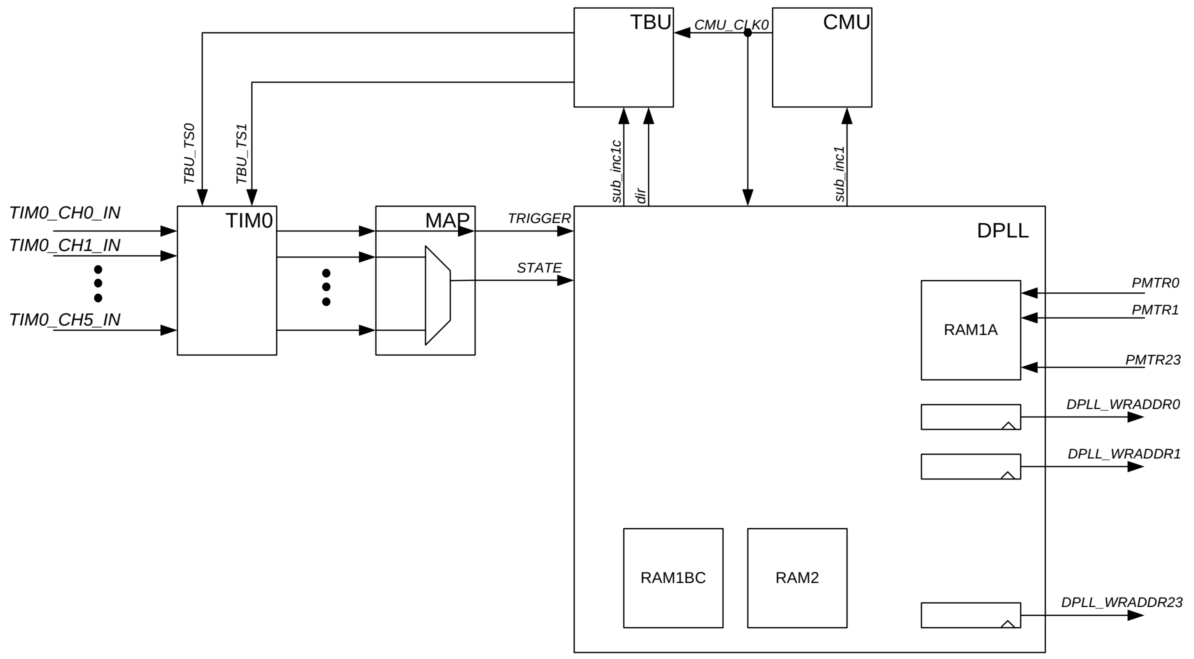


Figure 1.1: Micro tick generation with DPLL.

The DPLL is able to generate a high frequency micro tick signal on behalf of one or more input signals with a lower frequency. These input signals have to be connected to the TIM0 submodule. In principle, there are two possibilities to provide input signals to the DPLL.
The first possibility, also shown in the figure above, is to connect the TIM0 channel 0 via the MAP submodule with the *TRIGGER* input signal of the DPLL and as an option one of the five TIM0 inputs TIM0 channel 1 to channel 5 via a multiplexer implement in the MAP to the *STATE* input of the DPLL.

To generate the micro ticks the DPLL needs information about the timing behaviour of the input signals. This is done with time stamps that are provided by the Time Base Unit (TBU). The TBU generates these time stamps with a clock signal that is generated inside of the Clock Management Unit (CMU).

The DPLL generates the micro ticks at output signals which are connected to the TBU and the CMU. **Figure 1.1** shows the two micro tick signals *sub_inc1* and *sub_inc1c*, where the *sub_inc1c* is connected to the TBU and the *sub_inc1* is connected to the CMU. The difference between *sub_inc1c* and *sub_inc* is that the signal *sub_inc1c* may stop on specific conditions of the associated input signal while the signal *sub_inc* provides continues micro ticks. In general, the DPLL can generate a second group of micro tick signals *sub_inc2c* and *sub_inc2* which are not shown in the figure.

For the micro tick generation and other calculations, the DPLL uses internal ALUs and external RAMs, holding data for the calculations. RAM1a is used for action calculations. Action calculation means that there can be requests coming from the ARU where time points and angle points in the future have to be predicted by the DPLL on behalf of the input signals.

RAM1bc holds calculation parameters for the DPLL, to do calculations for the *TRIGGER* and *STATE* inputs and it holds *STATE* input characteristic values. RAM2 is used to store *TRIGGER* input signal characteristics.

In the subsequent chapters the configuration of the GTM-IP infrastructural components for the DPLL and the DPLL configuration itself are explained.

2 GTM Infrastructure setup

2.1 Overview

For the micro tick generation application the DPLL needs several other GTM-IP submodules that produce useable infrastructure and input signals. Setup of these submodules is described in more detail in the next upcoming sections.

2.2 Clock Management Unit (CMU) Setup

The CMU is responsible to provide clock prescalers for the GTM-IP internal counters. Therefore, it has to be configured for a proper operation of the DPLL. A CMU block diagram can be obtained from **Figure 2.1**.

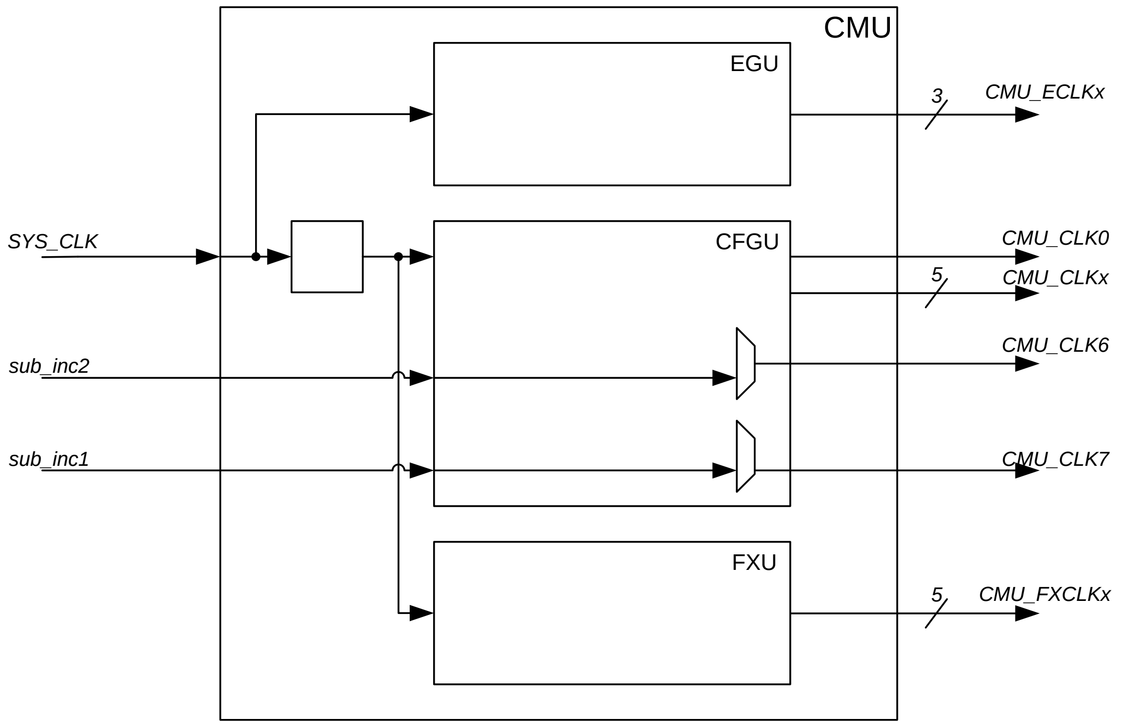


Figure 2.1: CMU Block diagram.

The user has to choose several CMU prescalers and configure these prescalers in a certain manner. First, the TBU needs a clock for the time stamp generation with which the TIM0 input signals are characterized and send to the DPLL. Therefore, the prescaler setup determines the granularity of the time stamps and since the chosen time stamp clock is also feed to the DPLL this prescaler determines the micro tick generation resolution.
A second important clock prescaler is the one connected to the clock signal *CMU_CLK0*. The *CMU_CLK0* frequency is always used by the DPLL to generate missing micro ticks or to correct micro ticks if configured and necessary.

