Freescale Semiconductor
Application Note

Document Number: AN4351
Rev 2, 4/2014

MPC57xxM Generic Timer Module (GTM) Quick Start Guide

An introduction to the GTM as implemented on Freescale MCUs

by: Inga Harris

Contents

1 GTM Overview

The GTM is a data flow driven, large scalable timer with a modular design and a central routing unit. It supports over 200 timed I/O channels and includes application specific modules for hardware support of 4, 5, 6, and 8 cylinder applications for powertrain, transmission, and motor control, including angle clock hardware and motor commutation submodules. The system functionality of the GTM module is similar to that of the Freescale eTPU and eMIOS modules.

The GTM has been designed to minimize the amount of interaction between the GTM and the CPU. This is achieved through specific integrated technology, which results in fewer interrupt requests and therefore reduced CPU loading. Most functions are performed in parallel within the GTM's dedicated hardware units, ensuring simple latency calculations. Although the GTM itself is physically a large module in terms of gate count, the overall system cost is lowered by reducing the software overhead.

Some modules can be used independently as standalone functions that are controlled completely by the MCU's host processor. However, these modules can be combined together to create complex timing functions by using the Multi-Channel Sequencer and the Advanced Routing Unit to control inputs and output to the other GTM submodules.

2 Example 1: Configuring the Microcontroller to use the GTM

2.1 Description

The GTM module runs on two clock domains (one for the logic and one for timing) and in multiple modes that must be configured prior to the module being enabled. To use the GTM's I/O, the pin multiplexing must also be configured prior to the GTM's initialization.

The GTM Integration module is clocked by the device's Peripheral Bridge A domain, on a special GTM slot. This clock is used for communication between the GTM and the CPU via the AEIMux and the GTM's register logic. This clock has a maximum speed of 100 MHz (twice the frequency of the other peripheral bridge slots) on the MPC5746M and MPC5777M devices . This clock domain is highly configurable and can be sourced from PLL0 (non-FMPLL), PLL1 (FMPLL), XOSC, or the IRCOSC. These clock sources can be divided down by any integer value between 1 and 64 to reach the desired frequency.

The GTM system clock (referred to as SYS_CLK in the GTM specification) is sourced from the chip-level peripheral clock (PER_CLK), which is one of the AUX Clock Selector 0 clocks, group 0. This group of clocks is highly configurable and can be sourced from PLL0 (non-FMPLL), XOSC, or IRCOSC. This clock has a maximum frequency of 80 MHz on all of the devices in the family. These clock sources can be divided down by any integer value between 1 and 16 to reach the desired frequency.

2.2 Implementation

To run the GTM104 on the MPC5777M at its maximum speed, it is recommended that PLL1 is configured to run at 600 MHz and PBRIDGEA_CLK set to PLL1 divided by 12, with PLL0 running at 400 MHz and divided by 5 for the source of the PER_CLK as shown in Figure 1 below. This 80 MHz clock is used by the GTM CMU sub module to derive the GTM's local clock signals for its submodules and the GTM's External Clock signals as shown in Figure 2.
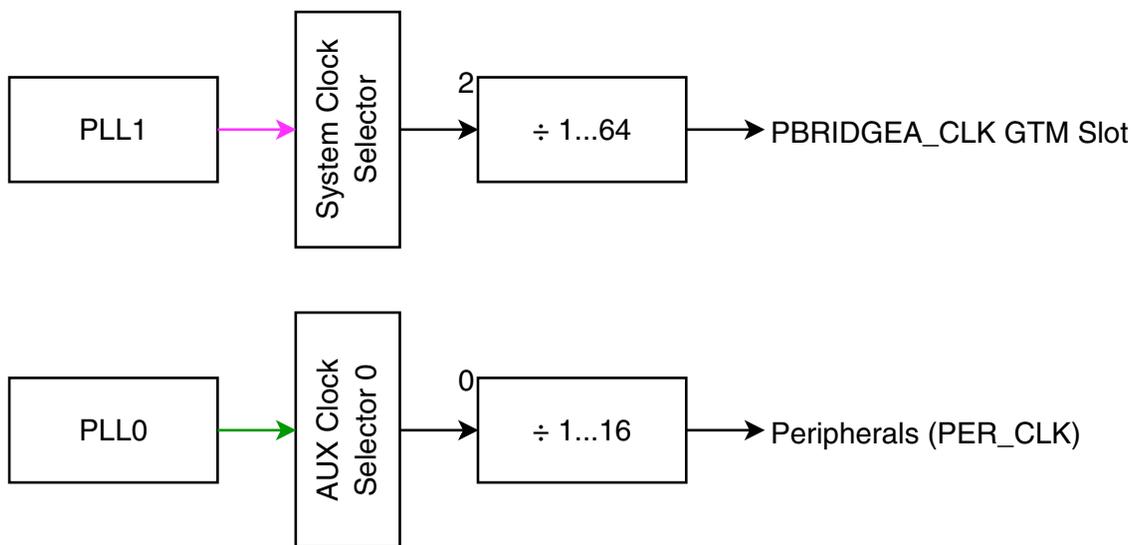

Figure 1. GTM clocking tree on MPC5777M

When the PLL is running at the desired frequency, the clocks can be fed to PBRIDGEA_CLK and PER_CLK by writing to the chip's Clock Generation Module (MC_CGM) configuration registers. Some examples for the preliminary MPC57xx devices set up as shown above are provided in the following table.

Device	PBRIDGEA_CLK (Max)	PER_CLK (Max)	System Clock Selector 0	System Clock Selector Divider	AUX Clock Selector 0	AUX Clock Selector 0 Divider
MPC5777M	100 MHz (600/6)	80 MHz (400/5)	MC_CGM.SC_DC1.R	6	MC_CGM.AC0_DC0.R	5
MPC5746M	100 MHz (600/6)	80 MHz (400/5)	MC_CGM.SC_DC1.R	6	MC_CGM.AC0_DC0.R	5

The GTM's timing clock source is SYS_CLK, which is the PER_CLK signal at the chip level. The device's reference manual describes the maximum frequencies for the PBRIDGEA_CLK and PER_CLK clocks inside their respective *System clock frequency limitations* section in the *Clocking* chapter.
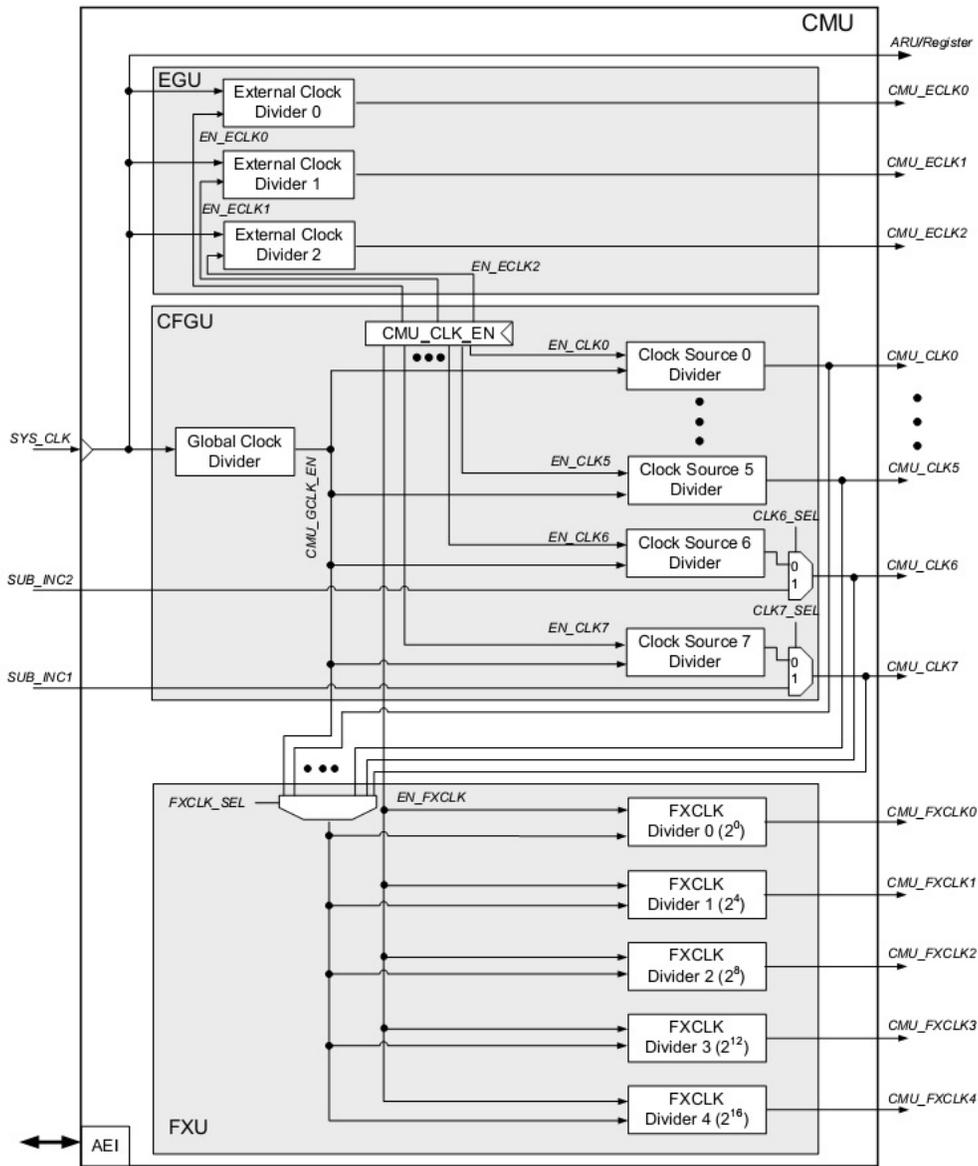
Figure 2. Clock tree inside GTM104

The inputs and outputs of the GTM's TOM, ATOM, and TIM submodules are not routed to the device pads by default. As the pads on the family of devices are highly multiplexed, each individual I/O to be used in the application must be configured in the SIUL2 module before use.

Inside the SIUL2 module are the registers that control the I/O, the Multiplexed Signal Configuration Registers (MSCR_MUX and MSCR_IO). The Source Signal Select (SSS) value selects which source signal is connected to the associated destination pin. The output channels (TOM and ATOM) functionality must be configured including drive strength, output drive circuit, and pullup/down. These output pins are controlled by the MSCR_MUX registers. The input channels (TIM) must have their Input Buffer enabled by setting the IBE bit in the MSCR_IO register. The example in Code is one possibility of the combinations of TIM, TOM and ATOM that will allow all the inputs and outputs to fit on the smallest MPC5746M 176 QFP package without conflict. The code below uses these arrays to set all the inputs and outputs up for use during development.

The I/O Signal Description Table excerpts below show the entries for Port B9 and Port F1. To configure Port B9 as the TOM0_CH0 function, the SIUL2 MSCR_MUX register number 25 should be written with the SSS value of 0000_1000 (8hex) with the other output pin characteristics required.

Port Pin	LVDS[4] Pair Port	SIUL MSCR Number	MSCR SSS	Function	Module	Description	Direction
PB[9]		25	0000_0000	GPIO[25]	SIUL2	General Purpose I/O 25	I/O
			0000_0011	CS0_5	DSPI_5	DSPI_5 Chip Select 0 Output	O
			0000_0100	CS6_1	DSPI_1	DSPI_1 Chip Select 6 Output	O
			0000_0110	CS0_2	DSPI_2	DSPI_2 Chip Select 0 Output	O
			0000_1000	TOM0_0	GTM	GTM Timer Output Module 0 Channel 0	O
			0000_1001	TOM1_0	GTM	GTM Timer Output Module 1 Channel 0	O
			0000_1010	ATOM2_0	GTM	GTM ARU Timer Output Module 2 Channel 0	O
			0000_1011	ATOM3_0	GTM	GTM ARU Timer Output Module 3 Channel 0	O
		512	0000_0100	TIM0_0	GTM	GTM Timer Input Module 0 Channel 0	I
		520	0000_0100	TIM1_0	GTM	GTM Timer Input Module 1 Channel 0	I
		758	0000_0100	MCAN1RX	M_CAN_1	M_CAN_1 Receive Input	I
		848	0000_1011	LIN0RX	LINFlexD_0	LINFlexD_0 Receive Data Input (Serial Boot Receive)	I
		888	0000_0011	SS_2	DSPI_2	DSPI_2 Slave Select Input	I
		897	0000_0001	SS_5	DSPI_5	DSPI_5 Slave Select Input	I

Figure 3. PB[9] MSCR table

To configure Port F1 as the TIM0_CH0 function, the SIUL2 MSCR Input register number 512 should be written with the SSS value of 0x0000_0011 (3hex) and the ports associated MSCR register number 81 Input Buffer Enabled (IBE).

Port Pin	LVDS[4] Pair Port	SIUL MSCR Number	MSCR SSS	Function	Module	Description	Direction
PF[1]		81	0000_0000	GPIO[81]	SIUL2	General Purpose I/O 81	I/O
			0000_0011	CS4_2	DSPI_2	DSPI_2 Chip Select 4 Output	O
			0000_0101	SDOUT1_PSI5_0	PSI5_0	PSI5_0 Sensor Data 1 Output	O
			0000_1000	TOM0_0	GTM	GTM Timer Output Module 0 Channel 0	O
			0000_1001	TOM1_0	GTM	GTM Timer Output Module 1 Channel 0	O
			0000_1010	ATOM2_0	GTM	GTM ARU Timer Output Module 2 Channel 0	O
			0000_1011	ATOM3_0	GTM	GTM ARU Timer Output Module 3 Channel 0	O
		512	0000_0011	TIM0_0	GTM	GTM Timer Input Module 0 Channel 0	I
		520	0000_0011	TIM1_0	GTM	GTM Timer Input Module 1 Channel 0	I
		578	0000_0010	EXT_DATA1	SDADC_1	Sigma-Delta ADC 1 Ext Modulator Data Input	I
		797	0000_0010	SENT4_1	SENT_1	SENT Receiver 1 Channel 4 Input	I

Figure 4. PF[1] MSCR table

NOTE

The I/O signal tables are included in the chip's reference manual as an attached Excel® spreadsheet.

2.3 Code

The arrays below show one possible way to have all of the GTM103 inputs and outputs on the MPC5746M routed to the device's I/O on the 292 MAPBGA package.

NOTE

Some ports are unavailable on the chip's smaller package options. Refer to the data sheet for details on which ports are available on specific packages.

Example 1: Configuring the Microcontroller to use the GTM

```
/* SIUL2 set up arrays **********/

int TIM_MSCR_SSS[32] = {/*TIM0*/3, 2, 5, 2, 2, 2, 2, 2,
                        /*TIM1*/8, 3, 7, 3, 4, 4, 7, 3,
                        /*TIM2*/7, 7, 7, 9, 8, 9, 7, 8,
                        /*TIM3*/7, 6, 6, 7, 7, 7, 8, 8};
int TIM_MSCR[32] = {/*TIM0*/81, 96, 118, 137, 141, 140, 139, 138,
                    /*TIM1*/122, 80, 113, 58, 145, 146, 126, 116,
                    /*TIM2*/197, 185, 185, 185, 185, 185, 195, 192,
                    /*TIM3*/200, 199, 157, 156, 159, 158, 190, 191};

int TOM0_MSCR_SSS[16] = {8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8};
int TOM1_MSCR_SSS[16] = {9, 9, 9, 8, 8, 8, 8, 8, 9, 9, 8, 8, 8, 8, 8, 8};
int TOM2_MSCR_SSS[16] = {9, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9};

int TOM0_MSCR[16] = {25, 26, 24, 10, 8, 9, 50, 49, 34, 32, 41, 40, 39, 38, 37, 36};
int TOM1_MSCR[16] = {35, 33, 57, 2, 1, 12, 13, 0, 62, 63, 190, 123, 3, 56, 71, 52};
int TOM2_MSCR[16] = {93, 196, 47, 91, 45, 44, 43, 42, 53, 72, 73, 82, 127, 70, 86, 87};

int ATOM0_MSCR_SSS[8] = {0xA, 0xA, 0xA, 0xA, 0xA, 0xA, 0xA, 0xA};
int ATOM1_MSCR_SSS[8] = {0xA, 0xA, 0xB, 0xB, 0xB, 0xB, 0xB, 0xB};
int ATOM2_MSCR_SSS[8] = {0xA, 0xA, 0xA, 0xA, 0xA, 0xA, 8, 8};
int ATOM3_MSCR_SSS[8] = {0xB, 0xB, 0xB, 0xB, 0xB, 0xB, 0xB, 0xB};
int ATOM4_MSCR_SSS[8] = {0xB, 0xB, 0xB, 0xB, 0xB, 0xB, 0xB, 0xB};

int ATOM0_MSCR[8] = {90, 55, 54, 14, 11, 15, 61, 67};
int ATOM1_MSCR[8] = {111, 84, 64, 65, 66, 60, 88, 117};
int ATOM2_MCSR[8] = {48, 147, 121, 120, 119, 143, 152, 150};
int ATOM3_MCSR[8] = {68, 76, 136, 46, 27, 51, 109, 110};
int ATOM4_MCSR[8] = {115, 75, 74, 114, 92, 124, 94, 69};
```

Refer to SIUL2 for GTM Inputs and Outputs for further details on the SIUL2 module and the mapping of GTM pins to the device port pins/pads on the GTM103 and GTM104 variants on the MPC57xx family.

