Application Note

AN2363/D
Rev. 0, 10/2002

Using the Frequency Measurement TPU Function (FQM) with the MPC500 Family

Randy Dees
TECD Applications

This TPU Programming Note is intended to provide simple C interface routines to the frequency measurement TPU function (FQM). [1] The routines are targeted for the MPC500 family of devices, but they should be easy to use with any device that has a TPU.

1 Functional Overview

The FQM function counts the number of pulses that are presented to a channel pin within a user specified time window. The pulse count is available to the user as a 16-bit number. Either rising or falling edges can be used as the beginning of a pulse. There are two basic modes of operation. In single shot mode, pulses are accumulated for a single window time. In continuous mode, pulses are automatically accumulated in repetitive windows. The FQM function is built into both ROM banks of the TPU3 modules on the MPC500 devices.

2 Description

The time window is specified as a number of TCR clock ticks. The maximum value is 0x8000. Either TCR may be used as the time base. The function is initialized by specifying the time window and issuing a host service request. The channel is then initialized by the microcode and waits for the first selected edge. The time window begins with the first selected edge following initialization. Thus the function can be pre-initialized to wait for pulses to appear on the pin.

The term “selected edge” is used to describe an edge, either rising or falling, that has been specified as the beginning of a pulse. Since the first selected edge to appear after initialization starts the first window, the second selected edge is counted as the end of the first pulse. Thus pulses are counted as they complete. The number of completed pulses is accumulated until the time window expires. At that time the accumulated number is written to parameter RAM and an interrupt service request is generated. Partial pulses are not counted. If pulse completion coincides with the end of window time, it is counted as a complete pulse.

In single-shot mode, the function idles when a pulse is complete. New single-shot accumulation times can be initiated by issuing a host service request. Window accumulation time can also be changed prior to issuing the request.

---

[1] The information in this Programming Note is based on TPUPN03. It is intended to compliment the information found in that Programming Note.

In continuous mode, a new time accumulation window begins coincident with the end of the previous window, and pulse counting restarts at zero. If a pulse begins in a window and continues into the next window, it is counted as the first pulse in the new window when the first selected edge in the new window is detected. Thus, no pulse is lost, even when it straddles accumulation windows. At the end of each time accumulation period, the newly-accumulated value is written to parameter RAM and an interrupt service request is generated. The accumulated value is available to the CPU until the end of the current window. When the current window ends, the accumulated value is overwritten. CPU interrupt latency must be less than window time in order to guarantee that no accumulation values are missed. In continuous mode, the user can modify the length of the time accumulation window at any time by changing the window value in parameter RAM. The new window length takes effect at the end of the current window time. It is not necessary to disable the function or issue a host service request in order to make the change.

The FQM function has been optimized for fast execution and small size. The function does not attempt to distinguish false edges due to noisy inputs. Any pulse long enough to pass through the pin synchronizer and digital filter is counted. The pin synchronizer and digital filter reject all pulses narrower than two CPU system clocks and pass all pulses wider than four CPU system clocks. A companion function to FQM, called pulse accumulate in a programmable window (PAPW) offers noise rejection of longer pulses at the expense of slightly longer execution time and increased code size. PAPW can be used in the same manner as FQM.

3 FQM C Level API

- Initialization Functions
  - void tpu_fqm_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, UINT8 mode, UINT8 edge, UINT8 timer, UINT16 wind_sz)
- Change Operation Function
  - void tpu_fqm_update_window_size(struct TPU3_tag *tpu, UINT8 channel, UINT16 wind_sz)
- Value Return Functions
  - UINT16 tpu_fqm_get_pulse(struct TPU3_tag *tpu, UINT8 channel)
- General TPU Functions (defined in mpc500_util.h):
  - void tpu_enable(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority)
  - void tpu_disable(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority)
  - void tpu_interrupt_enable(struct TPU3_tag *tpu, UINT8 channel)
  - void tpu_interrupt_disable(struct TPU3_tag *tpu, UINT8 channel)
  - void tpu_clear_interrupt(struct TPU3_tag *tpu, UINT8 channel)
  - UINT8 tpu_check_interrupt(struct TPU3_tag *tpu, UINT8 channel)

3.1 Initialization Functions (tpu_fqm_init)

This function initializes the FQM function and sets the initialization mode. To change the operating mode, this function can be called again. This function has the following parameters:

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.
- priority - This is the priority to assign to the FQM function. The TPU priority definitions are defined in mpc500_utils.h. See Table 1 for values that are defined for the channel priority.

Table 1. TPU Priorities

TPU Priorities	Definition
TPU_PRIORITY_DISABLE	0b00
TPU_PRIORITY_LOW	0b01
TPU_PRIORITY_MEDIUM	0b10
TPU_PRIORITY_HIGH	0b11

- mode - This parameter sets the operating mode of the TPU3 FQM function either single-shot mode or continuous mode.
- edge - This parameter sets whether the rising or falling edge should be used in measuring the incoming frequency. It is used both in the Host Sequence bits and for the PAC definition in the Channel_Control.
- timer - This parameter sets whether the TPU3 TCR1 or TCR2 clock should be used for measuring the incoming frequency.
- wind_sz - This parameter sets the initial value for the window size for measuring the incoming frequency.The window should be less than or equal to 0x8000.