Last, please note, that the two clock signal lines *CMU_CLK6* and *CMU_CLK7* can be used to distribute the micro ticks *sub_inc2* and *sub_inc1* generated by the DPLL to the other submodules of the GTM-IP. **Since these two clock signals are not stopped on specific input signal conditions by the DPLL, these two signal lines should not be provided to the TBU as a clock signal.** For this purpose a second input path to the TBU exists.

2.3 Time Base Unit (TBU) Setup

The Time Base Unit provides common time bases for the GTM-IP. Typically, for a motor management system, the time stamp *TBU_TS0* provides a 24 bit time while *TBU_TS1* provides the angle of the engine for the system.
A TBU block diagram is shown in **Figure 2.2**.
For resolution issues, the *TBU_TS0* time stamp is 27 bit wide. In configuration register TBU_CH0_CTRL bit LOW_RES defines if the lower or the upper 24 bits are provided to other GTM sub-modules. Only for TIM channel 0 and DPLL one can program which inside these sub-modules bunch of 24 bit is used.

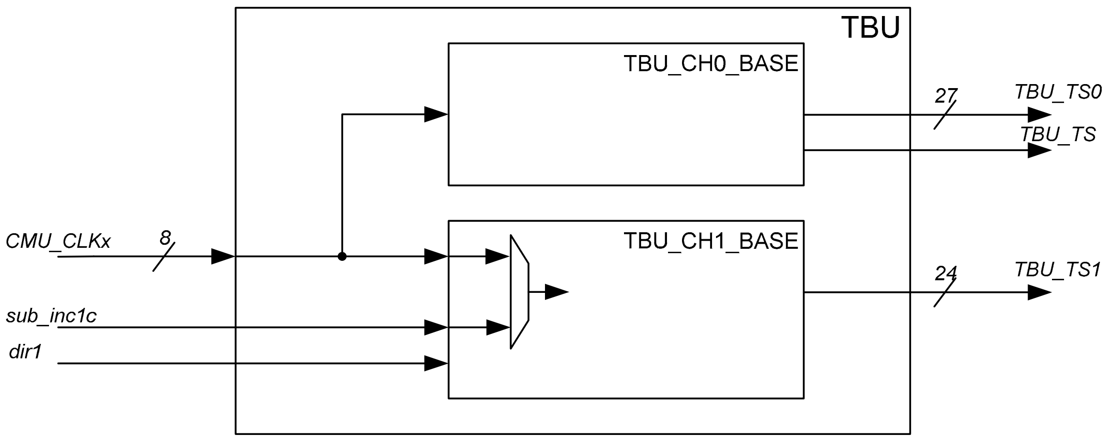


Figure 2.2: TBU Block diagram.

Please note, that for the channel 1 time base the *sub_inc1c* input clock should be chosen by bit CH_MODE of the TBU channels control register. This is done by selecting the *Forward/Backward Counter Mode*. Although the sub_inc1 clock can be selected via *CMU_CLK7*, it is not recommended to use this clock as input since it does not exactly reflect the input behavior of the engine.

2.4 Timer Input Module (TIM0) Setup

The Timer Input Module 0 is dedicated to sample and preprocess the input signals for the DPLL. The TIM consists of channels and inside of the channels there are filter and signal processing units. A schematic of a TIM channel is shown in **Figure 2.3**.

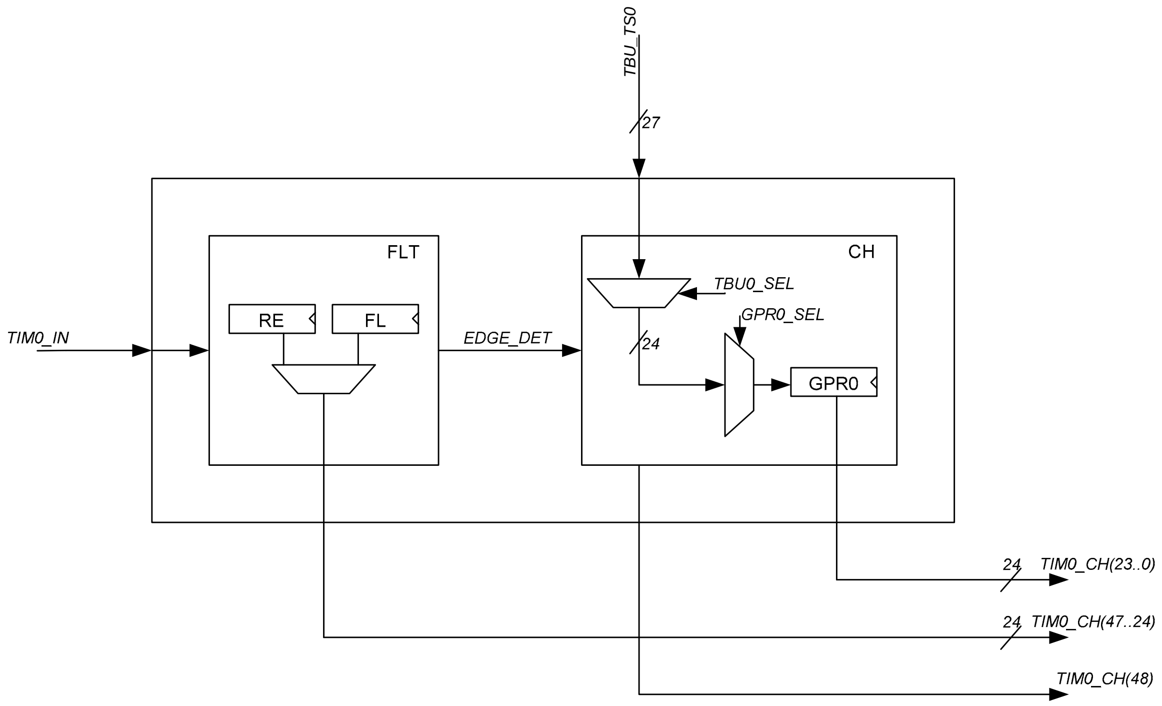


Figure 2.3: TIM Channel block diagram.

The 49 bit wide signal TIM0_CH is connected via the MAP submodule to the DPLL. To work properly, the DPLL needs specific settings inside the corresponding TIM channel. This is on one hand the filtering and on the other hand the time stamp format sampled inside of the TIM channel.

2.4.1 TIM0 Filter configurations

The TIM0 submodule comes with three general filter modes which can be configured individually for each edge. The DPLL is able to do input signal timing corrections on behalf of the filter thresholds for each individual edge. There are several assumptions necessary for these filter corrections. The mechanisms should be shown in Figure 2.4.