Below is the function SetUpSIUIO which uses the above data to configure all the GTM I/O on the GTM103-based MPC5746M device.

```
void SetUpSIUIO()
{
    unsigned char i;

    //Sets up all TIM channels
    for (i=0; i<32; i++){
        SIUL2.MSCR_MUX[TIM_MSCR[i]].B.SSS = TIM_MSCR_SSS[i];
        SIUL2.MSCR_IO[TIM_MSCR[i]].B.IBE = 1;
    }

    //Sets up all TOM channels
    for (i=0; i<16; i++){
        SIUL2.MSCR_IO[TOM0_MSCR[i]].R = (0x02080000|TOM0_MSCR_SSS[i]);
        SIUL2.MSCR_IO[TOM1_MSCR[i]].R = (0x02080000|TOM1_MSCR_SSS[i]);
        SIUL2.MSCR_IO[TOM2_MSCR[i]].R = (0x02080000|TOM2_MSCR_SSS[i]);
    }

    //Sets up all ATOM channels
    for (i=0; i<8; i++){
        SIUL2.MSCR_IO[ATOM0_MSCR[i]].R = (0x02080000|ATOM0_MSCR_SSS[i]);
        SIUL2.MSCR_IO[ATOM1_MSCR[i]].R = (0x02080000|ATOM1_MSCR_SSS[i]);
        SIUL2.MSCR_IO[ATOM2_MSCR[i]].R = (0x02080000|ATOM2_MSCR_SSS[i]);
        SIUL2.MSCR_IO[ATOM3_MSCR[i]].R = (0x02080000|ATOM3_MSCR_SSS[i]);
        SIUL2.MSCR_IO[ATOM4_MSCR[i]].R = (0x02080000|ATOM4_MSCR_SSS[i]);
    }
}
```

These could be configured one step at a time instead of in one looped function. Below is an example of how one such input configuration would be done:

- SIUL2.MSCR_MUX[81].B.SSS = 3; sets up TIM0_CH0 on PF1.
- SIUL2.MSCR_IO[81].B.IBE = 1; sets TIM0_CH0 (on PF1) as an input (81+512 = 593).

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

Below is an example of how one such output configuration would be done:

- SIUL2.MSCR_IO[90].R = 0x0208000A; sets up ATOM0_CH0 as a digital output pin on PF10 with weak drive, push-pull, safe mode and the weak pull disabled.

NOTE
There are not enough I/Os on the MPC5777M to have all of the GTM104 inputs and outputs available on external pins.

3 Example 2: Enabling the GTM

3.1 Description

The Generic Timer Module (GTM) is a complex timing subsystem intended to be used in automotive powertrain applications. To enable the the GTM, several operations must be performed. Both the clocks to the GTM Integration Module and the GTM module itself are gated off out of reset.

The GTM can operate in all SoC modes if it is configured to do so. The GTM Stop mode referenced in the GTM specification is any SoC mode in which the GTM clock sources are not enabled.

3.2 Implementation

The GTM module has two reset signals, both of which must be released before the GTM can be enabled and configured. The first reset signal is from the chip's slave bus and the second is from the peripheral clock domain. The release of the last reset signal resets the the GTM logic and the GTM registers.

When the GTM reset signals are both released, the GTM wrapper is in asynchronous mode and write responses are not masked but the module is still disabled with MDIS = 1. When the GTM's two reset signals are released the GTM's MCS and DPLL RAM memories are automatically initialized by writing to each location. This is required to ensure that the ECC parity bits for each RAM address are true.

NOTE
The GTM's FIFO memories are not initialized because the FIFOs are not read directly. They are written before any read access occurs from either the GTM or by the device's cores.

The first step to enabling the GTM is to enable the Slow Crossbar and peripheral clocks to the Mode Entry (ME) module's Run Peripheral Configuration 0 register (ME_RUN_PC0).

The clocks to the GTM do not become active until the run mode is updated in the ME module. Normally, this would be done after initialization of all of the peripheral clocks and core operating modes is completed.

Finally, the GTM itself can be enabled by clearing the Module Disable (MDIS) in the GTM Module Configuration Register (GTMMCR) located at the Base Addr + 0xC0. The default value of this register is 0x4000_4000 (the module is disabled and the GTM is in Stop mode). The GTM Stop Mode Status (STPS) bit is cleared when the GTM is enabled. This MDIS bit must be cleared to enable writes to the GTM registers for configuration before operation. The other registers in this integration register set are the Interrupt Clear and AEI Control registers which are unlikely to be written during initialization.

Verification that the GTM is enabled can be done by reading the GTM Version Control register (GTM_REV). This register is not readable when the GTM is disabled and contains the GTM module type (3-digit Device Encoding), the GTM major and minor revision number, the development year, and IP delivery number. See GTM module definition and revision information.
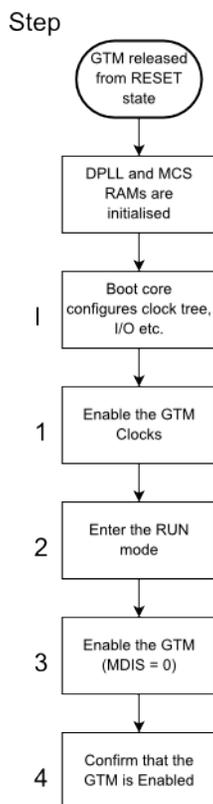
Step

Figure 5. Enabling the GTM

Table 2.   Steps to enable the GTM

Step	Operation	Description	Pseudo Code
I	Initialized State	See Example 1	—
1	Enable the GTM clock	Enable the clock in the ME Peripheral Configuration 0 Register (ME_RUN_PC0). A value of 0xF8 enables the following modes: RUN3, RUN2, RUN1, RUN0, and DRUN.	Write ME_RUN_PC0 = 0x0000_00F8
2	Change the run mode to enable clocks	Write target mode (DRUN) in the ME Mode Control register (ME_MCTL) with the first key	Write ME_MCTL = 0x3000_5AF0
		Write target mode (DRUN) in the ME Mode Control register (ME_MCTL) the inverse key	Write ME_MCTL = 0x3000_A50F
3	Enable the GTM	Enable the GTM by clearing the MDIS bit in the MCR.	Write GTM_MCR = 0x0000_0000
4	Confirm GTM enabled	Read GTM Version Control register[1]	Read GTM_REV

1.   The GTM Version control register cannot be read if the GTM is disabled.

3.3   Code

This function is an example based on the MPC5777M device. It only configures the clocks and modes required for the GTM module to run, assuming the PLLs and all other sources have been configured independantly.

```
void GTM_CLK_INIT(void){

/* Assumes PLL0 at 400 MHz from 40 MHz XOSC and PLL1 @ 600 MHz from 40 MHz XOSC with
progressive clock switching enabled */
/* All cores enabled and all MC_RGM faults cleared */

/** Step 1 **/
   /* Setting RUN Configuration Register */
   MC_ME.RUN_PC[0].R=0x000000F8;           /* GTM ON in RUNx and DRUN modes */

/** Step 2 **/
   /* Mode change Re-enter the DRUN mode, to start cores, clock tree & PLL1 */
   MC_ME.MCTL.R = 0x30005AF0;                              /* Mode & Key */
   MC_ME.MCTL.R = 0x3000A50F;                         /* Mode & Key inverted */

   while(MC_ME.GS.B.S_MTRANS == 1);           /* Wait for mode entry complete */
   while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);       /* Check DRUN mode entered */

/** Step 3 **/
   GTMINT.GTM_MCR.R = 0x000000000; //Enable GTM Module, MDIS = 0

/* Step 4 */
   if (GTM_REV != 0x104155A1){
   /* See Appendix A for the value for each GTM Derivatives Revision Number */
      /* report an error */
   }
}
```

4   Example 3: Initializing the GTM

4.1   Description

Once the GTM chip-top integration configuration has been initialized, the other top-level GTM Configuration Registers in the GTM Integration module should be initialized for use by the application. These include the GTM Control, Bridge Mode, and interrupt (IRQ) registers. The GTM specification refers to two bridge modes.

NOTE

The integration of the GTM module in MPC57xx devices is the asynchronous bridge mode (BRG_MODE = 1). When the GTM is used in asynchronous mode, the two clock inputs to the GTM subsystem must have identical frequency and phase, which cannot be guaranteed within the MPC57xx integration scheme.

In general, the next stage of initialization for the GTM is the Clock Management Unit (CMU) and the Time Base Unit (TBU). The CMU submodule controls the GTM clocking. There is a global clock divider ratio applied to all the internal GTM clocks, which is not applied to the CMU's External Clock Generation Unit. The Global Clock Divider has a mechanism to protect it from being misconfigured (that is, to prevent the numerator or denominator from being 0 or the ratio from being less than 1). The clock dividers, clock select, and clock enable bits control the CMU submodules' outputs. The TBU can then be configured to generate the different time bases that can be used by the other GTM submodules.

Interrupts and DMA must also be configured for use with the GTM. Outside of the GTM modules, the interrupts and DMA configuration is the same as any other module. Refer to AN4352, *Initializing the MPC5746M*, for further information on how to configure these events to meet your application's demands.

Inside the GTM the interrupt and DMA events sent to the chip's interrupt and DMA controllers are all handled as interrupts. Each individual interrupt event that will be used in the end application must be enabled for use inside the respective submodule.

NOTE
The GTM IP has the ability to handle four interrupt modes: Level, Pulse, Pulse-notify, and Single-pulse modes. However, the integration of the GTM IP inside of the MPC57xx devices uses Level mode (b00, reset state) only. Writing any value other that b00 to these bit fields results in an AEI_STATUS of Illegal Address Access (b10) and generates the GTM AEI shared interrupt, vector number 706, GTM_ICM_IRQG_0(AEI_IRQ).

4.2   Implementation

The GTMINT module will exit reset in a default state which does not need any modifications to work with the GTM-IP in the MPC57xx chips.

There are four steps to enable the GTM internal clocks using the CMU submodule.

1. Set the global clock divider ratio.
2. Select a fixed-frequency clock divider.
3. Select the source and divider for each CLKn signal.
4. Enable the clocks.

Table 3.   Initializing the internal GTM clocks

Step	Operation	Description	Psuedo Code
1	Define the global clock as a fraction of the PER_CLK	Write the numerator and denominator values of the divider	CMU_GCLK_NUM = 0xN; CMU_GCLK_DEN = 0xN;
2	Select the fixed frequency clock divider source	Chose from any other internal CMU clock	CMU_FXCLK_CTRL = 0xN;
3	Define the CMU_CLKn source and divider	Define the integer divider value for the clock and the source where available (CLK6 and CLK7)	CMU_CLK_0_CTRL = 0xN; CMU_CLK_1_CTRL = 0xN; CMU_CLK_2_CTRL = 0xN; CMU_CLK_3_CTRL = 0xN; CMU_CLK_4_CTRL = 0xN; CMU_CLK_5_CTRL = 0xN; CMU_CLK_6_CTRL = 0xN; CMU_CLK_7_CTRL = 0xN;
4	Enable the clocks	Enable all the clock prescalers and FXCLK	CMU_CLK_EN = 0x00AAAAAA;

4.3   Code

Below is a C function that shows an example of the CMU clock configurations.

```
void init_GTM_clocks (void){

  GTM_CMU.GCLK_NUM.R = 0xFFFFFF;
  GTM_CMU.GCLK_DEN.R = 0xFFFFFF; // define CMU_GCLK_EN as PER_CLK / 1

  GTM_CMU.FXCLK_CTRL.R = 0x1; //FXCLK sourced from CMU_CLK0

  GTM_CMU.CLK_CTRL[0].R = 0x0;    // define CMU_GCLK_EN/1 clock
```

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

```
GTM_CMU.CLK_CTRL[1].R = 0x1;     // define CMU_GCLK_EN/2 clock
GTM_CMU.CLK_CTRL[2].R = 0x3;     // define CMU_GCLK_EN/4 clock
GTM_CMU.CLK_CTRL[3].R = 0x7;     // define CMU_GCLK_EN/8 clock
GTM_CMU.CLK_CTRL[4].R = 0x9;     // define CMU_GCLK_EN/10 clock
GTM_CMU.CLK_CTRL[5].R = 0x4F;    // define CMU_GCLK_EN/80 clock
GTM_CMU.CLK_6_CTRL.R = 0x4F;     // define CMU_GCLK_EN/80 clock
GTM_CMU.CLK_6_CTRL.R = 0x4F;     // define CMU_GCLK_EN/80 clock


    // enable all the clock prescalers and FXCLK
    GTM_CMU.CLK_EN.R = 0x00AAAAAA;

}
```

NOTE

The Time Base Unit channels can also be enabled, if required by the GTM submodules, by writing to the TBU_CHEN register (for example, `TBU_CHEN = 0x0000002A;`).

5 Example 4: Simple PWM

When first starting with a module as big and complex as the GTM it is a good idea to start with a simple function that enables you to check the clock speed and toggle some GTM pins.

5.1 Description

The Timer Output Module (TOM) is the simplest of the submodules to turn on since it is not connected through the ARU. This example uses TOM0, channels 0, 1, 2, and 3.

The TOM0 Channel registers Compare Match 0 (CM0) and Compare Match 1 (CM1) control the PWM frequency and duty cycle. TOM Channel register Counter (CN0) is the counter. The mode and functionality of the TOM is set from the Channel Control Register (TOMi_CHn_CTRL). The TOM channels and outputs are enabled with the ENDIS and OUTEN control registers.

5.2 Implementation

The following example sets up TOM0 channels 0–3 in continuous PWM mode running from the CMU_FXCLK[0] with high (SL = 1) output level. It does not use any interrupts or time bases and all other setting are default.

There are three main steps that must be taken to generate these synchronized PWM outputs on the TOM module.

1. Set up channel period and duty as this channel triggers the others.
2. Configure the channel mode.
3. Enable the TOM outputs.

Table 4. Simple PWM outputs

Step	Operation	Description	Psuedocode
I	Initial state	Enable the GTM and clocks by completing examples 1, 2, and 3.	—
1	Set up CHn period	Enter the number of counts that the counter will compare with for the period and duty.	Write (A)TOMn_CHn_CM1 = duty count and (A)TOMn_CHn_CM0 = period count
3	Configure CHn	Set channel up for cont mode, SL=1, RST_CCU0=1	(A)TOMn_CHn_CTRL = mode required

Table continues on the next page...

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

Table 4. Simple PWM outputs (continued)

Step	Operation	Description	Psuedocode
5	Enable outputs	Enable the outputs for the channels in TGC0/1	(A)TOMn_TGCn_OUTEN_CTRL = 0x000000AA;

The resultant waveform is PWM outputs of 1.22 kHz from a GTM clock of 80MHz as shown in Figure 6.
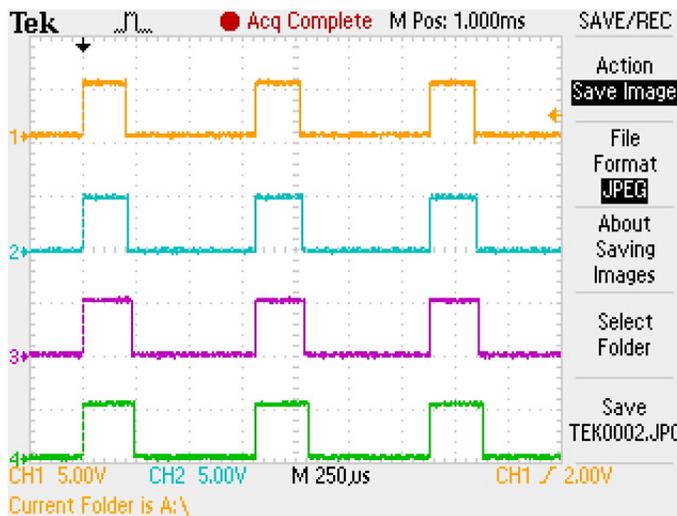
Figure 6. PWM waveform output

5.3 Code

```
/* Step 1 */
GTM_TOM_0.CH0_CM1.R = 0x00004000; // 16k counts @ 80MHz
GTM_TOM_0.CH0_CM0.R = 0x0000FFFF; // 64k counts @ 80MHz
GTM_TOM_0.CH1_CM1.R = 0x00004400;
GTM_TOM_0.CH1_CM0.R = 0x0000FFFF; // 64k counts @ 80MHz
GTM_TOM_0.CH2_CM1.R = 0x00004800;
GTM_TOM_0.CH2_CM0.R = 0x0000FFFF; // 64k counts @ 80MHz
GTM_TOM_0.CH3_CM1.R = 0x00004C00;
GTM_TOM_0.CH3_CM0.R = 0x0000FFFF; // 64k counts @ 80MHz

/* Step 2 */
GTM_TOM_0.CH0_CTRL.R  = 0x00100800;      // Ch0: cont mode, SL=1, RST_CCU0=1
GTM_TOM_0.CH1_CTRL.R  = 0x00100800;      // Ch1: cont mode, SL=1, RST_CCU0=1
GTM_TOM_0.CH2_CTRL.R  = 0x00100800;      // Ch2: cont mode, SL=1, RST_CCU0=1
GTM_TOM_0.CH3_CTRL.R  = 0x00100800;      // Ch3: cont mode, SL=1, RST_CCU0=1

//Enable TOM global Channel control units to output
/* Step 3 */
GTM_TOM_0.TGC0_OUTEN_CTRL.R = 0x000000AA;
GTM_TOM_0.TGC0_ENDIS_CTRL.R = 0x000000AA;
GTM_TOM_0.TGC0_OUTEN_STAT.R = 0x000000AA;
GTM_TOM_0.TGC0_ENDIS_STAT.R = 0x000000AA;
```

6 Example 5: Synchronizing the TOM and the ATOM Submodules

6.1 Description

The GTM TOM and ATOM submodules have a synchronous update mechanism. The compare registers, CM0 and CM1, have shadow register equivalents, SR0 and SR1. Synchronous updates of the PWM duty and/or period can be achieved by loading the shadow registers with the next required value of CM0 and CM1 while the update mechanism is enabled. The shadow register values are transferred to the compare registers at the start of the next period.

The TOM and ATOM Clock Source register can also be synchronously updated by writing a new value to the CLK_SRC register.

The TGC0/1 Global Control register UPEN_CTRLn bit field controls the update mechanism. This bit field is modified by writing 0b01 to disable updates or 0b10 to enable updates. Writing 0b00 or 0b11 has no effect. This bit field can be read to find the status of the update mechanism: 0b00 means the channel is disabled and 0b11 means the mechanism is enabled.