3.2 Change Operation Function

tpu_fqm_update_window_size

This functions allows the window size to be modified.

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.
- wind_sz - This parameter sets the new size of the window used to measure the input frequency.

3.3 Value Return Function

tpu_fqm_get_pulse

This function returns the number of TCR ticks that were measured for the incoming frequency.

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.

3.4 General TPU Functions

The following routines are generic and are useful for all TPU functions.

void tpu_enable(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority)

This function enables the TPU channel and can be used to change the channel priority.

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.

- priority - This is the new channel priority.

void tpu_disable(struct TPU3_tag *tpu, UINT8 channel)

This function disables the TPU channel. It sets the priority to 0 to disable the channel.

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.
- priority - This is the new channel priority.

void tpu_interrupt_enable(struct TPU3_tag *tpu, UINT8 channel)

This function enables the interrupt bit for the specified channel.

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.

void tpu_interrupt_disable(struct TPU3_tag *tpu, UINT8 channel)

This function disables the interrupt bit for the specified channel.

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.

void tpu_clear_interrupt(struct TPU3_tag *tpu, UINT8 channel)

This function clears the interrupt bit for the specified channel.

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.

UINT8 tpu_check_interrupt(struct TPU3_tag *tpu, UINT8 channel)

This function checks the interrupt bit for the specified channel to see if it is set. This function returns TRUE if this channel caused the interrupt, FALSE otherwise.

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the FQM function assigned to it.

4 Configuration of FQM Function

The CPU configures the FQM function as follows:

1. The appropriate channel priority bits are cleared, disabling the channel.
2. The FQM function number is written to the channel function select bits.
3. CHANNEL_CONTROL and WINDOW_SIZE are written to channel parameter RAM.
4. The host sequence bits are written, selecting the desired action edge and mode of operation.
5. An HSR is issued to initialize the function.
6. The channel priority bits are written to enable the function and assign channel priority.
7. The TPU executes the initialization state.

All of these steps are included in the C level *tpu_fqm_init*() function. See Section 3.1, “Initialization Functions (tpu_fqm_init).”

After initialization, the TPU waits for the first selected edge to begin the time window. At the expiration of the time window, the accumulated value is written to the PULSE_COUNT parameter and an interrupt service request is made. This can be read with the C level function tpu_fqm_get_pulse. See Section 3.3, “Value Return Function.” In single-shot mode, the function then goes to an idle state. In continuous mode, the function immediately begins pulse accumulation in a new window. In continuous mode an interrupt service request is made at the completion of every window time. The tpu_fqm_get_pulse function can be called after an interrupt has occurred.

Once single-shot mode has completed, another single-shot sequence can be scheduled by issuing an initialization HSR. Mode and window time parameters can be modified before writing to the HSR register. If an initialization HSR is made prior to the expiration of a current time window, that accumulation is aborted and a new accumulation begins.

During continuous mode operation, the window time parameter can be modified without re-initialization. The new time period takes effect as soon as the current window time expires. The C level function tpu_fqm_update_window_size can be called to updated the window size. See Section 3.2, “Change Operation Function.” If an initialization HSR is made prior to the expiration of a current time window, the current accumulation is aborted and a new accumulation begins. To change mode during continuous mode operation, first disable the channel, then write the appropriate parameter registers and host sequence bits, issue an HSR, and then write the priority bits. This procedure prevents indeterminate results due to modification of sequence bits while the function is running.

5 Example Code

The following code shows an example program that initializes the TPU FQM ROM function using the C level API (see Section 5.1, “Code Listing”). The example shows the FQM function on channel 0. It then waits until it has made a measurement and then reads the result. The program then goes into an endless loop. Also shown (see Section 12.1, “MIOS Initialization Script for WindRiver SingleStep') is a debugger script that unitizes the MIOS PWM channel 0 to the frequency of approximately 4882 hertz. This can be connected to TPU A channel 0 for to make a measurement. Also included is a SingleStep script for reading the channel 0 parameter RAM (see TPU Read Channel 0 Parameter RAM Script).

This example code was written and tested for the WindRiver DiabData C Compiler version 4.3g, but should be potable to other compiler environments.