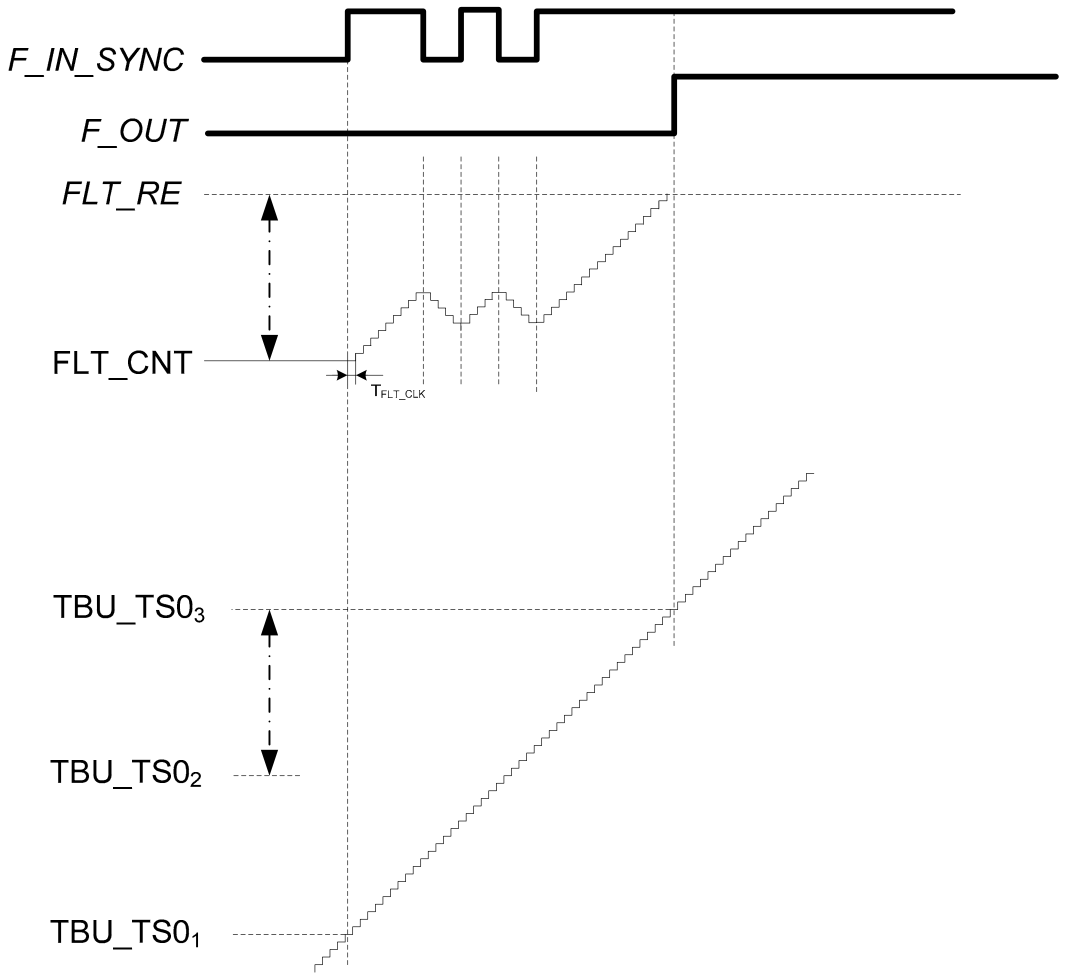


Figure 2.4: DPLL Input signal correction mechanism.

The figure shows a time input filter for the rising edge which is configured in *Up/Down Counter Mode*. The filter threshold is given by value *FLT_RE*. At the bottom of the picture, the running time base TBU_TS0 is shown. As it can be observed, depending on the input signal characteristic, the real edge of the input signal F_IN_SYNC can be at TBU_TS0$_1$. Since the filter delays the input signal until the filter threshold is reached, the DPLL will see the time stamp TBU_TS0$_3$. This introduces an error on the input signal.
The DPLL can correct this error to a certain amount by subtracting the filter threshold from the time stamp TBU_TS0$_3$. This will result in a new corrected time stamp TBU_TS0$_2$ for the rising edge which is closer to the real time stamp TBU_TS0$_1$. The TIM channel sends the filter threshold value together with each edge time stamp to the DPLL (in the case of Figure 2.4 the values FLT_RE and TBU_TS0$_3$ are send to the DPLL).
The filter correction can be enabled in the DPLL with the IDT and IDS bits in the DPLL_CTRL_0 register. There, with the IFP bit it can also be defined if the TIM filter counts on behalf of CMU_CLK ticks or in behalf of sub_inc ticks.
When the filter correction is done on a CMU_CLK tick base, it is important to configure the filter input clock to be the same as the TBU_TS0 time stamp clock.

Otherwise, the two frequencies would not fit together and the time stamp correction within the DPLL would deliver wrong results.
The same holds if the TIM channel time stamp sampling is done with the low resolution, while the filter runs with a high resolution.
Another important issue is, that when the TIM filter is configured in *Immediate Edge Propagation Mode*, the filter correction inside of the DPLL has to be disabled. Otherwise, the DPLL will use the filter disable windows, programmed also as filter threshold value, to correct the time stamp. This will result in a new time stamp which will be located before the immediate propagated edge.

2.4.2 TIM0 Signal sampling configuration

For the DPLL operation, the DPLL needs the time stamps of each incoming edge. Therefore, the corresponding TIM0 channel should be configured to capture at each incoming edge in the GPR0 register the associated time stamp. Inside of the TIM0_CH[x]_CTRL register there are three configuration bits which have to be set accordingly:

ISL (Bit 14)	Ignore signal level: This bit has to be set, to force the TIM0 channel to react on each incoming edge.
GPR0_SEL (Bit 9:8)	Selection for GPR0 register: These two bits have to be set to “00” to sample the TBU_TS0 in the register. This value is transmitted to the DPLL when a valid edge occurs.
TBU0_SEL	TBU_TS0 bits input select: This bit has to be set according to the requested time stamp resolution.

For a more detailed description of the TIM input channel, please refer to the GTM-IP specification.

2.5 MAP Setup

When the signal preprocessing inside the TIM0 channel is done, the 49 bit data is transferred to the DPLL via the MAP module. Inside of the MAP module two data path exist. The first path goes from TIM0 channel 0 to the DPLL *TRIGGER* input directly. The second path for the DPLL STATE input has to be chosen out of the five TIM0 channel 1 to 5 inputs via a multiplexer.

3 DPLL Configuration

3.1 Overview

The DPLL is a highly configurable dedicated submodule of the GTM-IP that can transform an input signal frequency in a higher frequent signal called micro tick further on. The DPLL can generate one micro tick signal that is dependent on *TRIGGER* and *STATE* input, where *TRIGGER* and *STATE* have some kind of relationship to each other. The other possibility is to generate two independent micro tick signals from two independent *TRIGGER* and *STATE* input signals.

Since there are several different signal combinations and characteristics possible, there are a lot of configuration parameters which can be configured by the CPU. Besides the high configurability, there are also a lot of calculations done inside of the PWM which need a high amount of local variables. Since the implementation of registers for these local variables would have introduced high costs, local variables are located inside of a RAM whenever possible. Therefore, the DPLL needs to have access to the RAM during calculation.

The DPLL can be divided in several functional blocks which are depicted in Figure 3.1 and are described further on in a little bit more detail. For a more detailed description of the DPLL and to use all the features of the DPLL please refer to the GTM-IP specification.

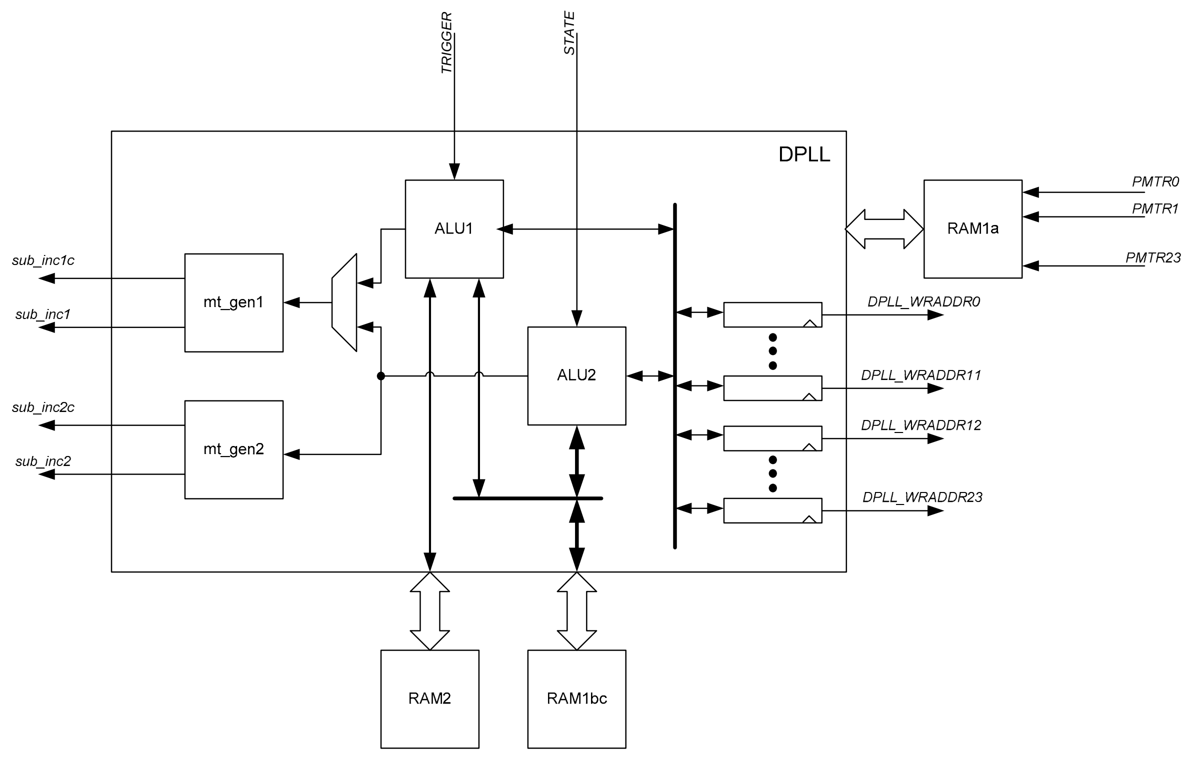


Figure 3.1: DPLL Block diagram.