6.2 Implementation

The following example sets up TOM0 channels 0–3 in continuous PWM mode running from the CMU_FXCLK[0] with high (SL = 1) output level with channel 0 as the triggering master. It does not use any interrupts or time bases and all other setting are default.

There are eight main steps that must be taken to generate these synchronised PWM outputs on the TOM module.
1. Configure channel 0 mode.
2. Configure channel 1, 2, and 3 modes.
3. Set up channel 0 period and duty, as this channel triggers the others.
4. Set up channel 1, 2, and 3 period and duty.
5. Enable the TOM outputs.
6. Force an update of the TOM registers.
7. Wait some time to observe the waveform on an oscilloscope.
8. Alter channel 1 period and duty synchronously.

Table 5. Synchronized PWM outputs

Step	Operation	Description	Psuedocode
I	Initial state	Enable the GTM and clocks by completing examples 1, 2 and 3	—
1	Configure CH0 mode	Set channel up for cont mode, SL=1, Trigout = 1	TOM0_CH0_CTRL = 0x01001800;
2	Configure CHn mode	Set channels up for cont mode, SL=1, RST_CCU0 = 1	TOM0_CHn_CTRL = 0x00101000;
3	Set up CH0	Enter the number of counts that the counter will compare with for the period and duty, including the shadow register values	TOM0_CH0_CM1 = TOM0_CH0_SR1 = 0x00000032; TOM0_CH0_CM0 = TOM0_CH0_SR0 = 0x00000064; TOM0_CH0_CN0 = 0x00000000;

Table continues on the next page...

Table 5. Synchronized PWM outputs (continued)

Step	Operation	Description	Psuedocode
4	Set up CHn	Enter the period and duty for all other channels	TOM0_CHn_CM1 = TOM0_CHn_SR1 = duty; TOM0_CHn_CM0 = TOM0_CHn_SR0 = period;
5	Enable outputs	Set OUTEN for the channels used	TOM0_TGC0_OUTEN_CTRL = 0x000000AA; TOM0_TGC0_ENDIS_CTRL = 0x000000AA; TOM0_TGC0_OUTEN_STAT = 0x000000AA; TOM0_TGC0_ENDIS_STAT = 0x000000AA;
6	Force update	Enable update of registers CM0, CM1, and CLK_SRC from SR0, SR1, and CLK_SRC_SR and generate a software trigger	TOM0_TGC0_GLB_CTRL = 0x00AA0001;
7	Wait	Leave some time so you can observe the PWM signals	Delay function
8	Synchronously update only CH1	Alter the PWM characteristics in the channel shadow registers	TOM0_CH1_SR1 = duty; TOM0_CH1_SR0 = period;

The example below of a synchronous update of the period and duty is shown on CH1 of TOM0. On the ATOM submodule, this update would come from the ARU word rather than a CPU access. The TOM is set up similarly to the set up shown in Example 4: Simple PWM. The duty cycle and period are updated for CH1 by writing to the SR0/1 registers.
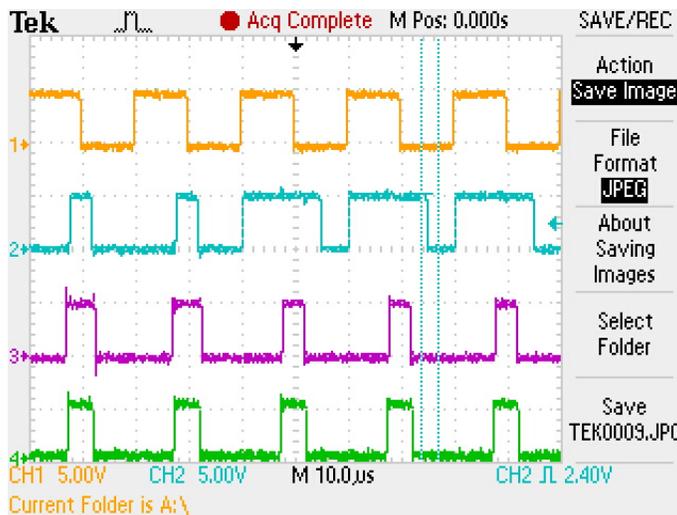
**Figure 7. Synchronous Update of PWM Period and Duty**

To enable the oscilloscope to capture the event where the PWM was changed as shown in Figure 7, the trigger was set to CH2 on a pulse width >10μs.

6.3 Code

```
void Synchronous_PWM_Update(void)
{
  //*****************************************************************
  // Channel 0 is master that defines period of PWM
```

```
    // Channels 1 and 2 are synchronised to channel 0
    //*******************************************************************
    GTM_TOM_0.CH0_CTRL.R  = 0x01001800;      // Ch0: cont mode, SL=1, Trigout=1
    GTM_TOM_0.CH1_CTRL.R  = 0x00101000;      // Ch1: cont mode, SL=1, RST_CCU0=1
    GTM_TOM_0.CH2_CTRL.R  = 0x00101000;      // Ch2: cont mode, SL=1, RST_CCU0=1


    GTM_TOM_0.CH0_CM1.R = 0x00000032;
    GTM_TOM_0.CH0_SR1.R = 0x00000032;
    GTM_TOM_0.CH0_CM0.R = 0x00000064;
    GTM_TOM_0.CH0_SR0.R = 0x00000064;
    GTM_TOM_0.CH0_CN0.R = 0x00000000;

    GTM_TOM_0.CH1_CM1.R = 0x00000028;
    GTM_TOM_0.CH1_SR1.R = 0x00000028;
    GTM_TOM_0.CH1_CM0.R = 0x0000003C;
    GTM_TOM_0.CH1_SR0.R = 0x0000003C;


    GTM_TOM_0.CH2_CM1.R = 0x00000026;
    GTM_TOM_0.CH2_SR1.R = 0x00000026;
    GTM_TOM_0.CH2_CM0.R = 0x0000003E;
    GTM_TOM_0.CH2_SR0.R = 0x0000003E;


    //Enable TOM Global Channel control units to output
    GTM_TOM_0.TGC0_OUTEN_CTRL.R = 0x0000002A;
    GTM_TOM_0.TGC0_ENDIS_STAT.R = 0x0000002A;


    Delay(100);
    GTM_TOM_0.TGC0_GLB_CTRL.R = 0xAAAA0001;


    // Synchronous update of PWM characteristic on channel 1
    GTM_TOM_0.CH1_SR1.R = 0x00000002;
    GTM_TOM_0.CH1_SR0.R = 0x0000004C;
}
```

7 Example 6: Pulse Period Accumulate

The Pulse Period Accumulate (PPA) function provides Pulse Width Modulation and Period Measurement accumulated over one or more pulses or periods. The maximum value of the accumulated pulse or period is limited to 24 bits by the TIM submodule. Two different types of pulse width measurement are supported: either the high pulse width or the low pulse width is measured. Two different types of period measurement are supported: the period measurement is updated either on the rising edge of the waveform or the falling edge.

7.1 Description

This example uses the ATOM, ARU, and PSM submodules to generate a complex PWM waveform with variable period/duty on ATOM0_CH0. This can be connected on the board to TIM0_CH0, which captures the data in TPWM mode and routes it to the FIFO for a core to read and process. Alternatively, the captured data could be sent to the MCS for further processing—for example, summing of a number of low time values or dividing the pulse and period values to calculate a duty value.

The PPA measurement is expressed in counts of the selected GTM timebase. The example uses TBU_TS0 as the timebase for the measurement. The number of TBU counts that have been accumulated multiplied by the period of one system clock count will give the total measurement expressed in seconds.

7.2   Implementation

The PPA function is realized using the TIM PWM submodule PWM Measurement Mode (TPWM). In TPWM Measurement Mode, the TIM channel measures the duty cycle and period of an incoming PWM signal and stores the pulse width in the GPR0 register and the period in the GPR1 register which can be read by any destination submodule such as the MCS or the PSM.

In this example, the data is read by the PSM. The selection of the measurement of the high or low pulse or the start of the measurement of the period from the rising or falling edge is controlled by the Signal Level Control bit (DSL) in the TIM channel control register (TIMx_CHn_CTRL). The selection of the period or duty is controlled by the PSM's F2A submodule by the transfer mode (TMODE) in the F2A Channel Stream Configuration register F2Ax_CHn_STR_CFG.

Table 6.   Setting for each PPA mode

Mode	Result storage	TIM CH DSL setting	F2A CH TMODE setting
High Pulse	GPR0	b0	b00
Low Pulse	GPR0	b1	b00
Rising Edge	GPR1	b0	b01
Falling Edge	GPR1	b1	b01
High Pulse, Rising Edge	GPR0 and GPR1	b0	b10
Low Pulse, Falling Edge	GPR0 and GPR1	b1	b10

The following steps are required to perform the PPA function as described:

1. Configure the TIM channel in TPWM mode with the counter clocked from the TBU_TS0 and the GPR0/1 registers set to use CNT as the source and the ARU enabled.
2. Configure the FIFO start and end address along with the mode and RAM lock required, flush the FIFO is necessary. Enable the FIFO interrupt/DMA.
3. Configure the F2A to read from the TIM channel and which direction and words to read. Then enable the stream.
4. Generate the signal for the TIM channel to capture. In this example, the ATOM is used; it pulls its next value from another FIFO stream in ring buffer mode.

Table 7.   Steps to Perform PPA on the GTM

Step	Operation	Description
I	Initial state	Repeat examples 1, 2, and 3
1	Configure TIM Channel	TPWM mode with the counter clocked from the TBU_TS0 and the GPR0/1 registers set to use CNT as the source and the ARU enabled.
2	Configure FIFO	Start and end address defined along with the FIFO mode and RAM lock enable/disable. Flush the FIFO. Enable the FIFO interrupts/DMAs.
3	Configure F2A	Read from the TIM channel and which direction and words to read. Then enable the stream.
4	Generate input stimuli	Configure a TOM/ATOM or signal generator to input a PWM for the TIM.

7.3 Code

```
#define FIFO_SIZE 32
#define TIM0_WRADDR0 0x001
#define F2A0_WRADDR0 0x051

int FIFO_values[33] = {0x00010000, 0x00000F00, 0x0000F000, 0x00000F00, 0x0000E000,
0x00000F00, 0x0000D000, 0x00000F00, 0x0000C000, 0x00000F00, 0x0000B000, 0x00000F00,
0x0000A000, 0x00000F00, 0x00009000, 0x00000F00, 0x00008000, 0x00000F00, 0x00007000,
0x00000F00, 0x00006000, 0x00000F00, 0x00005000, 0x00000F00, 0x00004000, 0x00000F00,
0x00003000, 0x00000F00, 0x00002000, 0x00000F00, 0x00001000, 0x00000F00, 0x00010000};

int TIM_results[32];

void PPA(void){

GTM_TIM_0.CH0_CTRL.R = 0x00000F21;
/* --    CLK_SEL     = 0b00 => CMU_CLK0 selected
--    FLT_CTR_FE  = 0 => Up/Down Counter
--    FLT_MODE_FE = 0 => Immediate edge propagation mode
--    FLT_CTR_RE  = 0 => Up/Down Counter
--    FLT_MODE_RE = 0 => Immediate edge propagation mode
--    FLT_CNT_FRQ = 0b00 => FLT_CNT counts with CMU_CLK0
--    FLT_EN      = 0 => Filter disabled
--    ISL         = 0 => use DSL bit for selecting active signal level
--    DSL         = 0 => Measurement starts with rising edge
--    CNTS_SEL    = 0 => use TBU_TS0 as input to CNT
--    GPR1_SEL    = 0b11 => 00 = use CNT as input
--    GPR0_SEL    = 0b11 => 00 = use CNT as input
--    CICTRL      = 0=> use signal TIM_IN(x) as input for channel x
--    ARU_EN      = 1 => enabled
--    OSM         = 0 (continous mode) ;
--    TIM_MODE    = 0b000 => PWM Measurement Mode;
--    TIM_EN      = 1 => enabled */
while(GTM_TIM_0.CH0_CTRL.R!=0x00000F21);

//config FIFO output channel
GTM_FIFO_0.CHANNEL[0].END_ADDR.R = 0x0000001F; // Start address = 0, size = 32
GTM_FIFO_0.CHANNEL[0].CTRL.R = 0x0000000D; // RAM write unlocked, FIFO flushed and Ring
Buffer Mode
GTM_FIFO_0.CHANNEL[0].IRQ_EN.R = 0x00000000; // Disable all the FIFO0 Ch0 Interrupts
/************************/
// LOAD FIFO MEMORY
// preload input FIFO with ATOM duty/period values
/************************/
for (i = 0; i < FIFO_SIZE; i++ ) {
    GTM_AFD_0.CH[0].BUF_ACC.R = FIFO_values[i];
}
GTM_FIFO_0.CHANNEL[0].CTRL.R = 0x00000001; // RAM write locked
GTM_F2A_0.CH_STR_CFG[0].R = 0x00060000; // Transport both words from FIFO to ARU

//config FIFO input channel
GTM_FIFO_0.CHANNEL[1].END_ADDR.R = 0x0000021F; // Start address = 0x200, size = 32
GTM_FIFO_0.CHANNEL[1].CTRL.R = 0x00000004; // RAM write locked, FIFO flushed and Normal FIFO
Mode
GTM_FIFO_0.CHANNEL[1].IRQ_EN.R = 0x00000002; // Enable the FIFO0 Ch1 Full Interrupt
GTM_F2A_0.CH_ARU_RD_FIFO[1].R = TIM0_WRADDR0;
GTM_F2A_0.CH_STR_CFG[1].R = 0x00020000; // Transport both words from ARU to FIFO

GTM_F2A_0.ENABLE.R = 0x0000000A; // Enable streams 0 and 1;

//config ATOM
GTM_ATOM_0.CH0_RDADDR.R  = ((F2A0_WRADDR0<<16)+(F2A0_WRADDR0));
GTM_ATOM_0.CH0_CTRL.R = 0x0800000A; // SOMP, TB1_SEL=0, ARU_EN=1, SL=0, WR_REQ=0
GTM_ATOM_0.AGC_OUTEN_CTRL.R = 0x00000002;
GTM_ATOM_0.AGC_ENDIS_CTRL.R = 0x00000002;
while (GTM_ATOM_0.AGC_ENDIS_STAT.R != 0x00000000);
```

```
GTM_ATOM_0.AGC_GLB_CTRL.R = 0x00020001; // setting of bit 0
}
```

NOTE

Ensure that the ATOM and TIM port pins are configured for a high drive strength to get accurate results.

Below is an example of a ISR that could test the values received against those that were transmitted by the ATOM.

```
void IRQ_GTM_PSM0_CH1 (void){

int i, error = 0;

for (i = 0; i < 32; i++ ) {
  TIM_results[i] = GTM_AFD_0.CH[0].BUF_ACC.R;
  if (TIM_results[i] != FIFO_values[i+1]){
    error = (error|1<<i);
  }
}

GTM_FIFO_0.CHANNEL[1].IRQ_NOTIFY.R = 0x0000000F; // Clear All IRQs
GTM_FIFO_0.CHANNEL[1].CTRL.R = 0x00000004; // Flush FIFO

}
```

8 Example 7: Writing, Compiling, and Programming MCS Code

8.1 Description

The HighTec™ assembler tool generates machine code for the Multi Channel Sequencer (MCS) submodule. The assembler takes a user-created assembler source file (.mcs) for a specific instantiation of MCS and creates the executable and linkable format file (.elf) that is used to initialize the memory of the MCS instantiation in the chip environment. The HighTec tool also requires an include file (.inc) that includes predefined architecture specific definitions (instruction definitions, for example), a list of predefined assembler symbols and the ARU write address labels (which should not be modified), and a linker description file (.lin) where the memory description of the GTM/MCS is defined.
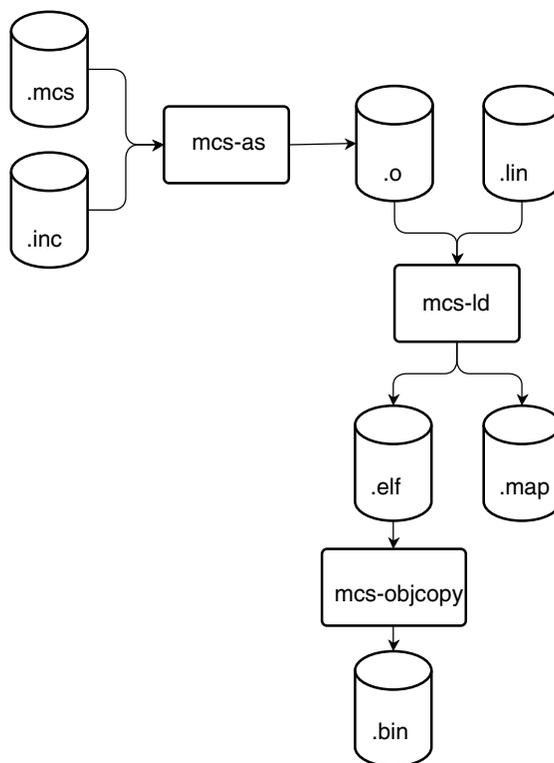
Figure 8. MCS code process flow chart

NOTE

The HighTec MCS assembler generates an ELF file that is big endian, whereas the
MPC57xx chip is little endian.

NOTE

Depending on the C compiler used for the MPC57xx chip and the project environment,
you may wish to transform the .elf file with the GNU utility mcs-objcopy in to a raw
binary file (.bin) for direct import in to the chip. This example does this and gives an
example of one possible way in which this can be incorporated it in to a Green Hills™
project.

8.2 Implementation

Follow these nine steps to generate MCS code and data for each MCS instantiation. That is, if you are using all four MCS
modules, these steps must be repeated for each MCS; however, all eight MCS channels can be configured and coded inside
one .mcs program.