5.1 Code Listing

```
/*********************************************************************/
/* FILE NAME: tpu_tsm_ex1.c.............COPYRIGHT (c)  2002 */
/*********************************************************************/


#include "mpc555.h" /*Define all of the MPC555 registers, */
                        /* Change for other MPC500 devices. */
#include "mpc500.c"     /* Configuration routines for MPC555 EVB, */
                        /* change if other hardware is used. */
#include "mpc500_util.h"    /* Utility routines */
```

Code Listing

```
#include "tpu_fqm.h"    /* TPU FQM functions */

     struct TPU3_tag *tpua = &TPU_A;   /* pointer for TPU routines */

void main ()
{
     int x;  /* Just an integer to hold a value  */
     UINT8 chan=0; /* set to  channel 0 */
     UINT16 freq;


/* Hardware Setup -- machine settings (watchdog, timers, speed, etc.) */
   setup_mpc500(40);       /*Setup device and program PLL to 40MHz*/
   setup_tpu(tpua); /* Do general TPU set up. */

   tpu_fqm_init(tpua, chan, TPU_PRIORITY_HIGH, TPU_FQM_FALL_EDGE_CONT,
                TPU_FQM_FALL, TPU_FQM_TCR1,0x8000);

   tpu_ready(tpua, chan);
   freq = tpu_fqm_get_pulse(tpua, chan);

   while(1){
      /* Hold at end of program */
      x=4;

     };
}    /* End of main */


void setup_tpu(struct TPU3_tag *tpu)
{
     tpu->TPUMCR.R = 0x0020; /* divide by 1, supervisor and user access. */
tpu->TPUMCR3.R = 0x0040; /* enable enhanced prescaler - divide by 2 */
     tpu->TICR.B.CIRL = 5; /* set interrupt level to 5.... */
     tpu->TICR.B.ILBS = 0;
}
```

5.2 Example Description

This example sets the Enhanced prescaler to divide by 2 and it sets the TCR1 prescaler to 2. This gives a TCR1 clock frequency of 10 MHz (divide by 4 of the 40 MHz system clock). This yields a resolution of 100ns. The sample window is open for 0x8000 (32768) TCR1 clock periods which gives a window time of 3.2768 mS. This allows for a minimum frequency measurement of approximately 305 Hz.

$$\text{Frequency} = \frac{\text{counts}}{\text{TCR1period}(x)\text{windowsize}}$$

Figure 1. Frequency Formula

The MIOS example PWM shown in Section 12.1, “MIOS Initialization Script for WindRiver SingleStep” generates a frequency of 4882.8 Hz. The FQM gives a measurement of 0x10 (16).

$$\text{Frequency} = \frac{16}{100\text{ns}(x)32768}$$

$$\text{Frequency} = \frac{16}{3.27\text{mS}}$$

$$\text{Frequency} = 4882.8\text{Hz}$$

Figure 2. Frequency Calculation

6 Header File Definitions

All of the previous sections of code are found in the mpc500_util.c file that is contained in the Freescale MPC500 header files. The following definitions for the code prototypes, TPU parameter RAM structures, and common TPU definitions, including the function ID numbers of the internal TPU3 ROM functions, are found in the file mpc500_util.h.

```
#ifndef _MPC5xx_UTIL_H
#define _MPC5xx_UTIL_H

#include "m_common.h"
#include "m_tpu3.h"

void  tpu_func(struct TPU3_tag *tpu, UINT8 channel, UINT8 function_number);
UINT8 tpu_get_func(struct TPU3_tag *tpu, UINT8 channel);
void  tpu_hsr(struct TPU3_tag *tpu, UINT8 channel, UINT8 hsr);
UINT8 tpu_get_hsr(struct TPU3_tag *tpu, UINT8 channel);
void  tpu_hsq(struct TPU3_tag *tpu, UINT8 channel, UINT8 hsq);
UINT8 tpu_get_hsq(struct TPU3_tag *tpu, UINT8 channel);
```

```
void  tpu_enable(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority);
void  tpu_disable(struct TPU3_tag *tpu, UINT8 channel);
void  tpu_interrupt_enable(struct TPU3_tag *tpu, UINT8 channel);
void  tpu_interrupt_disable(struct TPU3_tag *tpu, UINT8 channel);
void  tpu_clear_interrupt(struct TPU3_tag *tpu, UINT8 channel);
UINT8 tpu_check_interrupt(struct TPU3_tag *tpu, UINT8 channel);

#define tpu_ready(tpu, channel) while(tpu_get_hsr(tpu, channel)!=0)


/*********************************
 *     TPU3                      *
 *                               *
 *********************************/
/* Define data structure for one TPU channel. This is useful */
/* to allow indexing along the channels. */
struct TPU_param_tag {
    VUINT16 param0;
    VUINT16 param1;
    VUINT16 param2;
    VUINT16 param3;
    VUINT16 param4;
    VUINT16 param5;
    VUINT16 param6;
    VUINT16 param7;
};

struct TPU_param32_tag {
    VUINT32 param0;
    VUINT32 param2;
    VUINT32 param4;
    VUINT32 param6;
};

/* Define TPU Function numbers for standard TPU mask */
/* TPU BANK 0 */
#define TPU_FUNCTION_PTA 0xF
#define TPU_FUNCTION_QOM 0xE
#define TPU_FUNCTION_TSM 0xD
#define TPU_FUNCTION_FQM 0xC
#define TPU_FUNCTION_UART 0xB
```

```
#define TPU_FUNCTION_NITC 0xA
#define TPU_FUNCTION_COMM 0x9
#define TPU_FUNCTION_HALLD 0x8
#define TPU_FUNCTION_MCPWM 0x7
#define TPU_FUNCTION_FQD 0x6
#define TPU_FUNCTION_PPWA 0x5
#define TPU_FUNCTION_OC 0x4
#define TPU_FUNCTION_PWM 0x3
#define TPU_FUNCTION_DIO 0x2
#define TPU_FUNCTION_SPWM 0x1
#define TPU_FUNCTION_SIOP 0x0

/* TPU BANK 1 */
/* Only 2 functions are different in bank 1 */
#define TPU_FUNCTION_ID 0x5
#define TPU_FUNCTION_RWTPIN 0x1

/* TPU Scheduler Priorities */
#define TPU_PRIORITY_HIGH   3
#define TPU_PRIORITY_MIDDLE   2
#define TPU_PRIORITY_LOW   1
#define TPU_PRIORITY_DISABLE   0

/* TPU General */
#define TPU_CHANNEL_MASK 0xF
#define TPU_PRIORITY_MASK 0x3
#define TPU_HSR_MASK 0x3
#define TPU_HSQ_MASK 0x3

#endif /*ifdef _MPC5xx_UTIL_H */
```