There are several subunits inside of the DPLL. The micro tick generation can be done with two independent micro tick generation units *mt_gen1, 2*. Since the two input signals *TRIGGER* and *STATE* can arrive at the DPLL input totally independent, there are two independent data paths and ALU subunits implemented which calculate parameters for the micro tick generators and action values for the rest of the GTM-IP system.

Local variables, action parameters and system characterization data are located in three independent RAM modules RAM1a, RAM1bc and RAM2. RAM1a holds data which is needed for the action calculation. RAM1bc holds data for local variables and for the *STATE* input signal characterization and RAM2 holds data for the *TRIGGER* input signal characterization.

3.2 Micro tick generation

The micro ticks for the input signals *TRIGGER* and *STATE* are generated inside of the two subunits mt_gen1 and mt_gen2. The micro ticks are distributed over the four signal lines *sub_inc1*, *sub_inc1c*, *sub_inc2* and *sub_inc2c*. The micro tick generation is done with a 24 bit adder which generates a tick whenever the 24 bit accumulator register overflows. The mt_gen0 subunit principle is shown in Figure 3.2. For mt_gen1 the behaviour is accordingly.

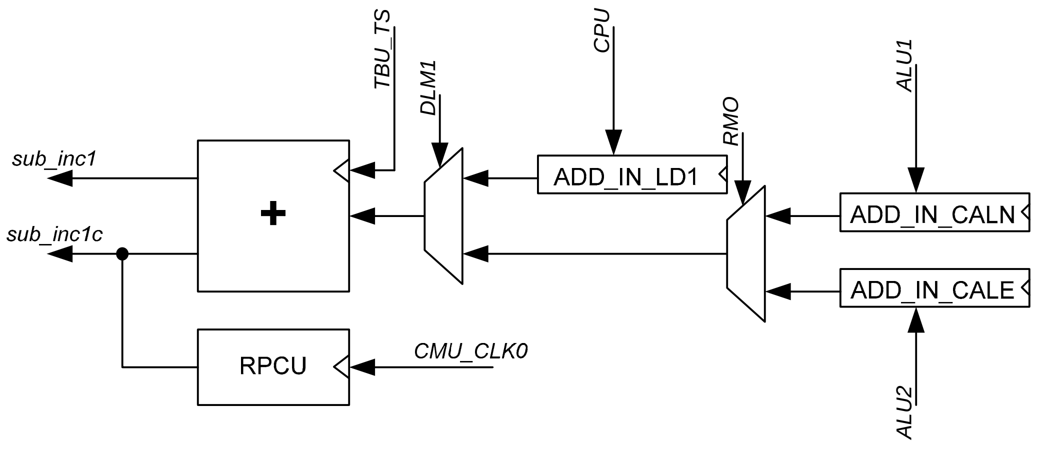


Figure 3.2: mt_gen0 subunit principle.

There are two input path for the adder. One path can be controlled via CPU by loading the adder values into the ADD_IN_LD1 register. This path can be enabled via the DLM1 bit inside of the DPLL_CTRL_1 register. The other path is controlled by the DPLL internal logic, where the two ALUs calculate the two adder values independently on behalf of their input signals. ALU1 calculates ADD_IN_CALN and ALU2 calculates ADD_IN_CALE. Which one of the two adder values is used can be controlled by the CPU by setting RMO bit in the DPLL_CTRL_1 register.

For the adder value calculation it is important, that the adder is clocked with the time stamp clock *TBU_TS* which comes from the TBU channel 0. The other input clock *CMU_CLK0* is used to generate fast pulses, if there are some missing micro ticks or

a direction change is detected. Therefore, the TBU channel 0 input clock as well as the *CMU_CLK0* clock have to be chosen carefully.
There is also a difference, between the output micro ticks generated. This should be demonstrated with Figure 3.3.

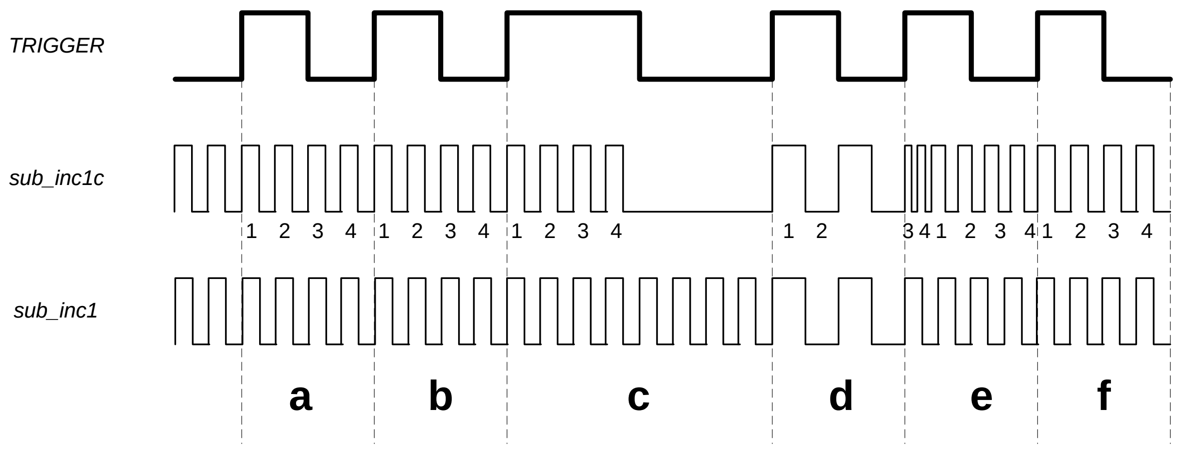


Figure 3.3: Compensated and uncompensated micro tick generation.

The DPLL is configured to generate 4 micro ticks per input tick (one tick from rising to rising edge) on the *TRIGGER* input signal. The four micro ticks are generated for the intervals ***a***, and ***b*** correctly. The DPLL predicts for interval ***c*** the same adder value. Unfortunately, the input signal frequency for interval ***c*** decreases.
When the DPLL is programmed in *Automatic End Mode*, which can be controlled by DMO bit in the DPLL_CTRL_1 register, the *sub_inc1c* output ticks will stop after four ticks were generated. For the next interval ***d*** another adder value is calculated which results in a smaller frequency of the *sub_inc* signal. Now, since the input signal accelerates, there are not enough micro ticks generated.
To compensate this, the DPLL offers two possibilities. One possibility is to distribute for the next interval ***e*** evenly six micro ticks or to generate two micro ticks fast and the regular 4 micro ticks evenly in the next interval. This behaviour can be configured with the COA bit in the DPLL_CTRL_1 register. If the micro ticks should be generated fast, the *CMU_CLK0* is used as tick frequency.
The aforementioned micro tick generation holds for signal *sub_inc1c*. As can be seen from the figure, *sub_inc1* is always generated with the frequency calculated from the last increment duration. Therefore, the *sub_inc1* ticks do not reflect the physical position of the *TRIGGER* input signal.

3.3 DPLL RAM Organization

3.3.1 Overview

As mentioned above the DPLL has three associated RAM blocks which can be accessed independently by the DPLL. The RAM1a is used for action calculation and can only be accessed by CPU when the DPLL is disabled. RAM1bc is used for local

variable storage during DPLL calculations and for the storage of the *STATE* input signal profile. RAM2 is used to store *TRIGGER* input signal related data.
The DPLL locates data inside of the RAMs with internal pointers. Besides the three different physical RAMs, each RAM is divided into several regions, each region with its own RAM pointer. The knowledge of the pointers is important for the usage of the DPLL. The following table describes the pointers for RAM1bc regions and associated pointers:

Region	**Description**	**Size**	**Pointer**
RAM1c1	Contains reciprocal values of durations of *STATE* signal	2*(SNU+1-SYN_NS)	APS
RAM1c2	Contains time stamps of *STATE* input signal	2*(SNU+1)	APS_1c2
RAM1c3	Contains *STATE* input signal profile	2*(SNU+1-SYN_NS)	APS_1c3
RAM1c4	Contains history of durations of *STATE* signal	2*(SNU+1-SYN_NS)	APS

It is important to know that the DPLL manipulates these pointers automatically after enable and when a valid *STATE* is detected. Nevertheless, the CPU has to observe the *STATE* input signal and set the APT_1c3 pointer according to the actual profile position which corresponds to the input. The DPLL will set than the LOCK1 bit in the DPLL_STATUS register to signal that the DPLL now can judge on the input signal on behalf of the profile information given by the user in RAM region RAM1c3.