1. Locate your latest HighTec installation directory (v4.6.1.2 or above), for example, C:\HIGHTEC\toolchains\ppc
   \v4.6.2.0
2. Write the MCS assembly code program. An example can be found in the software package that accompanies this
   application note, inside the QOM example folder (for example, MCS0_QOM.mcs). The GTM-IP specification
   described the assembler instruction operations.
3. Copy the *.mcs file, *.inc file, and *.lin in to the directory.
4. Type cd <HIGHTEC_MCS_DIR>\bin in the Windows command shell, where <HIGHTEC_MCS_DIR> refers the
   directory containing the installation located above in step 1.
5. Type mcs-as -o mcs0_qom.o MCS0_QOM.mcs in the Windows command shell to generate the object file mcs0_qom.o
   or your own example.
6. Check the directory to ensure mcs0_qom.o was generated.

Example 7: Writing, Compiling, and Programming MCS Code

7. Type mcs-ld -o mcs0_qom.elf -dT mcs.lin --extmap=a -Map=mcs.map mcs0_qom.o in the Windows command shell to generate the elf file mcs0_qom.elf or your own example.

> **NOTE**
> If you want a binary instead of an ELF file, use mcs-objcopy -O binary mcs0_qom.elf mcs0_qom.bin to convert the file format.

8. Copy the resultant file (.elf or .bin) from the HighTec directory into your project directory and include the file in your project environment.
9. Add software to copy the big endian array in to the little endian MCS memory.

Table 8.   MCS assembler process

Step	Operation	Description
1	Download and install HighTec	Locate your latest HighTec installation directory (v4.6.1.2 or above)
2	Write MCS algorithm	Write the MCS assembly code program
3	Move files	Copy the *.mcs file, *.inc file, and *.lin in to the directory
4	Open command prompt in HighTec directory	Type cd <HIGHTEC_MCS_DIR>\bin in the windows command shell
5	Create object code	Type mcs-as -o mcs0_qom.o MCS0_QOM.mcs in the Windows command shell to generate the object file mcs0_qom.o
6	Check for *.o generation	Check the directory to ensure mcs0_qom.o was generated
7	Generate *.elf file	Type mcs-ld -o mcs0_qom.elf -dT mcs.lin --extmap=a -Map=mcs.map mcs0_qom.o in the windows command shell to generate the elf file mcs0_qom.elf
8	Move files from HighTec to project	Copy the Executable and Linkable Format file (.elf) from the HighTec directory in to your project directory and include the .elf file in your project environment.
9	Write device code to copy the MCS code devices RAM	Add software to copy the big endian MCS assembly array in to the MCS's local RAM memory

8.3   Code

Write the MCS assembly code program. The code below is a HighTec assembly code example.

```
# Prepare assembler for MCS memory
# -----------------------------------
.arch mcs
.set memid ,0
.set memsize ,0x1800

# Define some constants
# -----------------------
.set EN_H_MSK ,0x000001
.set EN_L_MSK ,0xFFFFFE
.set IRQ_H_MSK ,0x000002
.set IRQ_L_MSK ,0xFFFFFD

# Initialize reset vectors of MCS channels 0 and 1
# ------------------------------------------------
.org 0x0
jmp tsk0_init
jmp tsk1_init
jmp tsk2_init
jmp tsk3_init
```

```
jmp tsk4_init
jmp tsk5_init
jmp tsk6_init
jmp tsk7_init

# Allocate and initialize memory variables
# ----------------------------------------
.org 0x20
mem_var_a: .lit24 17 # define memory variable mem_var_a with value of 17

# Allocate stack frames (each task has 16 memory locations)
# ---------------------------------------------------------------
.org 0x20
tsk0_stack:.lit24 0
.org 0x60
tsk1_stack:.lit24 0
.org 0xA0
tsk2_stack:.lit24 0
.org 0x100
tsk3_stack:.lit24 0
.org 0x140
tsk4_stack:.lit24 0
.org 0x180
tsk5_stack:.lit24 0
.org 0x1C0
tsk6_stack:.lit24 0
.org 0x200
tsk7_stack:.lit24 0

# Program entry for MCS-channel 0
# --------------------------------
tsk0_init:
movl R7, tsk0_stack      # initialize stack pointer
mrd  R4, tsk0_config  # load config value to register R0
btl r4, QOM_REF_MODE_RAM
jbc STA, Z, tsk0_ref_mode_ram
…

# ----------------------
tsk0_ref_mode_ram:
mrd r3, tsk0_ref          # ram value as reference
 jmp tsk0_reference_created
…

# Program entry for MCS-channel 1
# --------------------------------
tsk1_init:
movl R7, tsk1_stack      # initialize stack pointer
mrd  R4, tsk1_config  # load config value to register R0
btl r4, QOM_REF_MODE_RAM
jbc STA, Z, tsk1_ref_mode_ram
…etc.
```

Add software in to the CPU program to copy the assembly code n to the MCS memory

```
#define MCS_RAM_ENTRIES 1024

#define SWAPW(w) \
      (((w & 0xff) << 24) | ((w & 0xff00) << 8) \
       | ((w & 0xff0000) >> 8) | ((w & 0xff000000) >> 24)) /* change endianness */

extern int __MCS0_ADDR; /* Label of location of the raw data set in the linker */

// load raw bin data in to MCS0 RAM = 0xFFD38000
dest = (int)&MCS0_MEM; /* CPU view of the address of the MCS memory space */
src = (int)&__MCS0_ADDR; /* Label of location of the raw data set in the linker */
memcpy_swap_word(dest, src, MCS_RAM_ENTRIES);

void memcpy_swap_word(unsigned int * dst, unsigned int * src, signed int size)
{
```

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

```
while (size-- > 0)
   {
     *dst++ = SWAPW(*src);
     src++;
   }
}
```

NOTE

The syntax difference between the Bosch™ CASPR MCS assembler and the HighTec
assembler are as follows:

- Comments start with **;** (caspr) vs **#** (HighTec)
- Hexadecimal prefixed with **$** (caspr) vs **0x** (HighTec)
- Operands separated with a **blank** (caspr) vs **,** (HighTec)
- Constants defines with **.define** (caspr) vs **.set** (HighTec)
- 24-bit literal values defined with **lit24** (caspr) vs **.lit24** (HighTec)

9 Example 8: Queued Output Match (QOM)

Previous Freescale timer libraries have often contained a Queued Output Match (QOM) function that generates complex output pulse trains without CPU intervention using a sequence of output matches.

An output match occurs when a user-defined value is matched by the value of an internal timebase. When an output match occurs, a user-specified pin state is driven on the output pin.

The GTM QOM function, like the eTPU QOM function described in Freescale application note AN2857, generates multiple output matches using a table of offset times. These offset times, along with the corresponding pin states, are stored in Data Memory. The table size is user-programmable. Various modes of queue operation are supported.

9.1 Description

The following example contains Multi-Channel Sequencer (MCS) code which, together with the ATOM channel, provides the QOM functionality.

Entries in the QOM queue (event table) are relative match offsets, not absolute match times. The next match time in a sequence is calculated by adding the next offset in the table to the time of the last match. If the match is the first match in a sequence, the first offset value in the table is added to a selectable reference time.

The reference from which the first match in a sequence is scheduled can be the immediate value of the selected timebase, the time of the last match of a previous sequence, or a reference contained in Data Memory. Using the time of the last match of a previous sequence as a reference allows a series of sequences to be chained together. Using a reference from Data Memory allows a sequence of output matches to be referenced to a value supplied by another GTM channel.

Pin state (high or low) when a match occurs is programmable. Pin state is determined by the value of the LSB in each table entry.

The function can operate in three modes: Single-shot mode, Loop mode, and Continuous mode. In Single-shot mode, a sequence of match outputs is generated once. In Loop mode, a sequence of match outputs is generated a specified number of times (1 to 256). In Continuous mode, the entire sequence repeats until the channel is disabled.

In Single-shot and Loop modes, the event time of the last match in the table is written back into Data Memory, which can be accessed by the CPU. During initialization, the pin can be configured to be high, low, or no change. Matches are scheduled using the GTM's ATOM units. Each match offset can have a maximum value of 0x40_0000 counts. This allows the second future match to be up to 0x80_0000 counts in the future. If more than two table offset values are programmed for the same pin state, the duration of an output event can effectively be extended beyond the normal 0x80_0000 count limit.

Figure 9 below shows the main components of the QOM function and how these are constructed. The system consists of a number of inter-linked functions that are realized using submodules of the GTM configured to achieve the particular task. An overview of a single instance of the components of the system is shown in the following figure.
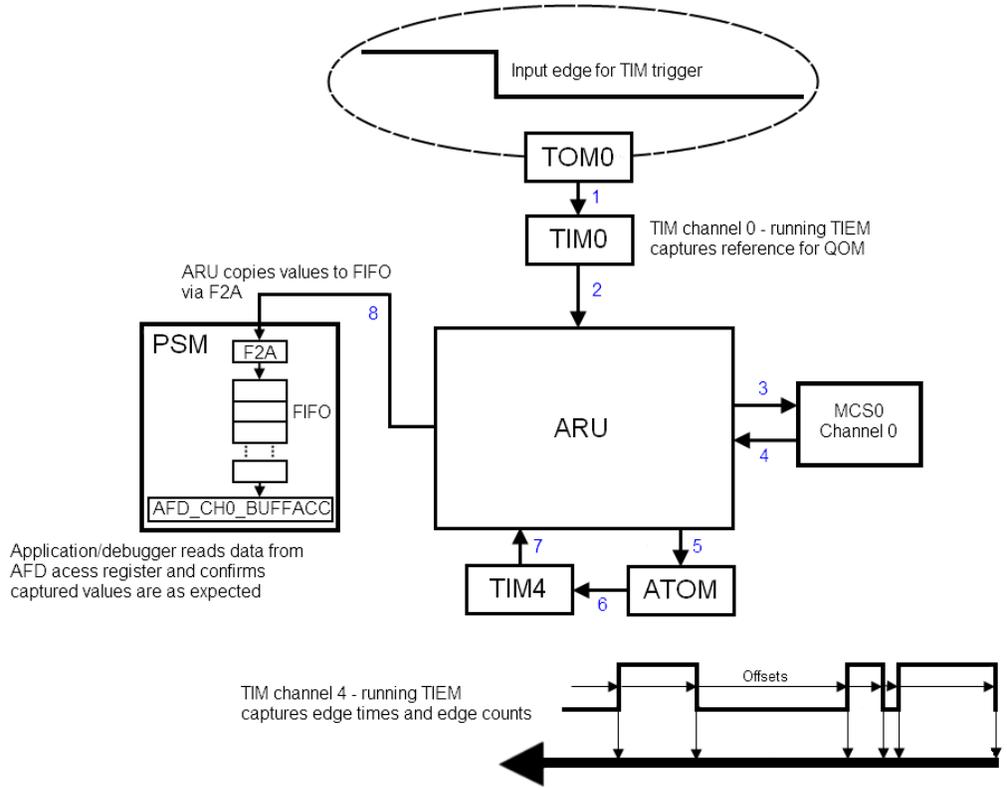

Figure 9. QOM implementation overview

Table 9.   QOM functional stages

Step	Operation	Description
1	Generate input edge	Generate a falling edge on TOM0
2	Capture edge event	Wait for a falling edge on TIM0
3	Route event to MCS	Send edge event to MCS0
4[1]	Process data	Calculate the next command for the ATOM submodule from the match table in RAM
5	Send command to ATOM	Send that data out for the ATOM
6	Read in waveform through TIM	Read the ATOM channel back in to the GTM through the TIM
7	Read in edge times	Move the edge times in to the ARU
8	Send the out of GTM via FIFO	Place the values in FIFO for CPU to test

1.   Steps 4 and 5 are repeated 6 times in this example

Step 1: TIM channel 0 is used in Timer Input Event Mode (TIEM) One-shot mode to capture the TBU_TS0 values when the TOM0 channel 0 signal goes low. A physical external wire connection is needed on the evaluation board between the TOM output pin and the TIM input pin.

Step 2 and 3: The Advanced Routing Unit (ARU) sends the TBU_TS0 value captured by the TIM to the Multi Channel Sequencer (MCS) channel. The MCS channel is running a rudimentary basic Queued Output Match(QOM) function (more details later). After the MCS channel is enabled, it waits for the ARU to provide the capture value from TIM channel 0.

Step 4 and 5: When this value becomes available to the MCS channel, the QOM function calculates a match value, relative to the TIM capture value, based on a table of offsets and uses the ARU to write the match value and future pin state to the ARU-connected Timer Output Module (ATOM channel).

When this match and subsequent matches occur, the MCS channel waits for a match value to be sent via the ARU from the ATOM channel. The ATOM channel has been configured in Signal Output Mode Compare (SOMC) mode.

Step 6, 7 and 8: A wired connection is needed between the ATOM QOM pin and the TIM pin reading it back in. This TIM channel is running in TIEM mode and copies the captured values and edge count via the AFD submodule to a FIFO.

Values can be read from the FIFO by the CPU. Because the QOM matches were scheduled relative to the TIM edge time (TBU_TS0 value), it is possible to check that the values read from the FIFO are correct by software.

The described systems consists of four input trigger channels (TIM0 channels 0–3), four ATOM channels (ATOM0_channel 0–3), four MCS channels, four TIM channels for capturing the edge times and counts (TIM0_channel 4–7), four AFD streams, and four FIFOs. For simplicity, only the first instance of each type is described. The remaining channels are operating in a similar way, only the time of the trigger event differs between them.

9.1.1   ARU function description

The ARU's role is to transfer data from a source submodule to destination submodules. The destination submodule controls the data stream configuration inside the ARU. That is, the destination submodule defines where its data will be sourced from; the ARU just enables that particular data stream, as shown in ARU function description. The routing is done in a deterministic manner with a basic round-robin scheduling scheme of connected channels that always receive 53 bits of data (two 24 bits of data and 5 control bits) through the ARU.

Data is only transferred once as as soon as the ARU reads the data from the source for delivery to the destination that data is marked as invalid at the source; in other words, the ARU access is a destructive read. This also means that if the source does not supply new data, the destination will not receive any further information. There is a worst case round-trip time associated with the data transfer.
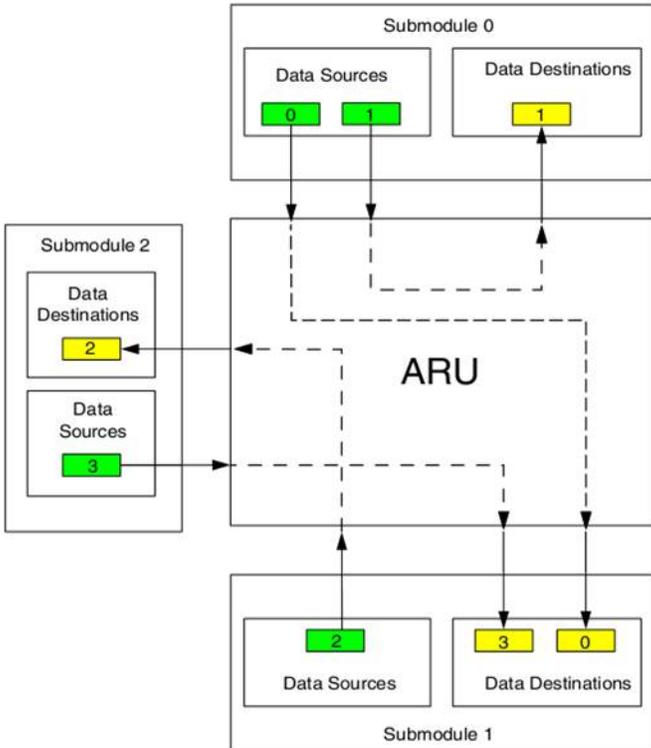
Figure 10. GTM routing unit

The specific round-trip delay for a given system depends upon the number of destinations connected to the ARU.The maximum round-trip time can be found in the GTM's *Appendix B* document (see GTM References).

Each data source has a fixed and unique source address. These are also defined in *Appendix B*. To configure a transfer to a specific destination, the source address of the data must be written to the destination's configuration registers.

The ARU supports a blocking mechanism to synchronize submodules to the routed data streams. This blocking mechanism means that it is possible for a channel to become inactive as long as no new input data is available for the channel. This means that there is no interrupt to inform the destination that new data is available at the source. The destination will be given new data by the ARU from the source and will continue to run without interrupting the CPU. In summary, 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.

This concept is used in the example system to enable TIM channel 0 to provide a reference for the MCS channel (step 3 from figure 10). The MCS channel also waits until the match event has occurred on the ATOM channel before deriving the next match time and writing it to the ATOM hardware (step 5 from figure 10) and the TIM channel 4 provides the captured waveform to the ARU for the PSM system to record (step 7 from figure 10)

Refer to the *GTM Architecture* and *Advanced Routing Unit (ARU)* chapters of the *Generic Timer Module (GTM) Reference Manual* for more information on the ARU operation.

9.1.2   TIM function description

The Timer Input Module (TIM) is responsible for the input signal capture and characterization in the GTM. The TIM channels have a dedicated filter mechanism with different filter strategies and edge filter thresholds for each channel. The TIM has shadow registers to hold measurement data while a new input signal is processed. The submodule can be controlled by the CPU or the ARU with five different, configurable, edge characterization modes. The TIM channels can work totally independently of each other in different operation modes.

In this example, the TIM channels are all operating in Timer Input Event Mode (TIEM). In TIEM, edges at the TIM input channel can be characterized with either two different time stamps or a time stamp and an edge counter. It is configurable whether both edges, only rising, or only falling edges should be considered as an input event. Measurement starts with the first relevant edge. This example captures the falling edge immediately, with filters disabled, and captures the Time Base Unit Time Stamp 0 (TBU_TS0) value at that event.

9.1.3 MCS function description

The MCS code along with the ATOM channel and ARU realize the bulk of the QOM functionality as described earlier.