7 FQM C Level API Code

7.1 FQM Initialization Function

The initialization routine initializes the channel to run the FQM function. It should be called a second time to change the operating conditions (rise to falling edge or single shot to continuous mode).

```
/*****************************************************************************
FUNCTION      : tpu_fqm_init
*****************************************************************/
```

FQM Initialization Function

```
void tpu_fqm_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority,
                                        UINT8 mode, UINT8 edge, UINT8 timer, UINT16 wind_sz)
{
UINT16 channel_control;
UINT16 pac;
UINT16 tbs;
UINT8 hsq;

    /* disable channels so they can be configured safely */
    tpu_disable( tpu, channel);

    /*  FQM is function 0xC                 */
    tpu_func( tpu, channel, TPU_FUNCTION_FQM);

    /* disable interrupts on channels so they can be configured safely */
    tpu_interrupt_disable( tpu, channel );

   /* mask off illegal values */
    mode = ( mode & TPU_FQM_MODE_MASK );
          edge = edge & TPU_FQM_PAC_MASK;
    pac = edge << 2;
    tbs = ( timer & TPU_FQM_TBS_MASK ) << 6;

    /* Initialize Parameter RAM                                        */
    channel_control = ( tbs | pac | TPU_FQM_PSC );

    tpu->PARM.R[channel][TPU_FQM_CHANEL_CONTROL] = channel_control;
    tpu->PARM.R[channel][TPU_FQM_WINDOW_SIZE] = wind_sz;



/***********************************************/
/*    Configure the Channels.                  */
/***********************************************/

   if ((edge == TPU_FQM_FALL ) && (mode == TPU_FQM_SINGLE)) {
                  hsq = TPU_FQM_FALL_EDGE_SING;
                  }
                  else if ((edge == TPU_FQM_FALL) && (mode == TPU_FQM_CONT)){
                            hsq = TPU_FQM_FALL_EDGE_CONT;
                            }
                            else if ((edge == TPU_FQM_FALL) && (mode == TPU_FQM_SINGLE)){
```

```
                                        hsq = TPU_FQM_RISE_EDGE_SING;
                                        }
                                        else hsq = TPU_FQM_RISE_EDGE_CONT;
    tpu_hsq(tpu, channel, hsq);
    tpu_hsr(tpu, channel, TPU_FQM_INIT);


     /* Enable channel by assigning a priority to them. */
     tpu_enable(tpu, channel, priority);

}  /* End tpu_fqm_init */
```

7.2 FQM Change Window Size Function

The tpu_fqm_update_window routine should be called to change the size of the sample window.

```
/*********************************************************************
FUNCTION      : tpu_fqm_update_window
*********************************************************************/

void      tpu_fqm_update_window_size(struct TPU3_tag *tpu,
                                              UINT8 channel, UINT16 wind_sz)
{
    tpu->PARM.R[channel][TPU_FQM_WINDOW_SIZE] = wind_sz;
}  /* End tpu_fqm_update_window */
```

7.3 FQM Get Value Return Function

The function *tpu_fqm_get_pulse* returns the number of TCR clocks of the frequency on the input channel.

```
/*********************************************************************
FUNCTION      : tpu_fqm_get_pulse
*********************************************************************/
UINT16    tpu_fqm_get_pulse(struct TPU3_tag *tpu, UINT8 channel)
{
          UINT16 pulse;

          tpu_ready(tpu, channel);
          pulse = tpu->PARM.R[channel][TPU_FQM_PULSE_COUNT];
    return (pulse);
}  /* End tpu_fqm_get_pulse */
```

8 FQM Function Parameters

This section provides detailed descriptions of FQM function parameters stored in channel parameter RAM. Figure 3 shows the parameter RAM assignment used by the FQM function.