The following table describes the pointers for RAM2 regions and associated pointers:

Region	**Description**	**Size**	**Pointer**
RAM2a	Contains reciprocal values of *TRIGGER* signal duration (RDT_T[i])	2*(TNU+1-SYN_NT)	APT
RAM2b	Contains time stamps of *TRIGGER* input signal (TSF_T[i])	2*(TNU+1)	APT_2b
RAM2c	Contains *TRIGGER* input signal profile (ADT_T[i])	2*(TNU+1-SYN_NT)	APT_2c
RAM2d	Contains history of durations of *TRIGGER* signal (DT_T[i])	2*(TNU+1-SYN_NT)	APT

It is important to know that the DPLL manipulates these pointers automatically after enable and when a valid *TRIGGER* is detected. Nevertheless, the CPU has to observe the *TRIGGER* input signal and set the APT_2c pointer according to the actual profile position which corresponds to the input. The DPLL will set than the LOCK1 bit in the DPLL_STATUS register to signal that the DPLL now can judge on the input signal on behalf of the profile information given by the user in RAM region RAM2c.

3.3.2 Calculation of the RAM2 pointers

In principle the RAM is organized into four equidistant regions, which are defined by the DPLL_OSW and DPLL_AOSV_2 registers. Depending on the DPLL_OSW register OSS bit field, the DPLL_AOSV_2 register is set by the hardware. The four possible settings are shown in the following table:

OSS	DPLL_AOSV_2
0x0	0x06040200
0x1	0x0C080400
0x2	0x18100800
0x3	0x30201000

This register map is important, when the *TRIGGER* profile has to be stored inside of the RAM region 2c.
The pointer value has always to be calculated starting from the offset of RAM region 2c, and the pointer always uses word aligned addresses. This means that the first entry in RAM region 2c has a pointer address APT_2c = 0x0, the second entry has APT_2c = 0x4 and so forth.

3.3.3 TRIGGER and STATE Input signal profiles

For proper operation and prediction of the DPLL, the user has to characterize the TRIGGER and/or STATE input signals in the profile regions of RAM1bc and RAM2. These are the regions RAM1c3 and RAM2c. The CPU has to correct the associated pointers for these regions when the actual position is known. The DPLL will then work on this profile. An example profile is shown in Figure 3.4.

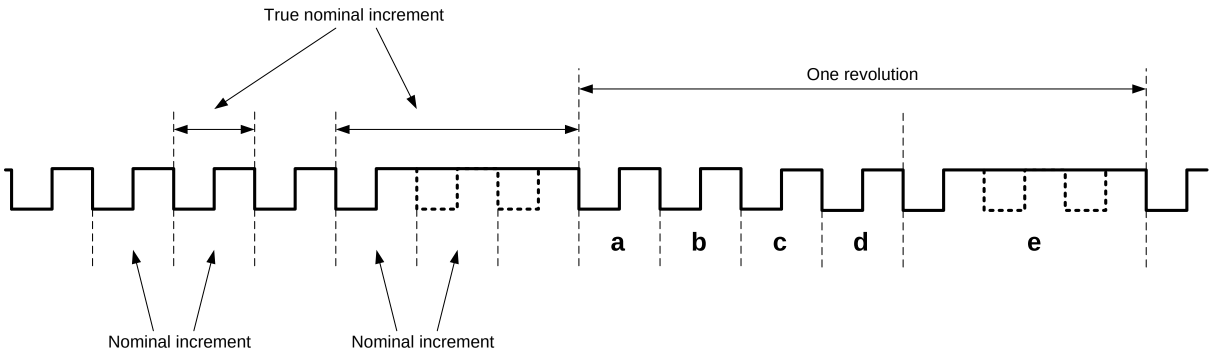


Figure 3.4: Compensated and uncompensated micro tick generation.

The figure shows two revolutions of a tooth wheel, where there two tooth are missing. The falling edges are defined as the active trigger slopes which define a valid tooth. The active edge has to be defined inside of the DPLL_CTRL_1 register. These active edges define the *true virtual increments*. Nevertheless, for the profile to work, the revolution has to be divided into equidistant *nominal increments*.

The user now has to define the profile in the RAM region RAM1c3 and/or RAM2c on behalf of the *true nominal increments* and their duration. For the example in Figure 3.4 this results in the RAM entries (shown as RAM2 entries):

```
ADT_T[0] = 0x10000      // a
ADT_T[1] = 0x12000      // b
ADT_T[2] = 0x10000      // c
ADT_T[3] = 0x10000      // d
ADT_T[4] = 0x30000      // e
```

The so called adapt values ADT_T[x] represent the profile of the tooth wheel. For each *true nominal increment*, the number of *nominal increments* is stored in the NT bit field. For ADT_T 0 to 3 there is one *nominal increment*. For ADT_T[4] three nominal increments are present. In addition for the *true nominal increment* ADT_T[1] there is a user defined interrupt specified which will cause an interrupt when this tooth is detected and the interrupt is enabled. In total there can be five user specific interrupts defined.
In addition, physical deviations of a tooth can be specified in the lower 13 bits of the RAM locations.

3.4 Action calculations

Actions are positions and times in the future which could be predicted by the DPLL on behalf of the past behaviour of the *TRIGGER* and/or *STATE* signal. This topic should not be discussed in this application note.

4 Sample implementation

4.1 Use case

4.1.1 Application parameters

For this application note the *TRIGGER* signal input path should be used to generate a specified number of micro ticks on the *sub_inc1c* signal line. As an input signal characteristic, a 60-2 tooth wheel should be used, which has to do two revolutions for one FULL_SCALE. Therefore, the number of nominal increments in the FULL_SCALE range is 120 events.

The DPLL should automatically generate 600 micro ticks per nominal increment. The micro ticks should be generated in *Automatic End Mode*. There are no adaptation values used and there is no emergency signal input (*STATE* path) in place.

The *TRIGGER* input signal reflects a tooth wheel with a constant speed of 6000 revolutions per minute. The input signal should be sampled with high resolution at the TIM0 input.

The GTM-IP system clock is supposed to be 100MHz.

4.1.2 Testbench setup

For the sample application several GTM-IP infrastructural submodules have to be used direct and indirect. A direct use of a submodule is meant, when the submodule would also be needed for a real world application. Indirect use means, the submodule is used to generate input stimuli or it observes the functionality. For this application note for example a PSM → ATOM path is used to generate the TRIGGER input signal characteristic.