A jump table based at location 0 defines program entry points for each of the MCS channels used in the example system, based on the reset value of the channel's program counter.

The MCS chapter of the *Generic Timer Module (GTM) Reference Manual* has further details on the assembler instructions.

9.1.4 ATOM function description

The ATOM submodule is used to output the signal calculated by the MCS submodule based on the TIM input edge.

9.1.5 F2A function description

The FIFO functionality in the QOM example is not essential to the example's operation. It is just a way for the timestamp values of the waveform output from the ATOM to be recorded to be read by the user, or by the CPU, to test that the output waveform was as expected rather than by testing by visual inspection of the output waveform on an oscilloscope.

The F2A is a part of the GTM parameter storage module (PSM) mechanism. The PSM is used to transfer data to and from the GTM in a buffered way. This decreases the CPU's interrupt load because the data is stored inside the PSM. Interrupts are only generated when programmable thresholds are reached. The PSM has an ARU interface and can act as source and destination at the ARU. The PSM can be organized as a FIFO where the data is transferred in first-in-first-out order.

9.2 Implementation
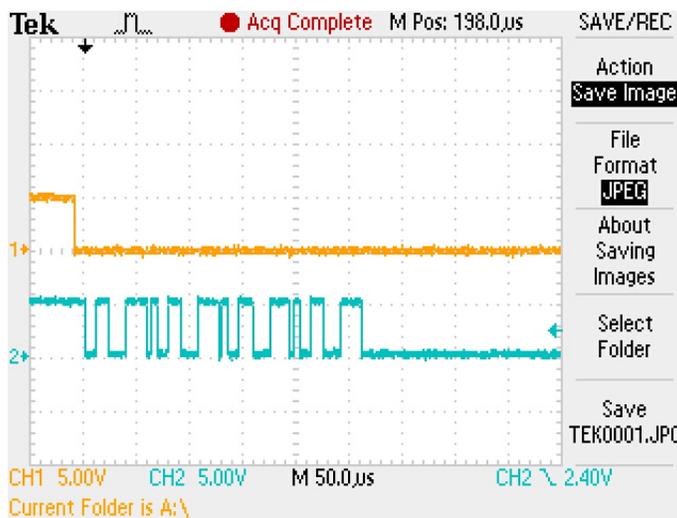
Figure 11. Queued Output Match

A table of match offset values and pin states resides in the RAM of the GTM. These will be used by the MCS and then ATOM to generate the output waveform.

The following table shows the values stored in MCS RAM to generate the above QOM.

Table 10.   TBU_TS0 match table

RAM location	Offset time value	RAM content	Description
1	0x100	0x200 (0x100<<1+pin_state)	After 256 timer counts output goes low
2	0x200	0x401 (0x200<<1+pin_state)	After 512 timer counts output goes high
3	0x300	0x600 (0x300<<1+pin_state)	After 768 timer counts output goes low
4	0x400	0x801 (0x400<<1+pin_state)	After 1024 timer counts output goes high
5	0x500	0xA00 (0x500<<1+pin_state)	After 1280 timer counts output goes low
6	0x600	0xC01 (0x600<<1+pin_state)	After 1536 timer counts output goes high

Matches are on a single timebase only hard-coded to be TBU_TS0 (could be configured to be selectable via configuration parameter).

The matches can begin :

- Immediately
- Because another channel provided a trigger value via the ARU, as shown in Figure 11
- Relative to some timer counter value (reference stored in RAM)

The MCS works its way through the table scheduling the ATOM output events. The whole process can be repeated:

- Once (Single-shot mode), as shown in Figure 11
- A number of times (Loop mode)
- Continuously (forever; until the MCS channel is disabled by the host or reset).

9.2.1   TIM trigger capture implementation

The steps to configure the TIM channel 0 (TIM0) in Timer Input Event Mode (TIEM) for use in this QOM functions are shown in the following table. The same steps can be used for each channel used in the example depending on how many channels you wish to use.

The TIM channel must be set up by the CPU to operate in the required manner. When a falling edge occurs on the input pin, the 24-bit TBU_TS0 value is copied to TIM0_CH0_GPR0. The TIM channel is clocked from the CMU_CLK0 clock source with no filtering applied to the input signal. The ARU can then transfer these TBU_TS0 values to MCS channel when an ARU read instruction is issued by the MCS channel; therefore, the ARU connection must be enabled. Note that the ARU source address is configured in the MCS assembler code (more on this later).

NOTE
Although in this example the input filters have been disabled, typical applications should enable the input filter to eliminate noise on the input signal.

The resultant behavior of this configuration is that when a falling edge occurs on the input pin, the TBU_TS0 value is copied to TIM0_CHn_GPR0 to be read by the ARU.

Table 11.   TIM0 channel configuration settings

Bit	Bit number	Value	Description
TOCTRL	[31:30]	0	Timeout feature disabled
EGPR1_SEL	29	0	—
EGPR0_SEL	28	0	—
FR_ECNT_OFL	27	0	Overflow will be signaled on ECNT bit width = 8
CLK_SEL	[26:24]	0b00	CMU_CLK0 selected
FLT_CTR_FE	23	0	Up/down counter
FLT_MODE_FE	22	0	Immediate edge propagation mode
FLT_CTR_RE	21	0	Up/down counter
FLT_MODE_RE	20	0	Immediate edge propagation mode
EXT_CAP_EN	19	0	External capture disabled
FLT_CNT_FRQ	[18:17]	0b00	FLT_CNT counts with CMU_CLK0
FLT_EN	16	0	Filter disabled
ECNT_RESET	15	0	ECNT counter operating in wrap around mode
ISL	14	1	Do not use DSL bit for selecting active signal level
DSL	13	0	Measurement starts with falling edge (low-level measurement)
CNTS_SEL	12	0	Use CNT register as input
GPR1_SEL	[11:10]	0b11	Use CNT as input
GPR0_SEL	[9:8]	0b00	Use TBU_TS0 as input
CICTRL	6	0	Use signal TIM_IN(x) as input for channel x
ARU_EN	5	1	Enabled
OSM	4	0	Continuous Operation
TIM_MODE	[3:1]	0b010	Input Event mode
TIM_EN	0	1	Enabled

Below are the steps required to set up TIM0 channels 0 and 4, including the CMU and TBU clock enabling.

Table 12.   TIM Configuration

Step	Operation	Description	Psuedocode
1	Enable CMU_CLK0	Write 0b10 to the CLK0 bit field and disable all others by writing 0b01	CMU_CLK_EN = 0x00555556;
2	Enable the TBU_TSn channels	Write 0b10 to CH0 and CH1 bit fields and disable CH2 by 0b01	TBU_CHEN = 0x0000001A;

Table continues on the next page...

Table 12. TIM Configuration (continued)

Step	Operation	Description	Psuedocode
3	Configure the TIM channels	Define and set the require TIM channel configuration by writing the configuration to the channel control register	TIM0_CHn_CTRL = 0x00004c25;

9.2.2   MCS function implementation

The flow chart in Figure 12 shows the nine fundamental functions of the MCS software in this QOM example.
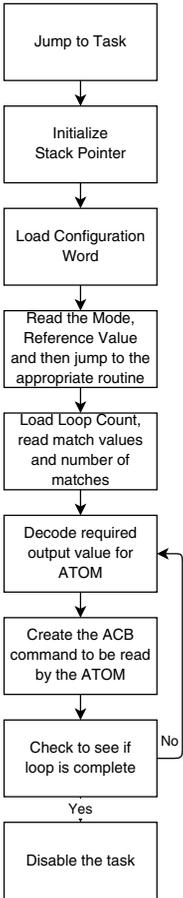
Figure 12. QOM MCS assembly code flow

Table 13.   QOM MCS functional steps

Step	Operation	Description	Code Reference
1	Jump to task	Move PC to start of code	jmp tsk*n*_init
2	Initialize stack pointer	Tell the MCS where the configuration data is located	movl R7 tsk*n*_stack

Table continues on the next page...

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

Table 13. QOM MCS functional steps (continued)

Step	Operation	Description	Code Reference
3	Load configuration word	Read the configuration word in to a general use register	mrd R4 tsk*n*_config
4	Read mode data from word	Check which bit is set in the tsk*n*_config word	btl R4 *MODE*
5	Load loop and match data	Find out how many loops are left and what the match data will be this time	tsk*n*_reference_created
6	Define the required action	For that loop/match value should the ATOM be instucted to toggle?	tsk*n*-inner_loop
7	Write CM0 and ACB to the ARU	Provide the ARU data stream with the calculated ATOM actions data	tsk*n*_derive_new_match_value_1
8	Has all data been sent	Has the QOM sequence completed	jbc STA Z tsk*n*_Outer_loop
9	Disable task	Stop further actions requests from TIM	andl STA EN_L_MSK

Configuration word: Each task has its own software configuration word stored as data in the MCS RAM. Task0 has a software configuration parameter called tsk0_config. The user selects how the task will behave by setting bits in the configuration word. In the configuration parameter, only one the following reference mode selection bits should be set:

- Bit 0 selects RAM reference mode if set
- Bit 1 selects Immediate reference mode if set
- Bit 2 selects Trigger reference mode if set

Bit 4 of the configuration parameter tsk0_config selects continuous mode if set; otherwise, the task will be in Loop mode if bit 4 is clear. If Loop mode is chosen, then the parameter tsk0_loop_count is used to determine how many loops are executed.

Match count parameter: The match count parameter is called tsk0_n_matches. It determines how many match events are in the table of events.

RAM reference parameter: The RAM reference parameter is called tsk0_ref. It is used only in RAM reference mode.

Match table: Match events are defined by parameters tsk0_match0…tsk0_matchN where N = tsk0_n_matches – 1.

Each tsk_matchx value equals (TBU_TS0 offset count at which the event will occur) × 2 + pin state when the match happens.

For example, tsk0_match0: lit24 0x201 means that after 0x100 counts (from a reference time), drive the output pin high as described previously in the TBU_TS0 Match table.

First match event: Depending upon the reference mode which has been chosen the first match time is calculated.

- If the reference mode is RAM reference mode, then the value stored in `tsk0_ref` is used.
- If immediate mode is chosen, then `TBU_TS0` is read and that value is used as reference.
- If trigger mode is chosen, then an ARU read is performed. The MCS waits until the ARU receives data from TIM0 channel 0, `TIM0_WRADDR0`. In this example, this occurs when the first falling edge happens on that TIM channel. The ARU copies across the captured value of TBU_TS0 when the falling edge happened.

A match value is calculated relative to the previously derived reference value, then an ARU write command is issued. This copies the match value into the CM0 register on ATOM0 channel 0 (via the ARU) along with the ACB value.

- ACB42 are the upper 3 bits in the ARU Control bits and are transferred to the ATOM channel along with the match value on an ARU write
- The ACB42 is defined as 8, which means compare CCU0 only on TBU_TS0. Depending upon the least significant bit of tsk_match0, 1 or 2 is added to ACB42 giving the full 5 bit ACB value for the ATOM module

- ACB = 0x9 (8+1) means the ATOM pin will be driven low on a match event, compare CCU0 only on TBU_TS0
- ACB = 0xA (8+2) means the ATOM pin will be driven high on a match event, compare CCU0 only on TBU_TS0

An ARU read is now performed. This effectively stalls the MCS channel until the previously scheduled match has occurred and the match value is copied via the ARU to the MCS channel.

Subsequent matches are scheduled relative to the match value read by the ARU from the ATOM channel.

A loop counter is used to control when the process stops (in Loop mode only; in Continuous mode the MCS channel will continue scheduling matches until the channel is disabled or reset).

A pointer is controlled so the next tsk0_matchx value is read. In Loop and Continuous modes, this must be reset to point to tsk0_match0 at the beginning of each loop.

9.2.3   ATOM function implementation

The ARU source address must be written into the ATOM channel's Read Address register. In the case of ATOM0 channel 0 in this example, the read address register is populated with the ARU address corresponding to the MCS channel responsible for generating the ATOM channel's match values (MCS0 channel 0).

ATOM channel 0 and MCS channel 0 communicate via the ARU address MCS0_WRADDR0.

For this QOM function, the ATOM channel is configured in Signal Output Mode Compare (SOMC) mode with the ARU enabled. ARU Blocking mode is enabled. This means that after updating CM0 and CM1 via the ARU, no new data is read from the ARU until a compare match event has occurred and SR0 and/or SR1 are read.

The output enable and enable bit for the channel are set in ATOM0_AGC_OUTEN_CTRL and ATOM0_AGC_ENDIS_CTRL.

These bits are copied to ATOM0_AGC_OUTEN_STAT and ATOM0_AGC_ENDIS_STAT by issuing a host trigger achieved by setting the HOST_TRIGGER bit (bit 0) in the ATOM0_AGC_GLB_CTRL register.

Table 14.   ATOM configuration

Step	Operation	Description	Psuedocode
1	Set up the channel	Define appropriate triggers, modes, and clock sources for the function by writing to the ATOM Channel Control register	ATOM0_CH0_CTRL = 0x08000809;
2	Enable the channels to output on the next trigger update	Write 0b10 to each channel's bit field in the AGC_OUTEN_CTRL register	ATOM0_AGC_OUTEN_CTRL = 0x0000AAAA;
3	Enable the channel update mechanism	Write 0b10 to each channel's bit field in the AGC_ENDIS_CTRL register	ATOM0_AGC_ENDIS_CTRL = 0x0000AAAA;
4	Trigger an update of the channels	Write 1 to the HOST_TRIG bit	ATOM0_AGC_GLB_CTRL = 0xAAAA0001;

The *Generic Time Module (GTM) Reference Manual* contains a detailed explanation of how SOMC mode works. Note that the MCS code configures the ACB bits so that only CCU0 and TBU_TS0 are used. This was done to keep the example system relatively simple and easier to understand.

9.2.4   F2A function implementation

In this case, the source address for the ARU read is written to the F2A register interface. This is the ARU address of GPR0/1 registers for TIM0 channel 4.

Stream 0 is enabled and configured to deal with data from TIM0 channel 4.

The data transfer direction is being set to "Transport from ARU to FIFO" and transfer mode is being set to "Transfer both words from/to FIFO." This means that both the count and match value from the TIM channel will be transferred to the FIFO via the ARU.

The CPU can read out the FIFO by accessing the AFD register for that particular FIFO channel.

9.3   Code

9.3.1   TIM trigger capture code

The following code is the I/O processor code required to set up TIM0_CH0 and TIM0_CH4, including the associated CMU and TBU clock enable.

```
GTM_CMU.CLK_EN.R = 0x00555556; // only enable CMU_CLK0
while (GTM_CMU.CLK_EN.R != 0x00000003); // confirm only CMU_CLK0 is on

GTM_TBU.CHEN.R = 0x0000002A; // Enable all TBU channels
while (GTM_TBU.CHEN.R != 0x0000003F); //confirm all are enabled

/* TIM Trigger Capture */
GTM_TIM_0.CH0_CTRL.R = 0x00004c25; //configure TIM0_CH0
while(GTM_TIM_0.CH0_CTRL.R!=0x00004c25); //confirm the configuration is effective

/* TIM Output Capture */
GTM_TIM_0.CH4_CTRL.R = 0x00004c25; //configure TIM0_CH4
while(GTM_TIM_0.CH4_CTRL.R!=0x00004c25); //confirm the configuration is effective
```

9.3.2   MCS code

In the example system, all eight MCS channels have been configured to do the same thing. Task 0 (the task assigned to MCS0 channel 0) is discussed here but can be read for any of the other channels used.