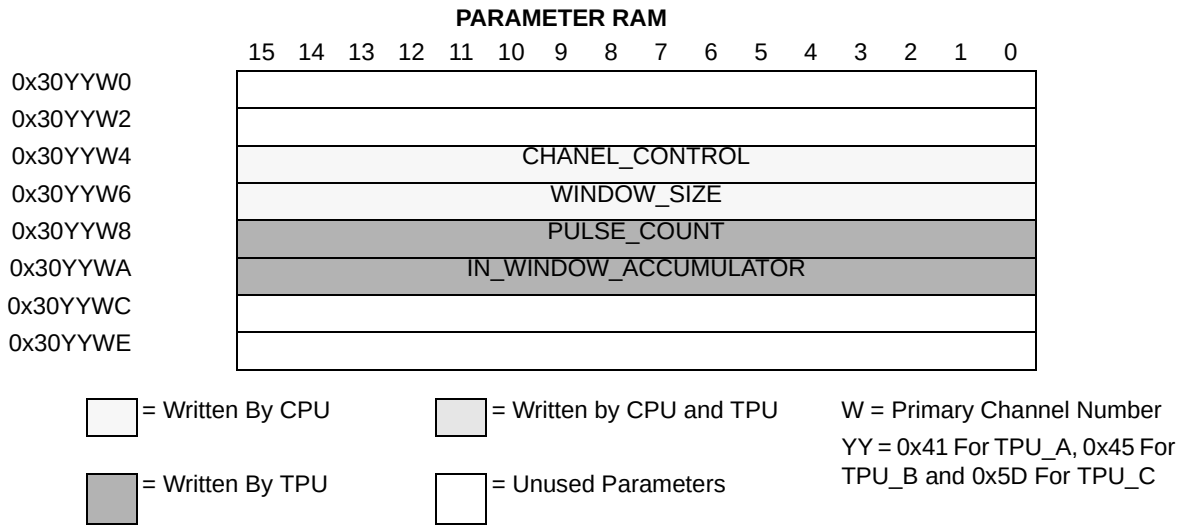


Figure 3. FQM Parameters

8.1 Channel Control

This parameter is used by the function to initialize the channel. It must be written by the CPU prior to issuing a host service request and assigning priority to the channel. The only legal values for this parameter are shown in Figure 3. Any other values cause indeterminate operation. Bits 9–15 are not used and are ignored.

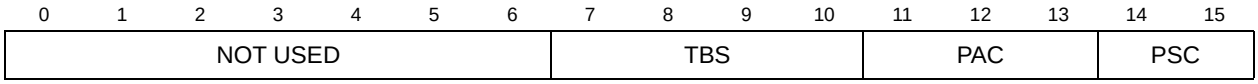


Figure 4. Channel Control Bit Encoding

Table 2. Channel Control Bit Definitions

Function	TBS				PAC			PSC	
-								1	1
Detect Rising Edge					0	0	1		
Detect Falling Edge					0	1	0		
Capture TCR1, Match TCR1	0	0	0	0					
Capture TCR1, Match TCR1	0	0	1	1					

8.1.1 PSC

These bits are used by the initialization state to configure the channel. Since this is an input function these bits must always be set to the NIL value (11).

8.1.2 PAC

These bits are used during function initialization to configure the transition detector. Although the detector itself can be set to detect rising edges, falling edges, both rising and falling edges, or to not detect any edge, FQM deals only with rising or falling edges, and thus recognizes only the PAC values that correspond to these two cases. The edge type selected must also be the same as the edge specified by host sequence bit 1. If the two selections are not the same, the function does not perform correctly.

Table 3. Channel Control PAC Definitions

PAC Setting	Definition	Value
Detect Rising Edge	TPU_FQM_RISE	0x1
Detect Falling Edge	TPU_FQM_FALL	0x2

8.1.3 TBS

These bits are used during initialization to select the timebase for the function. Either TCR1 or TCR2 can be selected.

Table 4. Channel Control TBS Definitions

TBS Setting		Value
Capture TCR1, Match TCR1	TPU_FQM_TCR1	0x1
Capture TCR2, Match TCR2	TPU_FQM_TCR2	0x3

8.2 WINDOW_SIZE

This parameter specifies the duration of the accumulation window. It is written by the CPU. Duration is specified in TCR clock ticks. The maximum value is 0x8000. Minimum window time is based on the service latency of the function, which varies according to the type and number of functions active at any one time. This parameter must be written prior to issuing a host service request and assigning priority to the channel. Once the channel is enabled, window size can be changed at any time while the channel is running. The new value takes effect when current window time expires.