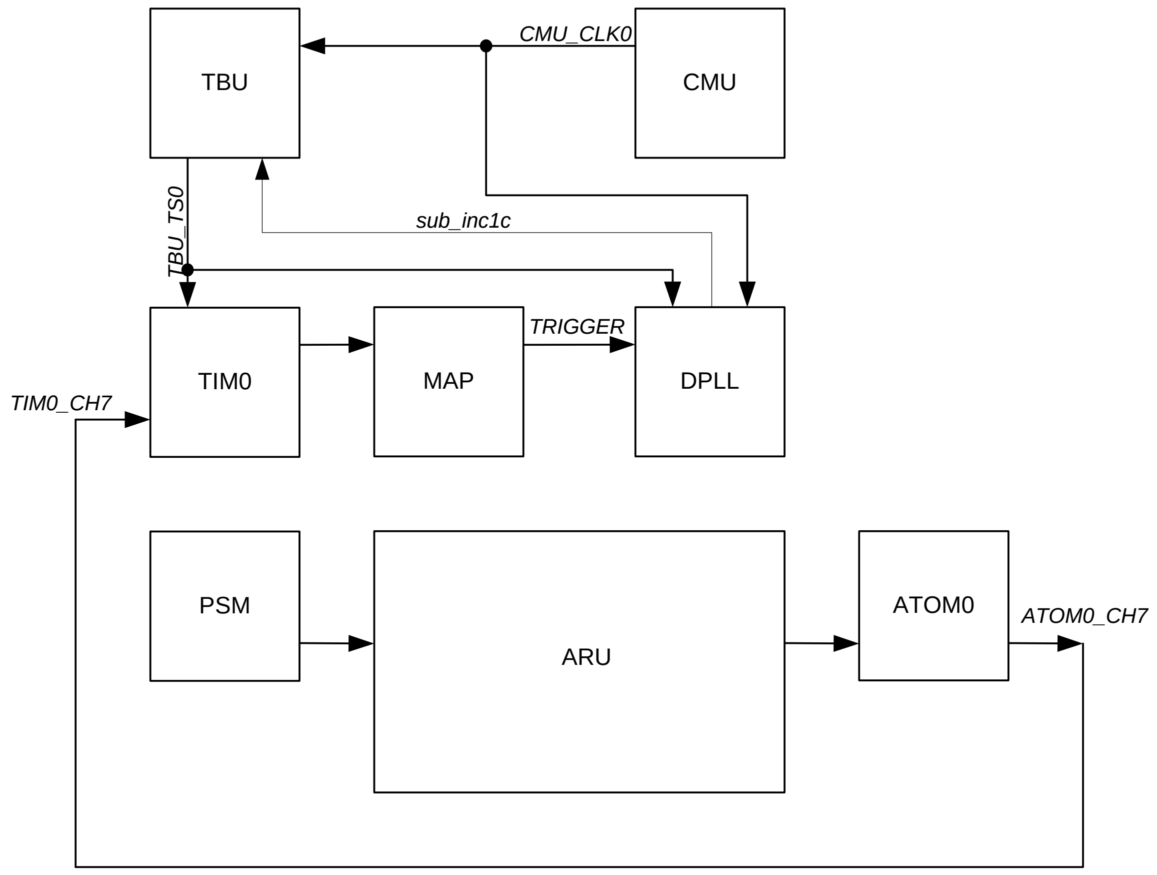


Figure 4.1: Testbench setup.

There are two possibilities to generate input stimuli for the GTM-IP. One is to use the CHKSIG-IFS module and specify the input signal characteristic with CHKSIG commands. The other is to generate the input stimuli with a GTM-IP output, in this case with ATOM channel 7 and to connect the GTM output port with the input port via a simple CHKSIG loop. The command sequence to do this is:

```
CHK ADD_INSTR 0 WSE #ATOM0_OUT7 FE      -- wait for falling edge
CHK ADD_INSTR 0 SSL #TIM0_IN7 LO        -- set input to '0'
CHK ADD_INSTR 0 WSE #ATOM0_OUT7 RE      -- wait for rising edge
CHK ADD_INSTR 0 SSL #TIM0_IN7 HI        -- set input to '1'
CHK ADD_INSTR 0 JMT 0 -4                -- JMP forever
ALL SYNC ALL
-- generate a DPLL input signal
CHK ENABLE 0 ON
ALL SYNC ALL
```

Code 4.1: DPLL signal routing from ATOM Channel 7 → TIM Channel 7.

The PSM channel holds the TRIGGER signal input characteristic as a sequence of PWM values. This sequence is feed to the ARU, while the PSM channel is configured

in *Ring Buffer Mode*. The ATOM channel is configured in SOMP mode with a duty cycle level of ‘0’.
For the *TRIGGER* input signal to work, the TIM0 input channel 7 has to be routed to the TIM0 input channel 0 via the TIM0 internal multiplexers at the input ports.
For the DPLL to work, one has to configure the TIM0 channel 0, the MAP, the CMU, TBU and the DPLL itself.

```
void tb_setup(void)
{
  int i = 0;
  // setup PSM channel 0
  FIFO0_CH0_CTRL        = 0x1;      // PSM operates in Ring Buffer Mode
  F2A0_CH0_STR_CFG      = 0x60000;  // transfer both words to ARU
  for (i=0; i<57; i++) {
      AFD0_CH0_BUFFACC = 3200;      // 57 times normal tooth
      AFD0_CH0_BUFFACC = 1600;      // 50% duty cycle
  }
  AFD0_CH0_BUFFACC      = 9600;     // 58 th tooth + 2 tooth gap
  AFD0_CH0_BUFFACC      = 1600;

  F2A0_ENABLE = 0x2;                // enable channel 0

  // configure ATOM0 channel 7 output
  ATOM0_CH7_RDADDR      = 0x051;    // get data from PSM channel0
  ATOM0_CH7_CTRL        = 0xA;      // ATOM channel 7 in SOMP, ARU enabled

  ATOM0_AGC_GLB_CTRL    = 0x80000000; // enable update for shadow registers
  ATOM0_AGC_OUTEN_STAT  = 0x8000;     // enable channel 7 output
  ATOM0_AGC_ENDIS_STAT  = 0x8000;     // enable channel 7
}
```

Code 4.2: Testbench setup for PSM and ATOM Setup.

The CMU is needed for the ATOM channel to work. But the CMU is set up in the main application code.

4.2 Infrastructural submodules configuration

4.2.1 CMU Configuration

For the DPLL application a resolution of 20 MHz is chosen for the CMU_CLK0. This clock is used for the TRIGGER input signal generation, the TBU time stamp generation and the DPLL fast update frequency for the micro ticks:

The CMU Clock setup procedure is shown in Code sample 4.3:

```
void init_cmu(tCmu_param cParam, unsigned int en_msk)
{
      // setup global clock divider
      CMU_GCLK_NUM      = cParam.gclk_num;
      CMU_GCLK_DEN      = cParam.gclk_den;

      // setup the CMU_CLKx prescalers
      CMU_CLK_0_CTRL    = cParam.clk_0;
      CMU_CLK_1_CTRL    = cParam.clk_1;
      CMU_CLK_2_CTRL    = cParam.clk_2;
      CMU_CLK_3_CTRL    = cParam.clk_3;
      CMU_CLK_4_CTRL    = cParam.clk_4;
      CMU_CLK_5_CTRL    = cParam.clk_5;
      CMU_CLK_6_CTRL    = cParam.clk_6;
      CMU_CLK_7_CTRL    = cParam.clk_7;

      // enable the clock prescalers
      CMU_CLK_EN        = en_msk;
}
```

Code 4.3: CMU Setup.

Please note that to enable the CMU channels the GTM-IP double bit enable/disable mechanism has to be applied on the CMU_CLK_EN register. For a detailed description please refer to the corresponding submodule specification.

4.2.2 TBU Configuration

The GTM-IP Time Base Unit has to be configured to provide on the TBU channel 0 time stamps to characterize the DPLL *TRIGGER* input signal and to provide on the channel 1 the micro tick time base provided by the DPLL.
For sake of simplicity, for the time base channel 0 the high resolution is chosen. For applications more complex it could be possible that a low resolution must be chosen for channel 0. Then, most of the GTM-IP submodules work with the lower resolution time stamps, while the DPLL and TIM0 can still work with the high resolution time base.
It is important to know, that there is a distinct micro tick input for the TBU directly driven by the DPLL. This direct input has to be configured to get a correct micro tick time base, since the DPLL only controls the compensated micro ticks to be in sync with the *TRIGGER*/*STATE* input signal.
The configuration code is shown in Code sample 4.4:

```
void init_tbu(void)
{
      // setup TBU channel 0
      TBU_CH0_CTRL = 0x0;     // no LOW_RES, choose CMU_CLK0
      // setup TBU channel 1
      TBU_CH1_CTRL = 0x1;     // Up/Down counter mode; sub_inc1c is chosen

      // enable TBU channels 0 and 1
      TBU_CHEN     = 0xA;
}
```

Code 4.4: TBU Setup.

Please note that to enable the TBU channels the GTM-IP double bit enable/disable mechanism has to be applied on the TBU_CHEN register. For a detailed description please refer to the corresponding submodule specification.

4.2.3 TIM0 Configuration

There is one register important for the TIM0 channel 0 configuration. This is the channels control register where the channels input multiplexer, the channels operation mode and the filter strategy can be defined.
Due to the special setup of the testbench, the channels input multiplexer has to be configured to provide the TIM0 channel 7 input to channel 0. The configuration of the multiplexer has to be done in a separate write operation, before enabling of the channel takes place. Otherwise, the channel could determine a signal transition if the TIM0_CH7_IN signal line and the TIM0_CH0_IN signal have different initial signal levels.
It is assumed that for this application note no filtering is necessary for the input signal. The channel is configured to sample the high resolution time base TBU_TS0 for each incoming edge in the GPR0 register.
Code sample 4.5 shows the configuration code sequence:

```
void init_tim0_ch0(void)
{
      // setup and enable TIM0 channel 0
      TIM0_CH0_CTRL = 0x00000040;  // set CICTRL MUX first
      TIM0_CH0_CTRL = 0x00004045;   // CH_EN, TIEM, CICCTRL=1 (ch7 in),
                                    // GPR0_SEL=TBU_TS0, ISL=1 (Both edge)
}
```

Code 4.5: TIM0 Channel 0 setup.