The MCS assembly code below:

```
.org $0
jmp tsk0_init
jmp tsk1_init
jmp tsk2_init
jmp tsk3_init
jmp tsk4_init
jmp tsk5_init
jmp tsk6_init
jmp tsk7_init
```

compiles into machine code that looks like the following:

```
0xe0000380 ,      /* JMP    0x0380 */
0xe000040c ,      /* JMP    0x040C */
0xe0000498 ,      /* JMP    0x0498 */
0xe0000524 ,      /* JMP    0x0524 */
0xe00005b0 ,      /* JMP    0x05B0 */
```

```
0xe000063c ,      /* JMP    0x063C */
0xe00006c8 ,      /* JMP    0x06C8 */
0xe0000754 ,      /* JMP    0x0754 */
```

The MCS code below sets up the data for the MCS to use:

```
tsk0_config: .lit24 18#= QOM_REF_MODE_IMMED + QOM_LOOP_MODE_CONTINUOUS
tsk0_loop_count: .lit24 4 #
tsk0_n_matches: .lit24 6 # number of events
tsk0_ref:    .lit24 0x300
tsk0_match0: .lit24 0x200 # match_value * 2 + pin_state
tsk0_match1: .lit24 0x401
tsk0_match2: .lit24 0x600
tsk0_match3: .lit24 0x801
tsk0_match4: .lit24 0xA00
tsk0_match5: .lit24 0xC01
```

and compiles into data that looks like this:

```
0x00000012 ,      /* NOP */
0x00000004 ,      /* NOP */
0x00000006 ,      /* NOP */
0x00000300 ,      /* NOP */
0x00000200 ,      /* NOP */
0x00000401 ,      /* NOP */
0x00000600 ,      /* NOP */
0x00000801 ,      /* NOP */
0x00000a00 ,      /* NOP */
0x00000c01 ,      /* NOP */
```

at the address specified by the programmer by `org $20 tsk0_stack:lit24 0`

Below is the remainder of the MCS assembly code described in MCS function implementation.

```
tsk0_init:

movl R7,tsk0_stack      # initialize stack pointer

mrd  R4,  tsk0_config  # load config value to register R0

btl r4, QOM_REF_MODE_RAM
jbc STA, Z, tsk0_ref_mode_ram

btl r4, QOM_REF_MODE_IMMED
jbc STA, Z, tsk0_immediate_mode

btl r4, QOM_REF_MODE_TRIG
jbc STA, Z, tsk0_ref_mode_trig

tsk0_immediate_mode:
  mov r3, TBU_TS0            # capture current TBU_TS0 value
  jmp tsk0_reference_created

tsk0_ref_mode_ram:
  mrd r3, tsk0_ref          # ram value as reference
  jmp tsk0_reference_created

tsk0_ref_mode_trig:
  ard  r3, ZERO, TIM0_WRADDR0 # read tim0 ch0 capture value from ARU

tsk0_reference_created:
  mrd   r1, tsk0_loop_count

tsk0_outer_loop:
  movl R5, tsk0_match0    # load address of match values to register R5
  mrd  R6, tsk0_n_matches  # load number of matches to register R6

tsk0_inner_loop:
  mrdi r0,  r5            # match offset + pin state to r0
```

Example 8: Queued Output Match (QOM)

```
  shr  R0, 1                  # shift right once to test LS bit
                              # r0 now has offset value
  jbc STA, 4, tsk0_drive_zero_1
  movl r2, (ACB42 + 1)        # configure acb to drive low on match
  jmp tsk0_derive_new_match_value_1

tsk0_drive_zero_1:
  movl r2, (ACB42 + 2)        # configure acb to drive high on match

tsk0_derive_new_match_value_1:
  add R0, R3                  # derive new match value
  mov  acb, r2                # acb42 to r2
  awr  r0, r1, MCS0_POOL0     # write CM0 &ACB to ARU#
                              # NB r1 value is not used by the hardware
                              # NB the third operand is related to a value
                              # populated in ATOMx_CHy_RDADDR
                              # third operand = valueof(ATOMx_CHy_RDADDR) - 0x77
  ard  r3, ZERO, ATOM0_WRADDR0  # wait and then read sr0 & ACB from ARU
  addl  R5,  4                # increment pointer
  subl r6, 1
  jbc STA, Z, tsk0_inner_loop     # are we done innerloop?

  btl r4, QOM_LOOP_MODE_CONTINUOUS # in continuous mode?
  jbc STA, Z, tsk0_outer_loop

  subl r1, 1
  jbc STA, Z, tsk0_outer_loop     # are we done - outerloop?

  orl STA, IRQ_H_MSK          # raise IRQ flag
  andl STA, EN_L_MSK          # disable task
```

Before this code will run in the GTM application the MCS channel must be enabled by the CPU by writing to the enable bit in the MCS Channel Control register.

```
MCS0_CH0_CTRL = 0x00000001; // enable the MCS channel
```

NOTE

The GTM specification lists two sets of MCS registers: internal and configuration registers. These two register sets are equivalent. The MCS configuration registers are the CPU view of the MCS internal registers. They act in the same way with two exceptions. When the CPU is accessing MCS[i]_CH[x]_R[6], the register cannot be written while an ARDI or NARDI command is pending. When the CPU is accessing MCS[i]_CH[x]_ACB, this register is only readable.

9.3.3   ATOM code

The code below is the I/O processor code required to set up ATOM0 channel 0.

```
GTM_ATOM_0.CH0_RDADDR.R = ((MCS0_WRADDR0<<16)+(MCS0_WRADDR0));

GTM_ATOM_0.CH0_CTRL.R  = 0x08000809; // SOMC, TB1_SEL=0, ARU_EN=1, SL=1, WR_REQ=0

GTM_ATOM_0.AGC_OUTEN_CTRL.R = 0x0000AAAA;
GTM_ATOM_0.AGC_ENDIS_CTRL.R = 0x0000AAAA;
  while (GTM_ATOM_0.AGC_ENDIS_STAT.R != 0x00000000);

GTM_ATOM_0.AGC_GLB_CTRL.R = 0xAAAA0001; // setting of bit 0
```

NOTE

Bitfields RDADDR0 and RDADDR1 are populated with the same value. In the example, RDADDR1 is never used. See the *Generic Timer Module (GTM) Reference Manual* for more details on when RADDR1 would be used.

9.3.4   F2A code

The following code is the I/O processor code required to set up F2A channels 1–3, including the mode and ARU read addresses.

```
GTM_F2A_0.CH_ARU_RD_FIFO[0].R = TIM0_WRADDR4; // used for reading in from TIM

GTM_F2A_0.CH_STR_CFG[0].R = 0x00020000;

GTM_F2A_0.ENABLE.R = 0x00000002;
```

10   Example 9: Using the DPLL for a Simple Micro Tick Function

This example shows how to configure the GTM submodules to perform micro tick generation on the MPC5777M devices. It is based on a Bosch Automotive Electronics application note on DPLL micro tick generation.

10.1   Description

The DPLL submodule is designed to generate a high-frequency micro tick signal based on one or more input signals with a lower frequency. These input signals can only be connected to the TIM0 submodule as the MAP submodule is required by the function.

To generate the micro ticks the DPLL needs information about the timing behavior of the input signals. This is done with time stamps that are provided by the TBU. The TBU generates these time stamps with a clock signal that is generated inside of the CMU. The DPLL generates the micro tick output signals, sub_incs, which are connected to the TBU and the CMU.

For 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. An action calculation is a request 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.

Further details on sub_inc signals and RAM usage can be found in the GTM specification.
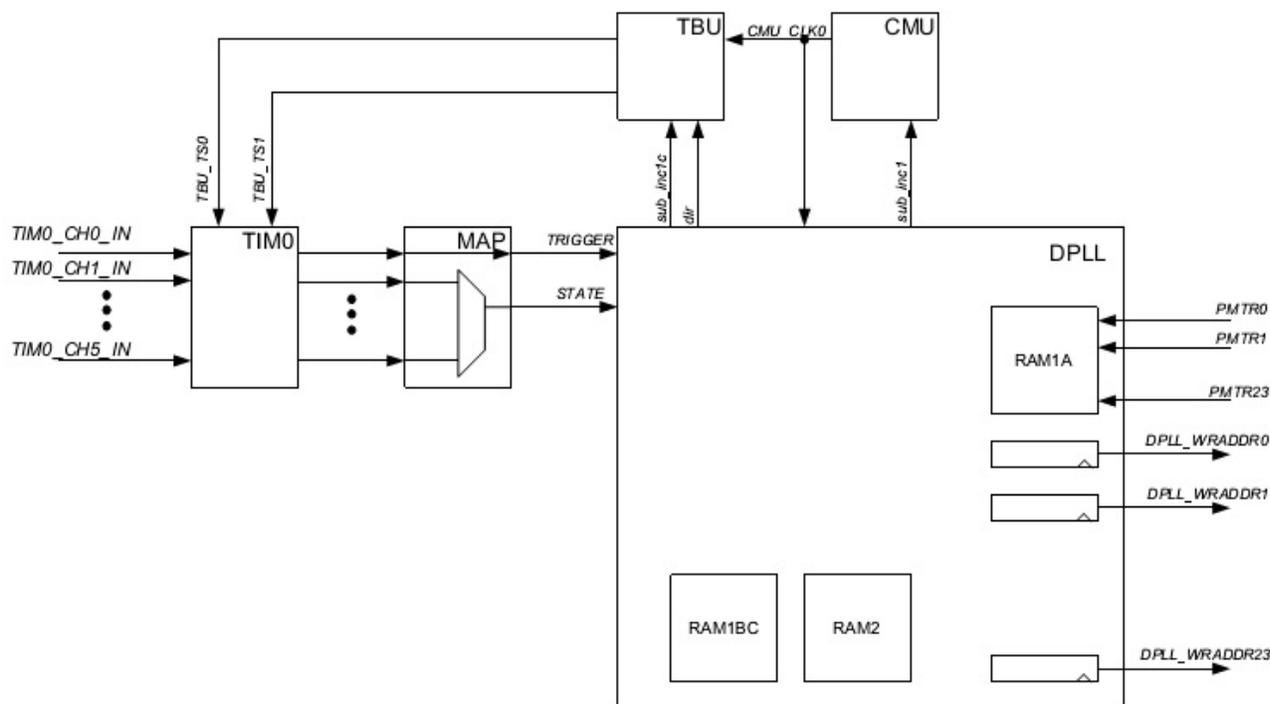
Figure 13. Micro tick example block diagram

10.2   Implementation

The following seven steps are the basic parts of the DPLL micro tick set up and function.

1. Configure GTM: Configure the device and the GTM as described in previous examples. 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.
2. Configure TBU: The TBU provides common time bases for the GTM. Typically, for an engine management system, the time stamp TBU_TS0 provides a 24-bit time based on CMU_CLK0 while TBU_TS1 provides the angle of the engine for the system in up/down counter mode from sub_inc1.
3. Configure TIM0: The Timer Input Module 0 is dedicated to sample and preprocess the input signals for the DPLL. The 49-bit wide signal is connected via the MAP submodule to the DPLL. To work properly, the DPLL needs specific settings inside the corresponding TIM channel: filtering and the time stamp format sampled inside of the TIM channel. The channel is configured to sample the high resolution time base TBU_TS0 for each incoming edge in the GPR0 register with no filtering in this example.
4. Configure MAP: When the signal preprocessing inside the TIM0 channel is done, the 49-bit data is transferred to the DPLL via the MAP module. Two data paths exist inside the MAP module. 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.
5. Configure DPLL: The micro ticks for the input signals TRIGGER and STATE are generated inside of the two DPLL 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 that generates a tick whenever the 24-bit accumulator register overflows.
6. Synchronize the tooth profile: Setting the DPLL input TRIGGER signal profile pointer APT_2c to the appropriate profile location. The DPLL profile synchronization can only be done when the input signal has at a specific location a specific pattern.
7. Monitor DPLL Lock: After the synchronization for the tooth wheel has been 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.

Table 15. High-level micro tick implementation steps

Step	Operation	Description
1	Configure GTM	Repeat examples 1, 2, and 3 with special consideration of the CMU
2	Configure TBU	TBU has to be configured to provide time stamps to characterize the DPLL TRIGGER input signal and to provide the micro tick time base
3	Configure TIM0	Configure TIM0_CHn in TIEM mode to react on both edges
4	Configure MAP	In this case, use default configuration of from TIM0 channel 0 to the DPLL TRIGGER input directly.
5	Configure DPLL	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.
6	Synchronize the tooth profile	Setting the DPLL input TRIGGER signal profile pointer APT_2c to the appropriate profile location
7	Monitor DPLL Lock	After the synchronization for the tooth wheel was established, the DPLL lock1 bit setting can be observed

More details on the steps involved in 5, 6, and 7 follow this section.

10.2.1   DPLL micro tick implementation

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. No adaptation of the tooth is considered in this example and no action generation takes place. Step 5 above is detailed further in steps 1 to 6 below.

1. Initialize the DPLL RAM Region 2c with the TRIGGER signal for the signal input profile for FULL SCALE.
2. Enable the Trigger Event 0 interrupt.
3. Configure the Timeout value.
4. Write DPLL_CTRL0 with the input signal characteristics (TRIGGER and STATE event characteristics, input filter characteristics etc).
5. Write DPLL_CTRL1 with the operation mode, micro tick generation, the time stamp resolution, and so on.
6. Enable the DPLL.

NOTE

Since there is no STATE signal input for this application note, the RAM Region 1c3 has not to be configured at all.

Table 16. Configure the DPLL

Step	Operation	Description
1	Initialize RAM Region 2c	Ensure RAM initialization is not already running and then fill the RAM with the tooth profile
2	Enable the Trigger Event 0 interrupt	Ensure RAM initialization is not already running and then fill the RAM with the tooth profile
3	Enable the Trigger Event 0 interrupt	Configure timeout value for actual TRIGGER slope
4	Write DPLL_CTRL0	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.
5	Write DPLL_CTRL1	DPLL_CTRL_1 register contains configuration bits for the DPLL operation. There, the behavior of the micro tick generators, the

Table continues on the next page...

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

Table 16.   Configure the DPLL (continued)

Step	Operation	Description
		number of virtual increments and the characteristic of the time stamps is defined.
6	Enable the DPLL	Set DEN bit in DPLL_CTRL1

The bitfield settings for DPLL_CTRL0 and DPLL_CTRL1 are shown in the following 2 tables.

Table 17.   DPLL_CTRL0 setting : 0x403A257

Bit Field / Position	Description	Setting	Value (dec)
MLT / [9:0]	Number of micro ticks between two TRIGGER events	Set to 599 to generate 600 micro ticks	599
IFP / 10	Filter value resolution	Filter is not used	0
SNU / [15:11]	Number of STATE events in HALF_SCALE	STATE is not used	0
TNU / [24:16]	Number of nominal TRIGGER events in HALF_SCALE	60 nominal TRIGGER events per revolution	60
AMS / 25	Adapt the tooth of STATE due to physical constraints	STATE is not used	0
AMT / 26	Adapt the tooth of TRIGGER due to physical constraints	No adapt values are taken into account	0
IDS / 27	Take input delay introduced by TIM0 filter into account for STATE	STATE is not used	0
IDT / 28	Take input delay introduced by TIM0 filter into account for TRIGGER	Filter is not used	0
SEN / 29	STATE input enable	STATE is not used	0
TEN / 30	TRIGGER input enable	Enable	1
RMO / 31	Configure which signal should be used for micro tick generation	TRIGGER only	0

Table 18.   DPLL_CTRL1 setting : 0x80020012

Bit Field / Position	Description	Setting	Value (dec)
DMO / 0	This bit defines the DPLL operation mode for micro tick generation	Automatic End Mode	0
DEN / 1	DPLL enable bit	Enable	1
IDDS / 2	Input direction detection strategy	The input direction is detected comparing the THMI value with the duration between valid and invalid slope of TRIGGER	0
COA / 3	Correction strategy for missing micro ticks, when the Automatic End Mode is chosen for micro tick generation	Missing micro ticks should be generated with the CMU_CLK0 clock frequency	0
PIT / 4	Plausibility window resolution definition	A time related plausibility window duration is considered	1

Table continues on the next page...

Table 18. DPLL_CTRL1 setting : 0x80020012 (continued)

Bit Field / Position	Description	Setting	Value (dec)
SGE1 / 5	This bit enables the sub_inc1 and sub_inc1c output signal line for micro tick generation	Start right after the synchronization condition is detected	0
DML1 / 6	In Direct Load Mode, the micro tick frequency is controlled by the CPU	Micro tick frequency should be calculated by the DPLL	0
PCM1 / 7	Pulse Correction	Not used	0
Various SUB_INC2 and STATE fields all not used [15:8]	Numerous SUB_INC2 and STATE and field unused and left out to shorten table	Not used	0
SYN_NT / [21:16]	This bit field summarizes the total number of virtual increments in a HALF_SCALE for the TRIGGER input signal	2 missing teeth per revolution	2
LCD / 22	Locking condition definition	1 missing trigger	0
SWR / 23	Software reset	Software reset	0
SYSF / 24	SYN_NS for FULL_SCALE	STATE not used	0
TS0_HRS / 25	Time stamp high resolution STATE	STATE not used	0
TS0_HRT / 26	Time stamp high resolution TRIGGER	Resolution of the used DPLL input TBU_TS0 bits is equal to the TRIGGER input time stamp	0
SMC / 27	Synchronous Motor Control	TRIGGER and STATE inputs are used for a control different to SMC	0
SSL / [29:28]	STATE slope select	STATE not used	0
TSL / [31:30]	The active TRIGGER slope has to be defined by these two bits	Falling edges	2

NOTE

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.

10.2.2 Synchronization and lock

The tooth synchronization and DPLL lock are executed in the TIM0_CH0 NEWVAL interrupt and the DPLL TRIGGER event interrupt 0.

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 newly synchronized profile.

This synchronization can only be done when the input signal is at a specific location of a specific pattern defined in the DPLL RAM. For this example, the specific pattern is two missing teeth in a normal 60 tooth wheel (only 58 real teeth available).

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. Gap detection is done on the last 2 tooth periods. If the tooth period before the last tooth period is twice the size, this preceding tooth period was the gap.
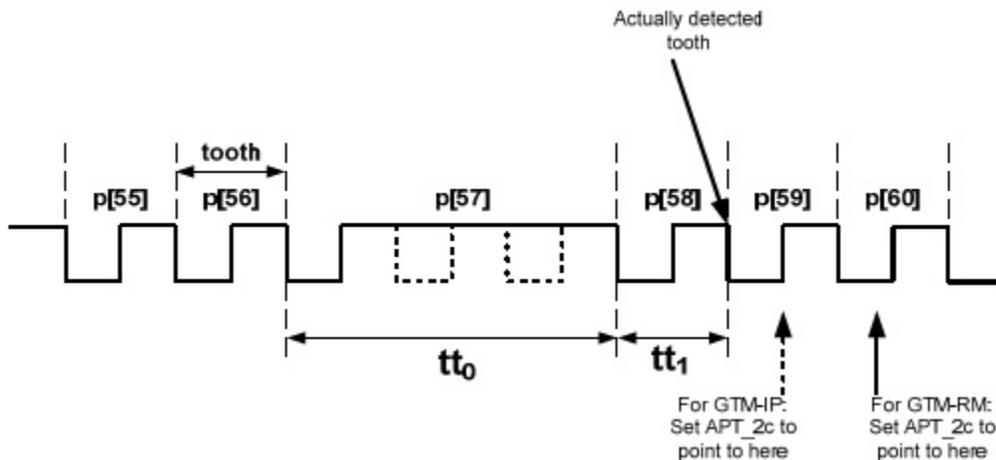

Figure 14. Tooth profile

After the gap is detected, the sub_inc1and sub_inc1c generation is enabled and the micro ticks are sent to the TBU channel 1 time base.

After the synchronization for the tooth wheel is 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.

An interrupt event inside of the tooth profile is defined after the second gap of the tooth wheel FULL_SCALE. A DPLL TRIGGER event 0 ISR handles the interrupt and determines if the lock1 bit is set.

10.2.3   Tooth signal creation on MPC57xx to run the example on the bench

To generate a 60-2 tooth signal for the TIM to read the PSM in Ring Buffer mode and ATOM0_CH7 is used in manner similar to the configuration in the PPA example.