8.3 PULSE_COUNT

This is the 16-bit result register for the function. It contains the number of pulses detected during the previous window. Since the register is written by the TPU at the end of each window time, in continuous mode the CPU has only one window time in which to read the stored value. In single shot mode, the value remains in the register until the next window accumulation is scheduled and that window time ends.

8.4 IN_WINDOW_ACCUMULATION

FQD uses this 16-bit location to store a running pulse accumulation value during each window time. The value is reset to zero at the beginning of each window. IN_WINDOW_ACCUMULATION is a scratchpad value. CPU reads of the register do not affect function operation, but CPU writes can corrupt an accumulation in progress.

9 Host Interface to the FQM Function

This section provides information concerning the TPU host interface to the FQM function.

9.1 Channel Function Select Registers

Encoded 4-bit fields within the channel function select registers specify one of 16 time functions to be executed on the corresponding channel. The Channel Function bits should be set to 0xC to select the FQM ROM function.

9.2 Host Sequence Registers

The host sequence field selects the mode of operation for the time function selected on a given channel. The meaning of the host sequence bits depends on the time function specified. Meanings of host sequence bits and host service request bits for pre-defined time functions will be provided in a subsequent draft of this document.

Table 5. Host Sequence Bit Definitions (0x30YY14 – 0x30YY16)

Bit Setting	Definition
0b00	Begin with Falling Edge, Single-Shot Mode
0b01	Begin with Falling Edge, Continuous Mode
0b10	Begin with Rising Edge, Single-Shot Mode
0b11	Begin with Rising Edge, Continuous Mode

9.3 Host Service Request Registers

The host service request field selects the type of host service request for the time function selected on a given channel. The meaning of the host service request bits is determined by time function microcode. See Table 6 for the Host Service routines defined for the FQM function

Table 6. Host Service Bit Definitions

Bit Setting	Definition
0b00	No Host Service (Reset Condition
0b01	Not Used
0b10	Initialize
0b11	Not Used

9.4 Channel Priority Registers

The channel priority registers (CPR1, CPR2) assign one of three priority levels to a channel or disable the channel.

10 Performance and Use of the FQM Function

10.1 Performance

Since microcode must execute whenever an edge is detected, there is a minimum pulse period which guarantees that all pulses will be counted. When a single FQM function is in use and no other TPU channels are active, the absolute minimum pulse width is 22 CPU cycles plus a TST time. To analyze the performance of an application that appears to approach the limits of the TPU, use the guidelines given in the TPU reference manual and the information in Table 7 below.

Table 7. FQM State Timing

State Number and Name	Max CPU Clock Cycles	RAM Accesses by TPU
S1 INIT_FQM	6	1
S2 FIRST_EDGE_FQM	8	2
S3 COUNT_EM_FQM Match only Transition only Match and Transition	 16 12 22	 4 2 4

NOTE: Execution times do not include the time slot transition time (TST = 10 or 14 CPU clocks).

10.2 Changing Mode

The host sequence bits are used to select the FQM function operating mode. Change host sequence bit values only when the function is stopped or disabled (channel priority bits =0b00). Disabling the channel before changing mode avoids conditions that cause indeterminate operation.

11 Frequency Measurement Examples

The FQM function counts pulses within a user-specified time window. The following examples show the capabilities of the function. Each example includes a description of the example, a diagram of initial parameter RAM content, initial control bit settings, and a diagram showing the relationship between the window and the pulses. Assume a pulse period of 75 TCR1 clock ticks. This is not required to set up the function but is used to illustrate what happens to partial pulses and edges concurrent with the end of window time. Unless otherwise noted, all examples use TPU channel 0.

11.1 Example 1

11.1.1 Description

Single-shot mode. Count the number of pulses beginning with a falling edge using TCR1. Accumulate for 500 (0x1F4) TCR1 clock ticks and stop. Store the accumulated value in location 0x30YYW08.

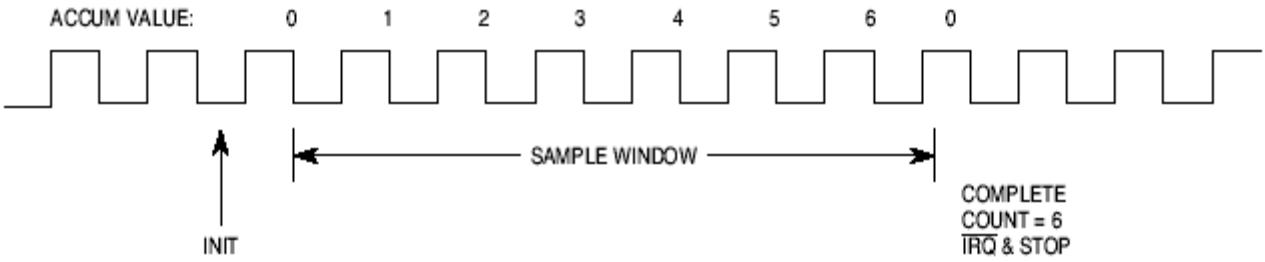
11.1.2 Initialization

Load parameter RAM as shown. Write HSQ =0b00, then issue HSR =0b10 to initialize. Enable channel interrupt, select the function in the channel function select register, and set the channel priority bits to start the function.