4.2.4 MAP Configuration

For this application note nothing needs to be done in the MAP submodule, since the TIM0 channel 0 input signal is connected to the DPLL *TRIGGER* input by default.

4.3 DPLL Configuration

This application note should show the basic operation principle of the DPLL submodule. Therefore, only a lean configuration is described, where only the a few registers of the DPLL are configured at all.
Basic operation in this context means, that the DPLL generates micro ticks on behalf of a TRIGGER input signal with 50% low level duty cycle, where the input signal characteristic reflects a 60-2 tooth wheel. There should be no adaptation of the tooth be considered and also there should no action generation take place.
The following table should give an overview about the parts of the DPLL that have to be configured:

DPLL part	Description
DPLL_CTRL_0	Contains input signal characteristics, like *TRIGGER/STATE* event characteristic, input filter characteristic etc.
DPLL_CTRL_1	Contains DPLL configuration for operation mode, micro tick generation, time stamp resolutions, etc.
DPLL_APT_2c	Actual RAM pointer address for RAM region 2c. This pointer has to be written in or after synchronization condition was met. In this application note, the pointer is written in the ISR `void isr_toothdet`(void).
DPLL_THMI	Minimum time to the next inactive *TRIGGER* slope. The time should be given in number of time stamp ticks TBU_TS.
RAM2c	This RAM region has to be initialized with the *TRIGGER* signal input profile for FULL_SCALE. Please note, that there could be a RAM2 initialization after reset. Therefore, the programmer has to wait until this RAM initialization ended.
DPLL_TOV	Timeout value for the actual *TRIGGER* slope. This value has to be provided to the DPLL since otherwise, the DPLL would generate timeout events after the first valid *TRIGGER* event.

Since there is no STATE signal input for this application note, the RAM region RAM1c3 has not to be configured at all.
The DPLL configuration is shown in code sample 4.6:

```
void init_dpll(void)
{
      unsigned int ram_ini_v = 0;
      unsigned int i = 0;
      gtm_ptr p;

      // initialize RAM; make sure that no RAM initialization takes place
      // in parallel
      ram_ini_v = DPLL_RAM_INI;
      while(ram_ini_v) { // we have to wait until RAM initialization ends
        ram_ini_v = DPLL_RAM_INI;
      }

      p = &DPLL_RR2;
      p = p + (0x00000400/4);
      for (i=0; i<57; i++){   // file profile for regular tooth
        p[i] = 0x10000; // first HALF SCALE
      }
      p[57] = 0x30000;   // 58th is special one!
      p[58] = 0x12000;   // gen TINT0 IRQ at the 59th tooth in FULL_SCALE
      for (i=59; i<116; i++){       // file profile for regular tooth
        p[i] = 0x10000; // second HALF SCALE
      }
      p[116] = 0x30000;

      DPLL_IRQ_EN = 0x00040000;  // enable TINT0 interrupt
      // configure timeout value for actual TRIGGER slope
      DPLL_TOV    = 0x780;

      // configure DPLL control registers
      DPLL_CTRL_0 = 0x403B0257;
      DPLL_CTRL_1 = 0x80020000;  // configure TRIGGER input charact. first
      DPLL_CTRL_1 = 0x80020012;  // now enable the DPLL

}
```

Code 4.6: DPLL configuration.

4.3.1 DPLL Control 0 register configuration

DPLL_CTRL_0 register contains input *TRIGGER* and *STATE* signal characteristics as well as the number of micro ticks that should be generated between two *TRIGGER* events. The following table shows the DPLL_CTRL_0 register 0 with meaning and initialization value for this application note.

Bit field	Description	Application note configuration value
MLT	Number of micro ticks between two TRIGGER events. MLT+1. Set to 599 to generate 600 micro ticks.	599
IFP	Filter value resolution (time or position related). For this application note don't care because filter is not used.	0
SNU	Number of *STATE* events in HALF_SCALE. For this application note don't care because *STATE* input is not used.	0
TNU	Number of nominal *TRIGGER* events in HALF_SCALE. We have a 60-2 tooth wheel, which means that there are 60 nominal *TRIGGER* events per HALF_SCALE (one revolution).	59+1
AMS	Adapt the tooth of *STATE* due to physical constraints. For this application note don't care because *STATE* input is not used.	0
AMT	Adapt the tooth of *TRIGGER* due to physical constraints. For this application note no adapt values are taken into account.	0
IDS	Take input delay introduced by TIM0 filter into account for *STATE*. For this application note don't	0

	care because *STATE* input is not used.	
IDT	Take input delay introduced by TIM0 filter into account for *TRIGGER*. For this application note don't care because filter is not used.	0
SEN	*STATE* input enable. For this application note don't care because *STATE* input is not used.	0
TEN	*TRIGGER* input enable.	1
RMO	Configure which signal should be used for micro tick generation. This is the TRIGGER only for this application so set to '0'.	0

With the above mentioned configuration parameters the DPLL_CTRL_0 register results in: DPLL_CTRL_0 = 0x403A257;

4.3.2 DPLL Control 1 register configuration

DPLL_CTRL_1 register contains configuration bits for the DPLL operation. There, the behavior of the micro tick generators, the number of virtual increments and the characteristic of the time stamps is defined.
For this application note, the *STATE* signal and time stamp characteristics are not of importance. The following register bit fields have to be configured:

Bit field	Description	Application note configuration value
DMO	This bit defines the DPLL operation mode for micro tick generation. For this application mode automatic end mode is chosen.	0
DEN	DPLL enable bit. The DPLL is enabled immediately by the application.	1
COA	Correction strategy for missing micro ticks, when the *Automatic End Mode* is chosen for micro tick generation. For this application note, the missing micro ticks should be generated with the CMU_CLK0 clock frequency.	0
PIT	Plausibility window resolution definition. A time related plausibility window duration is considered.	1
SGE1	This bit enables the sub_inc1 and sub_inc1c output signal line for micro tick generation. The micro tick generation should start right after the synchronization condition is detected. Therefore, at the beginning, this bit is set to zero.	0
DLM1	In Direct Load Mode, the micro tick frequency is controlled by the CPU. For this application note the micro tick frequency should be calculated by the DPLL itself.	0
PCM1	No pulse correction values are considered for this	0

	application note.	
SYN_NT	This bit field summarizes the total number of virtual increments in a HALF_SCALE for the *TRIGGER* input signal. In this application note, there are two missing tooth per revolution.	2
TSL	The active TRIGGER slope has to be defined by these two bits. For this application note, the falling edges are the relevant ones.	2

With the above mentioned configuration parameters the DPLL_CTRL_1 register results in: DPLL_CTRL_1 = 0x80020012;
Since the DPLL_CTRL_1 register bits 11 to 20 and 24 to 31 are write protected in case the DPLL is enabled, these bit field regions have to be written in an independent write access before the DPLL is enabled.

4.4 Application implementation

4.4.1 Main thread implementation

The DPLL application note consists of several C functions that have to be scheduled at specific points in time for the DPLL to run properly. This C functions were described in the previous sections.
The C functions are called within one main routine `dpll_an011()`. The testbench setup C function `tb_setup()` is used to generate the TRIGGER input signal. This routine is put at the beginning, because it is not directly related to the DPLL micro tick generation with the GTM.
After `tb_setup()`, the CMU, TBU and the TIM0 channel 0 have to be initialized. Due to the nature of this application note, the MAP submodule needs not to be configured. The DPLL configuration is done in the C function `init_dpll()`.
After the initialization was done, the rest of the application is controlled via the interrupt service routine for TIM0 channel0 interrupt.
The code for the main thread is shown in Code fragment 4.7.

```
//--------------------------------------------------
// main thread:

int dpll_an011()
{
      // required declaration for HAL
      gtm_ptr p;
      // CMU configuration structure
      theCmu_param theCmuConfig;

      int error = 0;

      // testbench setup
      tb_setup();

      // determine underlying hardware
      dut_is_rm   = GTM_REV;
      dut_is_rm >>= 20;
      if (dut_is_rm == 0x100)  // GTM-RM identified by revision ID 0x100…
        dut_is_rm = 1;
      else
        dut_is_rm = 0;

      theCmuConfig.gclk_num = 0xFFFFFF;   // define 100 MHz clock
      theCmuConfig.gclk_den = 0xFFFFFF;
      theCmuConfig.clk_0    = 0x4;        // define 20 MHz clock

      // configure CMU
      init_cmu(theCmuConfig, 0x2);  // configure and enable CMU_CLK0

      // configure TBU
      init_tbu();

      // configure TIM0 channel 0
      TIM0_CH0_IRQ_EN   =   0x1;          // enable NEWVAL IRQ
      TIM0_CH0_CTRL     =   0x00000040;   // set CICTRL MUX first
      TIM0_CH0_CTRL     =   0x00004045;   // configure and enable channel

      // configure DPLL
      init_dpll();
}
```

Code 4.7: Main thread for DPLL micro tick generation application.

4.4.2 Application note hookup

This application note can be integrated into the GTM Reference model code by declaring and installing the main application thread and the interrupt service routine code to the model. This installation is done within the implementation file gtm_main.c. The necessary code sequences are shown in code example 4.8:

```
int dpll_an011();
int dpll_an011_isr(int);

int gtm_main(int argc, char* argv[]) {

     …
     DEFINE_TEST_FCT(dpll_an011);
     …
}

int gtm_isr(int argc, char* argv[]) {

     …
     DEFINE_ISR_FCT(dpll_an011_isr);
     …
}
```

Code 4.8: Code sequences for application note hookup in GTM-RM.

For this application note, there is one major entry point for interrupts, which is defined by `void dpll_an011_isr(int)`.
This ISR is called from the interrupt system with the GTM-IP interrupt number. This interrupt numbers can be obtained from the GTM-IP Testbench guide. The relevant interrupt numbers for this application note are shown in the following table:

GTM-IP IRQ No.	Interrupt naming	ISR
118	DPLL_TIE0	`void isr_lockdet(void)`
200	TIM0_CH0_IRQ	`void isr_toothdet(void)`

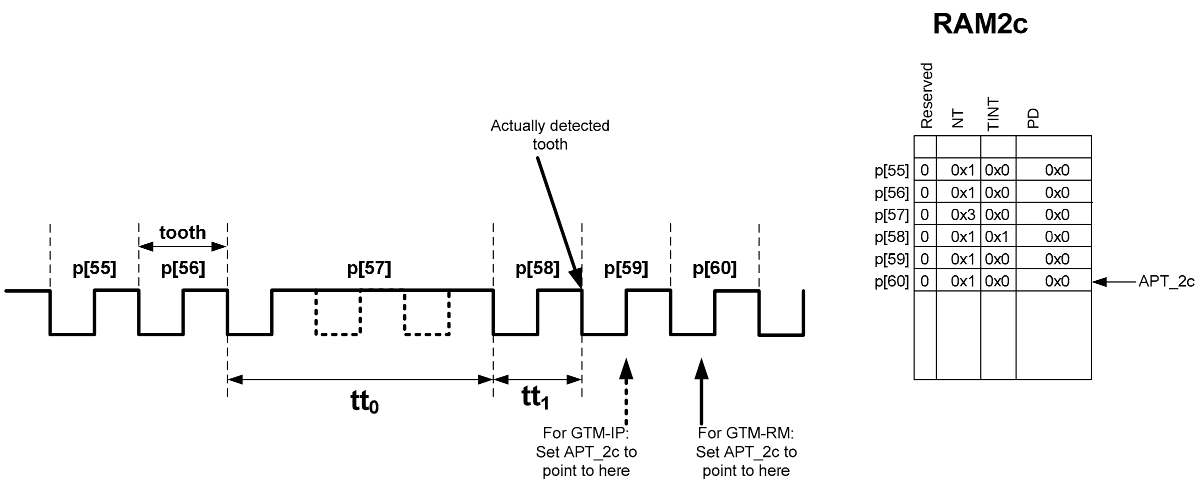
4.4.3 DPLL Profile synchronization

DPLL synchronization means setting the DPLL input *TRIGGER* signal profile pointer APT_2c to the appropriate profile location. When this pointer is set, the DPLL LOCK1 bit in the DPLL_STATUS register is set, and the DPLL operates on the synchronized profile.
The DPLL profile synchronization can only be done, when the input signal has at a specific location a specific pattern. For this application note, the specific pattern are two missing tooth in a normal 60 tooth wheel (only 58 real tooth available).
Therefore, the software has to detect this gap and then set the APT_2c pointer. Since the *TRIGGER* signal characteristic for HALF_SCALE looks the same, the DPLL profile pointer APT_2c is set to the first RAM2c location.
To detect the gap, an interrupt service routine (ISR) is set up on the TIM0 channel 0 NEWVAL interrupt. When this interrupt occurs, the following code fragment is executed:

```
void isr_toothdet(void)
{
  // disable NOTIFY bit
  TIM0_CH0_IRQ_NOTIFY = 0x1;
  // save actual time stamp
  actTS = TIM0_CH0_GPR0;
  // tooth starts with falling edge
  if (!(actTS & 0x01000000)) { // falling edge detected
    // save old time stamp and tooth charateristic
    oldTS   = newTS;
    oldDiff = diffTS;
    // determine new time stamp
    newTS = actTS & 0xFFFFFF;
    // gap detection after second valid edge
    iter++;
    if (iter > 1) {
      diffTS = newTS - oldTS;
      // gap characteristic for two missing tooth
      if (oldDiff >= 2*diffTS) {
        // synchronize DPLL
        if (dut_is_rm)
          DPLL_APT_2C = (57+3)*4;   // use first gap + 3 tooth
        else
          DPLL_APT_2C = (57+2)*4;   // use first gap + 3 tooth
        // enable DPLL sub_inc1c generation
        DPLL_CTRL_1 = 0x80020032;
        // disable TIM0 channel 0 NEWVAL interrupt
        TIM0_CH0_IRQ_EN = 0x0;
      }  // gap detected
    }  // second valid edge
  }  // falling edge
}
```

Code 4.9: Interrupt service routine to detect tooth wheel gap.

Gap detection is done on behalf of the last two tooth periods. If the tooth period before the last tooth period is twice the size, this preceding tooth period was the gap. This is because there are two tooth missing for the tooth wheel.
The APT_2c value has to be set accordingly to a position after this gap. **Due to the fact, that the GTM-RM and the GTM-IP use a different timing for the APT_2c pointer updates, when a new edge occurs, the pointer has to be initialized different for GTM-RM and GTM-IP.**
The GTM-IP updates the RAM region 2c pointer after a new tooth is detected, while the GTM-RM omits this pointer update at a new tooth. Therefore, the APT_2c pointer has to be set to the next tooth that will be detected by the DPLL in case of the GTM-RM. For the GTM-IP the pointer has to be set to the actual tooth. This pointer setting is shown in Figure 4.2.



	Reserved	NT	TINT	PD
p[55]	0	0x1	0x0	0x0
p[56]	0	0x1	0x0	0x0
p[57]	0	0x3	0x0	0x0
p[58]	0	0x1	0x1	0x0
p[59]	0	0x1	0x0	0x0
p[60]	0	0x1	0x0	0x0

Figure 4.2: Gap detection for a tooth wheel with a gap of two tooth.

After the gap is detected, the *sub_inc1* and *sub_inc1c* generation is enabled and the micro ticks are send to the TBU channel 1 time base.

4.4.4 DPLL Lock detection

After the synchronization for the tooth wheel was established, the DPLL lock1 bit setting can be observed, to determine if the DPLL has locked after the second gap of the tooth wheel was detected.
To do this, a user defined interrupt inside of the tooth profile is used. This user defined interrupt is defined after the second gap of the tooth wheel FULL_SCALE. The interrupt definition can be seen in Figure 4.2 at profile position `p[58]`.
An ISR than handles the interrupt and determines if the lock1 bit is set. The ISR code is shown in Code sequence 4.10.

```
void isr_lockdet(void)
{
      unsigned int actTS;

      // check DPLL TRIGGER lock1 is set
      DPLL_IRQ_NOTIFY = 0x00040000;  // reset IRQ NOTIFY bit
      actTS = DPLL_STATUS;  // use actTS as tmp variable
      actTS &= 0x40000000;
      if (!actTS)
        cout << "ERROR: DPLL not locked yet!" << endl;
}
```

Code 4.10: Interrupt service routine to check DPLL locking after second gap.