The ATOM channel is configured to read data from PSM channel 10 in SOMP mode with the ARU enabled. The FIFO is set to run in Ring Buffer mode and transfer both words from the FIFO to the ARU. A normal tooth is defined as a 12,800 count PWM with 50% duty in the FIFO RAM and the missing tooth is defined as a PWM with three times the normal period and 1/3 (16.667%) of the normal duty.

10.3   Code

```
void init_tbu(void)
{
  // setup TBU channel 0
  GTM_TBU.CH0_CTRL.R = 0x5;  // no LOW_RES, choose CMU_CLK6
  // setup TBU channel 1
  GTM_TBU.CH1_CTRL.R = 0x1;  // Up/Down counter mode; sub_inc1c is chosen
  // setup TBU channel 2
  GTM_TBU.CH2_CTRL.R = 0x0;  // Free Running counter mode, CMU_CLK0

  // enable TBU channels 0, 1 and 2
  GTM_TBU.CHEN.R = 0x2A;
}
```

```c
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 = GTM_DPLL.RAM_INI.R;
  while(ram_ini_v) { // we have to wait until RAM initialization ends
    ram_ini_v = GTM_DPLL.RAM_INI.R;
  }

  p = &DPLL_RR2;
  p = p + (0x00001000/4);  // RAM region 2 offset size is 1024
  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;   // Trigger interrupt
  for (i=59; i<115; i++){    // file profile for regular tooth
    p[i] = 0x10000; // second HALF SCALE
  }
  p[115] = 0x30000;

  GTM_DPLL.IRQ_EN.R = 0x00040000;  // enable TINT0 interrupt

  // configure timeout value for actual TRIGGER slope
  p = &DPLL_RR1B;
  p[0x428 >> 2] = 0x780;    // configure DPLL control registers

  GTM_DPLL.CTRL_0.R = 0x403B0257;
  GTM_DPLL.CTRL_1.R = 0x80020000;  // configure TRIGGER input charact. first
  GTM_DPLL.CTRL_1.R = 0x80020012;  // now enable the DPLL
}

void microTick(void)
{

  int i = 0;

  //Set Up PSM Channel 0
  GTM_FIFO_0.CHANNEL[0].CTRL.R = 0x1; // PSM operates in Ring Buffer Mode
  GTM_F2A_0.CH_STR_CFG[0].R = 0X60000; // transfer both words to ARU
  for (i=0; i<57; i++) {
    GTM_AFD_0.CH[0].BUF_ACC.R = 3200;  // 57 times normal tooth
    GTM_AFD_0.CH[0].BUF_ACC.R = 1600;  // 50% duty cycle
  }
  GTM_AFD_0.CH[0].BUF_ACC.R  = 9600;  // 58 th tooth + 2 tooth gap
  GTM_AFD_0.CH[0].BUF_ACC.R  = 1600;

  GTM_F2A_0.ENABLE.R = 0x2;       // enable channel 0

  // configure ATOM0 channel 7 output
  GTM_ATOM_0.CH7_RDADDR.R  = 0x051;  // get data from PSM channel0
  GTM_ATOM_0.CH7_RDADDR.R   = 0xA;  // ATOM channel 7 in SOMP, ARU enabled

  GTM_ATOM_0.AGC_GLB_CTRL.R = 0x80000000; // enable update for shadow registers
  GTM_ATOM_0.AGC_OUTEN_STAT.R = 0x8000;    // enable channel 7 output
  GTM_ATOM_0.AGC_ENDIS_STAT.R = 0x8000;    // enable channel 7

  // configure TBU
  init_tbu();

  // configure TIM0 channel 0
  GTM_TIM_0.CH0_IRQ_EN.R  =   0x1;    // enable NEWVAL IRQ
  GTM_TIM_0.CH0_CTRL.R  =   0x00004005;   // configure and enable channel

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

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

```
void IRQ_GTM_TIM0_CH0 (void)
{
  // disable NOTIFY bit
  GTM_TIM_0.CH0_IRQ_NOTIFY.R = 0x1;
  // save actual time stamp
  actTS = GTM_TIM_0.CH0_GPR0.R;
  // tooth starts with falling edge
  if (!(actTS & 0x01000000)) { // falling edge detected
    // save old time stamp and tooth characteristic
    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)
          GTM_DPLL.APT_2C.R = (57+3)*4; // use first gap + 3 tooth
        else
          GTM_DPLL.APT_2C.R = (57+2)*4; // use first gap + 3 tooth
        // enable DPLL sub_inc1c generation
        GTM_DPLL.CTRL_1.R = 0x80020032;
        // disable TIM0 channel 0 NEWVAL interrupt
        GTM_TIM_0.CH0_IRQ_EN.R = 0x0;
      } // gap detected
    } // second valid edge
  } // falling edge
}

void IRQ_GTM_DPLL_CH18 (void)
{
  unsigned int actTS;

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

Appendix A GTM module definition and revision information

The GTM module contains a register that documents the exact revision of the GTM that is instantiated on a device. The format of the register is shown in the following table.

Table A-1. GTM Revision (GTM_REV)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset 0x00	R	DEV_CODE2				DEV_CODE1				DEV_CODE0				MAJOR			
	W																
	Reset	*				*[1]				*[1]				*[1]			
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	R	MINOR				NO				STEP							
	W																
	Reset	*[1]				*[1]				*[1]							

1. Reset value is implementation specific. See Table 3

Table A-2.   GTM_REV Field Descriptions

Field	Description
DEV_CODE2	Most significant byte of the Development Code
DEV_CODE1	Second byte of the Development Code
DEV_CODE1	Least significant byte of the Development Code
MAJOR	Major revision of the GTM
MINOR	Minor revision of the GTM
NO	NO is the delivery number (version)
STEP	GTM-IP step number

The following table shows the expected revision values for the different production devices in the MPC57xx family. The development code is a 3 digit (decimal encoded) identifier for the features supported by the GTM-IP. Please refer to the GTMINT chapter of the device reference manual for the correct values for the device and mask set that is being used.

Table A-3.   Device family definitions

Device	GTM Module	DEV_CODE[2:0]	MAJOR	MINOR	NO	STEP
MPC5746M	GTM103	103	1	5	5	A1
MPC5777M	GTM104	104	1	5	5	A1

A.1   Block diagram

The following diagram shows the submodules and how they are linked within the GTM. The number of each of the modules is variable from 0 to 12. Simple output channels are 16 bits wide whereas the input and complex output channels are 24 bits wide. The MCU's CPU can be run with a slow clock in low-end powertrain applications giving low power dissipation and low electronic emissions energy (EME). It supports the reduction of data traffic between CPU and GTM due to dedicated hardware FIFOs, programmable cores, DMA integration, and engine position hardware. The ARU central routing unit manages all internal data movement between submodules. For a complete list of the GTM variants' support on Freescale devices, see GTM module definition and revision information in Appendix A.

GTM
(Generic Timer Module)
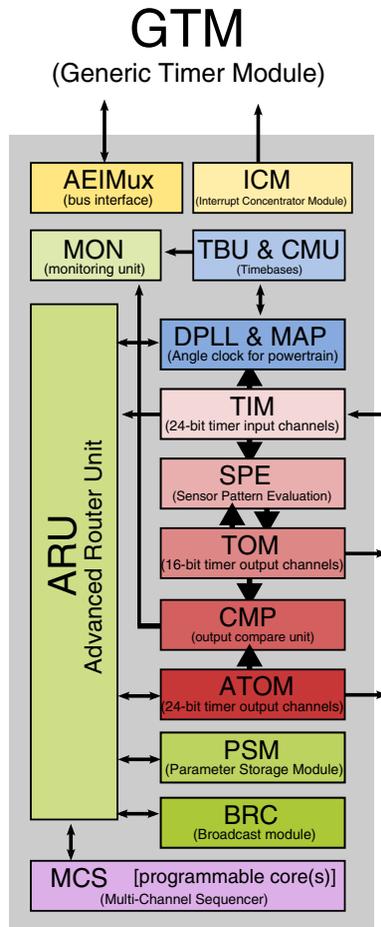
Figure A-1. Block diagram

The GTM module can be scaled up or down depending on the system requirements for timers. To date, there are 2 available configurations of the GTM.

- GTM103—mid sized GTM on MPC5746M
- GTM104—large GTM on MPC5777M

The GTM module integration is specific to the MCU. For example, in the case of the MPC5746M, the first device available in the family, the GTM logic operates on a double speed peripheral bus (Slow Crossbar) at up to 100 MHz while the timing is based on the device Peripheral Clock, up to 80MHz. All interrupts go to the interrupt controller with approximately 80% also connected to the DMA Controller. Interrupt clearing, resetting, and ECC of the local RAM memories are all specific to the MCU and not a part of the GTM module.

A.2   GTM submodule descriptions

The table below shows a basic summary of the different submodules of the GTM.

Table A-4.   GTM submodule descriptions

Submodule	Full name	Description
AEIMUX	AEI Interface	Generic bus interface for the GTM module. A bridge is required from the AEI to the MCU bus interface.

Table continues on the next page...

Table A-4. GTM submodule descriptions (continued)

Submodule	Full name	Description
AFD	AEI to FIFO Data Interface	Provides a data interface between the AEI bus and the FIFO submodule.
ARU	Advanced Routing Unit	Provides a mechanism for routing streams of data between data sources and transfer it to a destination. This is the heart of the GTM subsystem.
ATOM	ARU connected Timer Output Module	Capable of generating complex output signals through its interconnectivity with the ARU to other modules in the GTM subsystem.
BRC	Broadcast Module	Allows data streams to be duplicated and sent to multiple destinations.
CMP	Output Compare Module	Provides an XOR of duplicate outputs to provide an indication of differences for safety type applications.
CMU	Clock Management Unit	Generates all of the clocks and counters for the GTM subsystem. It contains a Configurable Clock Generation Unit (CFGU), a Fixed Clock Generation Unit (FXU), and a External Clock Generation Unit (EGU).
DPLL	Digital Phase Lock Loop	Provides the capability to multiply frequencies to provide a higher precision of position or value information. It performs calculations based on TRIGGER and STATE inputs from the MAP submodule to predict the duration of the current increment, generate pulses for up to two position counters, synchronise the actual position and predict position and time events without any CPU intervention. It can also seamlessly switch between modes under CPU control.
F2A	FIFO to ARU Interface	Provides the interface between the ARU and the FIFO.
FIFO	First in First Out Buffer	Provides a storage unit between the AFD and the ARU.
GTMDI	GTM Debug Interface	Provides an advanced, real-time development interface for the GTM, based on the IEEE-ISTO 5001-2011 Nexus standard. It provides both run control and trace capabilities.
GTMINT	GTM Integration Module	Provides a device specific wrapper around the GTM to handle specific MCU hardware interfaces including the module configuration control, AEI control, and interrupts.
ICM	Interrupt Concentrator Module	Gathers the GTM submodule interrupts into interrupt groups to provide a smaller number of interrupts to the host CPU of the microcontroller.
MAP	TIM0 Input Mapping Module	Generates two input signals (TRIGGER and STATE) for the DPLL submodule. The TIM can also be used as an input to the MAP submodule to provide additional filtering capabilities.
MCFG	Memory Configuration Module	Provides an infrastructure to organize physical memory blocks and maps them to the instances of the MCS submodules. This submodule is not normally shown on Block Diagrams as it is so closely tied to the MCS RAM.
MCS	Multi-Channel Sequencer	A generic data processing module that is connected to the ARU. It allows "programs" to be written to calculate complex output sequences that depend on Time Base values and ATOM signals. Other types of applications can also be handled by the MCS such as extending the operation of the TIM submodules, or using data from the host CPU to control GTM functions.

Table continues on the next page...

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

Table A-4. GTM submodule descriptions (continued)

Submodule	Full name	Description
MON	Monitoring Unit	Another submodule primarily for safety applications. It provides a mechanism to supervise common circuitry and resources by monitoring output channels using an MCS channel and a TIM to check for errors.
PSM	Parameter Storage module	Consists of the AEI-to-FIFO interface (AFD), the FIFO-to-ARU (F2A), and the FIFO itself.
SPE	Sensor Pattern Evaluation Module	Can be used to evaluate the three hall sensor inputs and together with the TOM to support driving a Brush-less DC motor.
TBU	Timer Base Unit	Provides a common time base that can be used throughout the GTM subsystem. The TBU is organized by channels. The number of channels is implementation specific.
TIM	Timer Input Module	Provides for filtering and capture of input signals. It allows several characteristics of the input to be measured, including the time stamping of rising and falling edges, as well as the number of edges since an enable.
TOM	Timer Output Module	Provides independent channels for generating simple Pulse Width Modulated signals.

A.3 GTM configurations

The table below shows the configurations of GTM modules that currently defined for use on different microcontrollers.

Table A-5. GTM configurations

GTM resource	GTM103	GTM104
MCS	4	6
TIM	4	6
TOM	3	5
ATOM	5	9
DPLL	1	1
DPLL RAM1a	96 × 24 bits	128 × 24 bits
DPLL RAM1b	384 × 24 bits	384 × 24 bits
DPLL RAM2	2048 × 24 bits	4096 × 24 bits
MCS RAM0	4KB	4KB
MCS RAM1	2KB	2KB
FIFO RAM	1024 × 29 bits	1024 × 29 bits
CMU	1	1
ARU	1 (120 slots)	1
BRC	1	1
PSM	1	2

Table continues on the next page...

Table A-5. GTM configurations (continued)

GTM resource	GTM103	GTM104
TBU	3 ch	3 ch
MAP	1	1
ICM	1	1
SPE	2	4
CMP	1	1
MON	1	1

Appendix B SIUL2 Configuration Examples

The SIUL2 supports mapping of module IP functions to physical pins. For example, if a user needs to support I2C communication in the application, the SIUL2 multiplexers must be programmed to connect I2C Serial Clock (SCL) and I2C Serial Data (SDA) to the appropriate pins of the device.

In order to configure the MPC57xx SIUL2 multiplexers, the user programs MSCR registers for input and output mapping. MSCR_IO registers define MCU function to pad output. MSCR_MUX registers define pad inputs to MCU function. The user executes the following three steps to configure pad input and output definition:
1. Configure pad (input buffer, output buffer, pull-up, open-drain, LVDS, etc.)
2. Configure the output signal source for the pad output (MSCR_IO registers)
3. Configure the input signal source for each pad input (MSCR_MUX registers)

The figure below illustrates the pad configuration options:

Figure B-1. Generic structure for I/O pad configuration
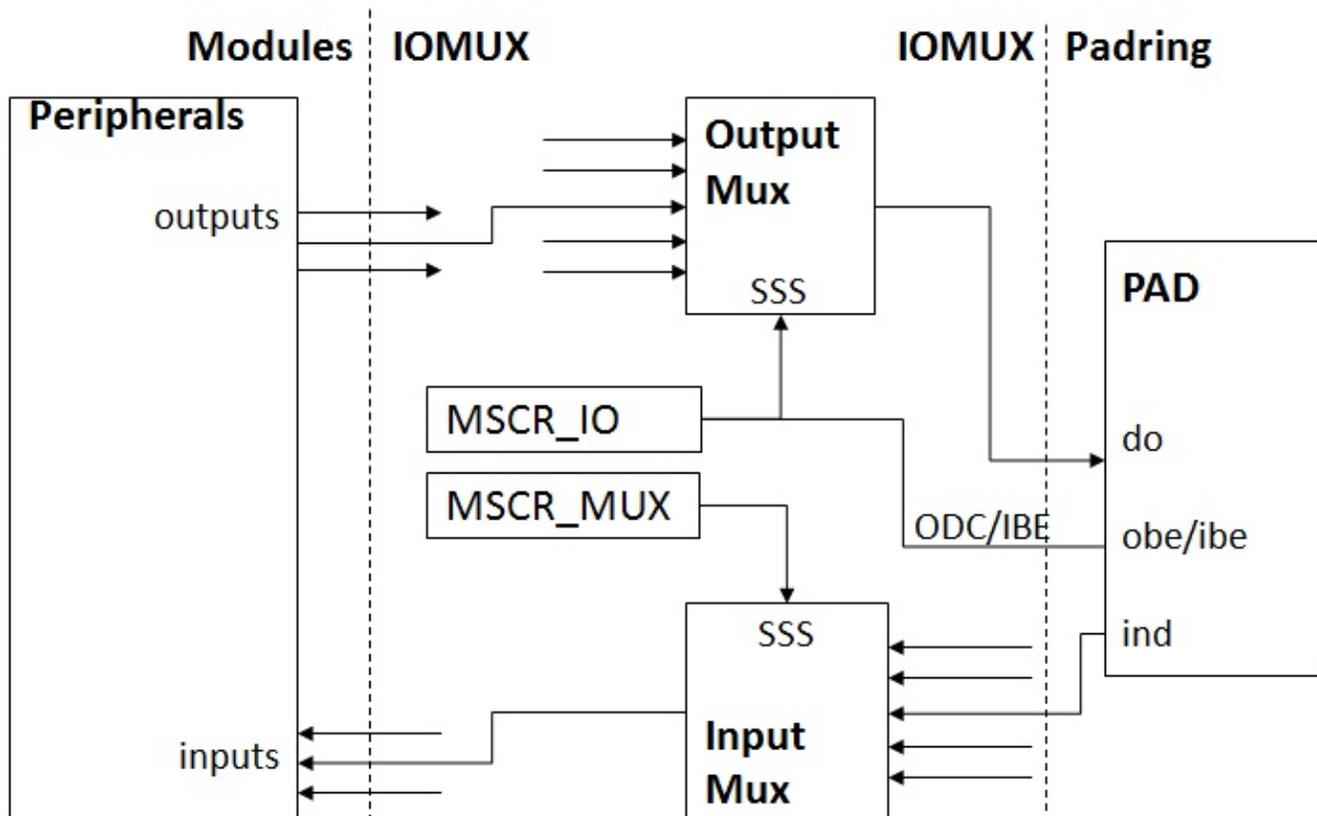
B.1   GTM I/O function to MPC57xx port mapping

The following three tables can be used to find the GTM input or output pin on the various MPC57xx ports.

Table B-1.   GTM TIM to MPC57xx port

Submodule	Channel	GTM103	GTM104	Port	MCSR	Alt MCSR for IBE	SSS value
TIM0	CH0	X	X	PF1	512	81	03
	CH1	X	X	PG0	513	96	02
	CH2	X	X	PH6	514	118	05
	CH3	X	X	PI9	515	137	02
	CH4	X	X	PI13	516	141	02
	CH5	X	X	PI12	517	140	02
	CH6	X	X	PI11	518	139	02
	CH7	X	X	PI10	519	138	02
TIM1	CH0	X	X	PH10	520	122	08
	CH1	X	X	PF0	521	80	03
	CH2	X	X	PH1	522	113	07
	CH3	X	X	PD10	523	58	03
	CH4	X	X	PJ1	524	145	04
	CH5	X	X	PJ2	525	146	04
	CH6	X	X	PH14	526	126	07
	CH7	X	X	PJ4	527	116	03
TIM2	CH0	X	X	PM5	528	197	07
	CH1	X	X	PL9	529	185	07
	CH2	X	X	PL10	530	185	07
	CH3	X	X	PL11	531	185	09
	CH4	X	X	PL12	532	185	08
	CH5	X	X	PL13	533	185	09
	CH6	X	X	PM3	534	195	07
	CH7	X	X	PM0	535	192	08
TIM3	CH0	X	X	PM3	537	199	06
	CH1	X	X	PM7	537	199	06
	CH2	X	X	PJ13	538	157	06
	CH3	X	X	PJ12	539	156	07
	CH4	X	X	PJ15	540	159	07
	CH5	X	X	PJ14	541	158	07
	CH6	X	X	PL14	542	190	08
	CH7	X	X	PL15	543	191	08
TIM4[1]	CH0		X	PM8	544	200	0A

Table continues on the next page...

Table B-1. GTM TIM to MPC57xx port (continued)

Submodule	Channel	GTM103	GTM104	Port	MCSR	Alt MCSR for IBE	SSS value
	CH1		X	PM7	545	199	0A
	CH2		X	PL1	546	177	08
	CH3		X	PL0	547	176	08
	CH4		X	PK15	548	175	08
	CH5		X	PL4	549	180	08
	CH6		X	PL3	550	179	08
	CH7		X	PM0	551	192	09
TIM5[1]	CH0		X	PX1	552	322	07
	CH1		X	PL15	553	191	09
	CH2		X	PX2	554	240	05
	CH3		X	PQ8	555	232	06
	CH4		X	PQ9	556	233	06
	CH5		X	PL6	557	182	08
	CH6		X	PQ7	558	231	07
	CH7		X	PL7	559	183	07

1. TIM4 and TIM5 cannot be made available on the MPC5777M device at the same time as all the other submodules due to pin count restrictions.

NOTE

Not all outputs are available on all package options.

Table B-2. GTM TOM to MPC57xx port

Submodule	Channel	GTM103	GTM104	Port	MCSR	SSS
TOM0	CH0	X	X	PB9	25	08
	CH1	X	X	PB10	26	08
	CH2	X	X	PB8	24	08
	CH3	X	X	PA10	10	08
	CH4	X	X	PA8	8	08
	CH5	X	X	PA9	9	08
	CH6	X	X	PD2	50	08
	CH7	X	X	PD1	49	08
	CH8	X	X	PC2	34	08
	CH9	X	X	PC0	32	08
	CH10	X	X	PC9	41	08
	CH11	X	X	PC8	40	08
	CH12	X	X	PC7	39	08
	CH13	X	X	PC6	38	08
	CH14	X	X	PC5	37	08
	CH15	X	X	PC4	36	08

Table continues on the next page...

Table B-2. GTM TOM to MPC57xx port (continued)

Submodule	Channel	GTM103	GTM104	Port	MCSR	SSS
TOM1	CH0	X	X	PC3	35	09
	CH1	X	X	PC1	33	09
	CH2	X	X	PD9	57	09
	CH3	X	X	PA2	2	08
	CH4	X	X	PA1	1	08
	CH5	X	X	PA12	12	08
	CH6	X	X	PA13	13	08
	CH7	X	X	PA0	0	08
	CH8	X	X	PD14	62	09
	CH9	X	X	PD15	63	09
	CH10	X	X	PL14	190	08
	CH11	X	X	PH11	123	08
	CH12	X	X	PA3	3	08
	CH13	X	X	PD8	56	08
	CH14	X	X	PE7	71	08
	CH15	X	X	PD4	52	08
TOM2	CH0	X	X	PF13	93	09
	CH1	X	X	PM4	196	08
	CH2	X	X	PC15	47	09
	CH3	X	X	PF11	91	09
	CH4	X	X	PC13	45	09
	CH5	X	X	PC12	44	09
	CH6	X	X	PC11	43	09
	CH7	X	X	PC10	42	09
	CH8	X	X	PD5	53	09
	CH9	X	X	PE8	72	09
	CH10	X	X	PE9	73	09
	CH11	X	X	PF2	82	09
	CH12	X	X	PH15	127	09
	CH13	X	X	PE6	70	09
	CH14	X	X	PF6	86	09
	CH15	X	X	PF7	87	09
TOM3[1]	CH0		X	PN5	213	08
	CH1		X	PN7	215	08
	CH2		X	PN15	223	08
	CH3		X	PL5	181	09
	CH4		X	PL6	182	09
	CH5		X	PN6	214	08
	CH6		X	PN9	217	08

Table continues on the next page...

Table B-2. GTM TOM to MPC57xx port (continued)

Submodule	Channel	GTM103	GTM104	Port	MCSR	SSS
	CH7		X	PN13	221	08
	CH8		X	PN8	216	08
	CH9		X	PN10	218	08
	CH10		X	PN14	222	08
	CH11		X	PQ0	224	08
	CH12		X	PQ1	225	08
	CH13		X	PQ2	226	08
	CH14		X	PN2	210	08
	CH15		X	PN4	212	08
TOM4[1]	CH0		X	PN5	213	09
	CH1		X	PN7	215	09
	CH2		X	PN15	223	09
	CH3		X	PN11	219	09
	CH4		X	PN12	220	09
	CH5		X	PN6	214	09
	CH6		X	PN9	217	09
	CH7		X	PL2	178	09
	CH8		X	PQ3	227	09
	CH9		X	PQ6	230	09
	CH10		X	PQ5	229	09
	CH11		X	PQ4	228	09
	CH12		X	PQ1	225	09
	CH13		X	PQ7	231	09
	CH14		X	PN1	209	09
	CH15		X	PN3	211	09

1. TOM3 and TOM4 cannot be made available on the MPC5777M device at the same time as all the other submodules due to pin count restrictions.

NOTE

Not all outputs are available on all package options.

NOTE

TOM0_CH4 and TOM0_CH5 on ports A8 and A9 are muxed with the JTAG TDI and TDO functions and are not visible on the MPC57xx motherboard port headers. They can only be observed on the daughtercard.

Table B-3. GTM ATOM to MPC57xx port

Submodule	Channel	GTM103	GTM104	Port	MCSR	SSS value
ATOM0	CH0	X	X	PF10	90	0A
	CH1	X	X	PD7	55	0A
	CH2	X	X	PD6	54	0A

Table continues on the next page...

MPC57xxM Generic Timer Module (GTM) Quick Start Guide, Rev 2, 4/2014

Table B-3.   GTM ATOM to MPC57xx port (continued)

Submodule	Channel	GTM103	GTM104	Port	MCSR	SSS value
	CH3	X	X	PA14	14	0A
	CH4	X	X	PA11	11	0A
	CH5	X	X	PA15	15	0A
	CH6	X	X	PD13	61	0A
	CH7	X	X	PE3	67	0A
ATOM1	CH0	X	X	PG15	111	0A
	CH1	X	X	PF4	84	0B
	CH2	X	X	PE0	64	0B
	CH3	X	X	PE1	65	0B
	CH4	X	X	PE2	66	0B
	CH5	X	X	PD12	60	0B
	CH6	X	X	PF8	88	0B
	CH7	X	X	PH5	117	0B
ATOM2	CH0	X	X	PD0	48	0A
	CH1	X	X	PJ3	147	0A
	CH2	X	X	PH9	121	0A
	CH3	X	X	PH8	120	0A
	CH4	X	X	PH7	119	0A
	CH5	X	X	PI15	143	0A
	CH6	X	X	PJ8	152	08
	CH7	X	X	PJ6	150	08
ATOM3	CH0	X	X	PE4	68	0B
	CH1	X	X	PE12	76	0B
	CH2	X	X	PI8	136	0B
	CH3	X	X	PC14	46	0B
	CH4	X	X	PB11	27	0B
	CH5	X	X	PD3	51	0B
	CH6	X	X	PG13	109	0B
	CH7	X	X	PG14	110	0B
ATOM4	CH0	X	X	PH3	115	0B
	CH1	X	X	PE11	75	0B
	CH2	X	X	PE10	74	0B
	CH3	X	X	PH2	114	0B
	CH4	X	X	PF12	92	0B
	CH5	X	X	PH12	124	0B
	CH6	X	X	PF14	94	0B
	CH7	X	X	PE5	69	0B
ATOM5[1]	CH0		X	PQ5	239	0A
	CH1		X	PM11	203	0A

Table continues on the next page...

Submodule	Channel	GTM103	GTM104	Port	MCSR	SSS value
	CH2		X	PM12	204	0A
	CH3		X	PN11	219	0A
	CH4		X	PN12	220	0A
	CH5		X	PM1	193	09
	CH6		X	PX0	321	0A
	CH7		X	PQ13	237	0A
ATOM6[1]	CH0		X	PJ7	151	09
	CH1		X	PQ14	238	0A
	CH2		X	PL14	190	09
	CH3		X	PQ5	229	0A
	CH4		X	PQ10	228	0A
	CH5		X	PL10	234	0A
	CH6		X	PN13	186	09
	CH7		X	PL12	221	0A
ATOM7[1]	CH0		X	PL12	188	09
	CH1		X	PM11	203	0B
	CH2		X	PM12	204	0B
	CH3		X	PM14	206	0B
	CH4		X	PM15	207	0B
	CH5		X	PN0	208	0B
	CH6		X	PN1	209	0A
	CH7		X	PN3	211	0A
ATOM8[1]	CH0		X	PN8	216	0B
	CH1		X	PN10	218	0B
	CH2		X	PN14	222	0B
	CH3		X	PQ0	224	0B
	CH4		X	PQ1	225	0B
	CH5		X	PQ2	226	0B
	CH6		X	PN2	210	0B
	CH7		X	PN4	212	0B

1. ATOM5, ATOM6, ATOM6 and ATOM8 cannot be made available on the MPC5777M device at the same time as all the other submodules due to pin count restrictions.

NOTE

Not all outputs are available on all package options.

NOTE

ATOM0_CH1, ATOM0_CH2, and ATOM0_CH3 on Ports A14, D6, and D7 are muxed with the SIPI RXP, TXN, and TXP functions and are not visible on the MPC57xx motherboard port headers. They can only be observed on the daughtercard.

Appendix C Include the MCS ASM Binary in a Greenhills MULTI Project

The HighTec MCS assembler can create a raw binary file of the machine code for the written GTM MCSx assembly code. This can be included in to the chip's I/O processor project through build and link options.

Create the sections .raw and .mymcsrawdata in the I/O processor linker file. For example:

```
.raw             : {} > .              /* GTM MCS binaries     */
.mcs0rawdata : {mcs0_qom.bin(.raw)} > .   /* MCS0 binaries */
```

By creating a label for this section in the linker, the I/O processor can find the raw data to be copied. For example:

```
__MCS0_ADDR = ADDR(.mcs0rawdata);
```

This section will be excluded from the build by default. This can be changed by modifying the build options for the I/O processor project as shown below.
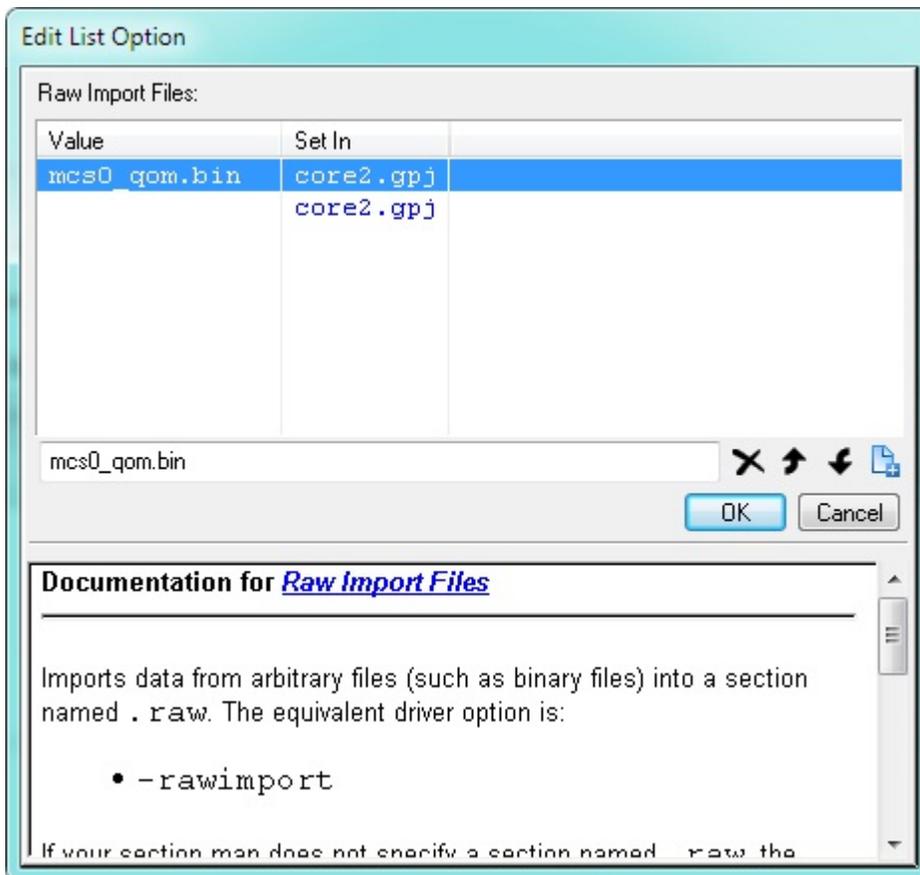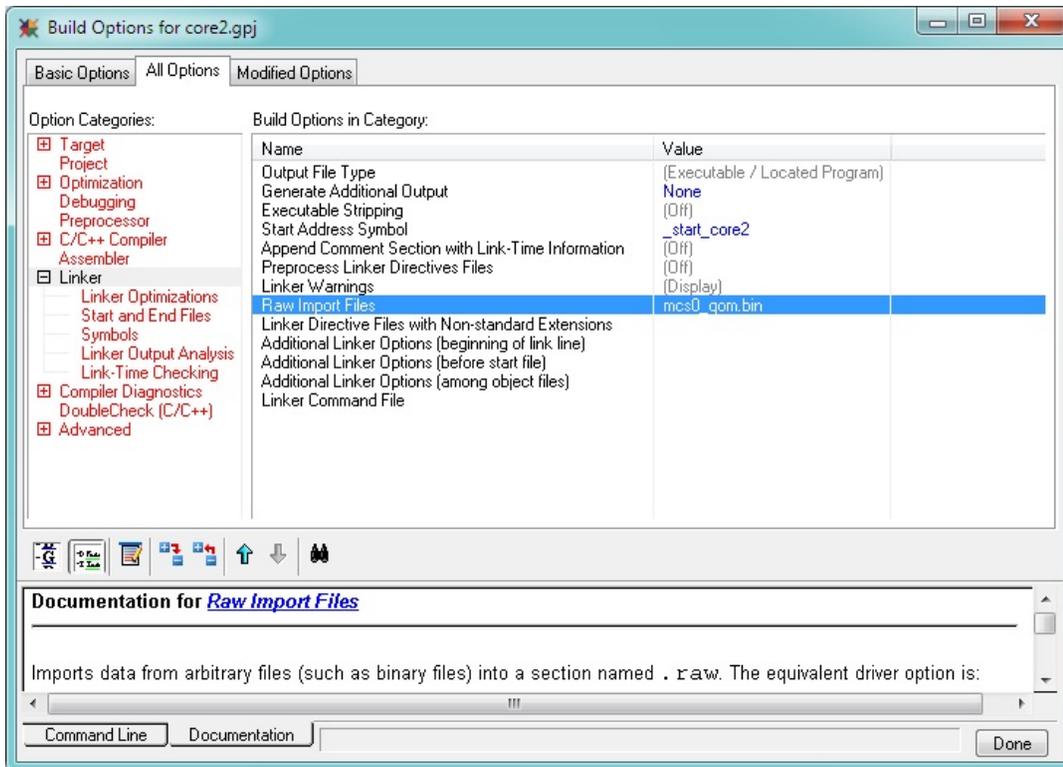
Appendix D GTM References

For more information on the GTM, see the additional documents listed in the following table.

Document	Title	Availability	Specific references
MPC5746MRM/ MPC5777MRM	MPC5746M/MPC5777M Reference Manual	www.freescale.com	"GTM103 Integration Module" chapter
			"GTM Development Interface" chapter
			"GTM subsystem configuration" sub-sections
GTM103RM/GTM104RM	GTM Reference Manual	www.freescale.com	—
GTM103/104 Appendix B	GTM Specification Appendix B	www.freescale.com	—

Appendix E Revision History

Major Revision	Minor Revision	Description	Date
1	n/a	Initial public release	8/2013
2	n/a	Updated code snippets to match the latest header file release format with GTM submodules included in the main chip header file, and updated SIUL2 section to match latest revision of devices and reference manuals	4/2014