CH_CNTL = 0xB

WIND_SZ = 0x1F4

11.1.3 Timing Diagram



11.2 Example 2

11.2.1 Description

Single-shot mode. Count the number of pulses beginning with a rising edge using TCR2. Accumulate for 300 (0x12C) TCR2 clock ticks and stop. Store the accumulated value in location 0x30YYW08. Note that in this example the end of a pulse is coincident with the end of a window. If this edge passes through the digital filter before the window expires it is counted. However, if the edge has occurred but the digital filter has not qualified it, the pulse is not counted.

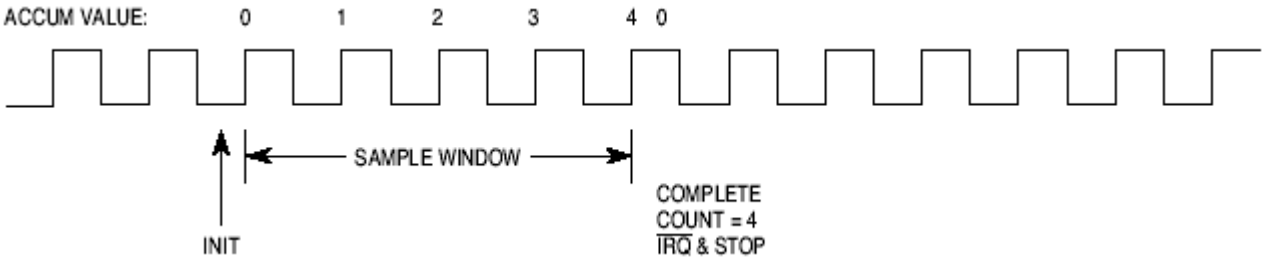
11.2.2 Initialization

Load parameter RAM as shown. Write HSQ =0b10, then issue HSR =0b10 to initialize. Enable channel interrupt, select the function in the channel function select register, and set the channel priority bits to start the function.

CH_CNTL = 0x67

WIND_SZ = 0x12B

11.2.3 Timing Diagram



11.3 Example 3

11.3.1 Description

Continuous mode. Count the number of pulses beginning with a falling edge using TCR2. Accumulate for 350 (0x15E) TCR2 clock ticks. At the end of each window time, store the accumulated value in location 0x30YYW08.

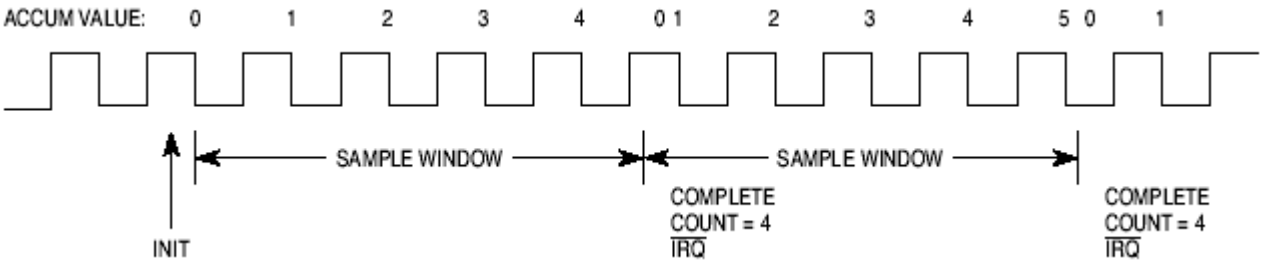
11.3.2 Initialization

Load parameter RAM as shown. Write HSQ = 0b01, then issue HSR = 0b10 to initialize. Enable channel interrupt, select the function in the channel function select register, and set the channel priority bits to start the function.

CH_CNTL = 0x6B

WIND_SZ = 0x15E

11.3.3 Timing Diagram



11.4 Example 4

11.4.1 Description

Continuous mode. Count the number of pulses beginning with a rising edge using TCR1. Accumulate for 300 (0x12C) TCR1 clock ticks and stop. Store the accumulated value in location 0x30YYW08. Note that in this example the end of a pulse is coincident with the end of a window. If this edge passes through the digital

filter before the window expires it is counted. However, if the edge has occurred but the digital filter has not qualified it, the pulse will be counted in the next window time.

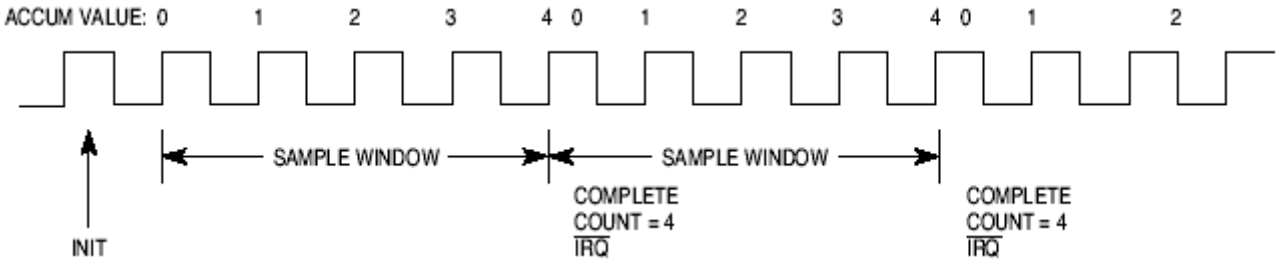
11.4.2 Initialization

Load parameter RAM as shown. Write HSQ = 0b11, then issue HSR = 0b10 to initialize. Enable channel interrupt, select the function in the channel function select register, and set the channel priority bits to start the function.

CH_CNTL = 0x07

WIND_SZ = 0x12C

11.4.3 Timing Diagram



12 Debugger Scripts

The following debugger scripts were written and used for the WindRiver SingleStep debugger version 7.6.2. The first just sets up the MIOS PWM0 to generate a continuous 4.882 kilo-hertz square wave.

12.1 MIOS Initialization Script for WindRiver SingleStep

```
echo "MIOS.DBG"
echo "SDS macro script to demonstrate the use of the MIOS"
echo "PWM and DASM. Based on appnote AN1778."
echo "al - original script, 1999"
echo "rd - modified with comments to the screen, 20APR2000"

#
echo "################################################"
#echo "set fast slew rate, and disable pull up/down in PDMCR"
#write -l 0x2fc03c = 0xf3000000

echo "################################################"
echo "ALL CHANNELS"
echo "Status and control for MCPSM (main clock"
```

```
echo "divider for the MIOS i.e. main divider for all"
echo "PWM's)"
echo "set prescaler to divide by 16"
write -w 0x306816 = 0x8000

echo "################################################"
echo "CHANNEL 0"
echo "PWM Period register (resolution of the waveform)"
echo "set PWM0 to generate a square wave 1/16 system clock"
echo "(assuming IMB running at half speed)"
write -w 0x306000 = 0x2
echo "PWM0 period = 0x2"

echo "PWM Pulse width register (How many periods --"
echo "from above-- that are high"
echo "PWM0 pulse width = 0x1"
write -w 0x306002 = 0x1

echo "PWM Status and control (Divider per PWM) = = 0x5400"
echo "PWM0 output, prescaler divide by 256"
write -w 0x306006 = 0x5400

echo "################################################"

exit
# End of script
TPU Read Channel 0 Parameter RAM Script
# start of script - tpu_pram.dbg
echo "this script reads TPU_A, TPU_B, or TPU_C channel 0 parameter RAM"
echo "rd. 18sept2002"
# Original version 18Sept2002
# modified from tpu_pwm.dbg
# 5 july 2001 - added uppercase.
echo ""
if ( $#argv < 1 ) then
echo "*****ERROR***** illegal TPU selected. Defaulting to TPU A"
  @ base = 0x304000
else
if ( $1 == 'a' || $1 == 'A' ) then
  @ base = 0x304000
else if ( $1 == 'b' || $1 == 'B' ) then
```

```
  @ base = 0x304400
else if ( $1 == 'c' || $1 == 'C' ) then
  @ base = 0x305c00
endif

@ hsradd = `? ( $base + 0x1a )`
echo "check location " `? -x ( $base + 0x1a )` " Host service request is cleared (=0)"
@ hstat = `read -Vrux $hsradd=short`
echo "Host service request = "$hstat
if ( $hstat == 0 ) then
   echo "Host service request completed"
 else
   echo "TPU " $1 " failed to start running"
endif

echo "Read TPU " $1 " Channel 0 Parameter RAM."
echo -b "Parameter 0 = \c"
read -ruxw `? -x ( $base + 0x100 )`=short
echo -b "Parameter 1 = \c"
read -ruxw `? -x ( $base + 0x102 )`=short
echo -b "Parameter 2 = \c"
read -ruxw `? -x ( $base + 0x104 )`=short
echo -b "Parameter 3 = \c"
read -ruxw `? -x ( $base + 0x106 )`=short
echo -b "Parameter 4 = \c"
read -ruxw `? -x ( $base + 0x108 )`=short
echo -b "Parameter 5 = \c"
read -ruxw `? -x ( $base + 0x10a )`=short
echo -b "Parameter 6 = \c"
read -ruxw `? -x ( $base + 0x10c )`=short
echo -b "Parameter 7 = \c"
read -ruxw `? -x ( $base + 0x10e )`=short

# end of script
```

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor