Application Note

AN2373/D
Rev. 0, 10/2002

Using the Pulse Width Modulation TPU Function (PWM) with the MPC500 Family

Randy Dees
TECD Applications

This TPU Programming Note is intended to provide simple C interface routines to the pulse width modulation TPU function (PWM). [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

This output function generates a pulse-width-modulated waveform in which the period and/or the high time can be changed at any time by the CPU. PWM uses two modes of operation: level and normal. In level mode, a 0% or a 100% duty-cycle waveform can be generated. In normal mode, waveforms with duty-cycles between 0% and 100% can be generated.

In general, any changed period or high time is used in subsequent waveform synthesis, after a low-to-high transition. An immediate update is possible in either mode. After an immediate update, the new period and/or high time is reflected in the output waveform during the immediate host-service state, instead of waiting for a subsequent low-to-high transition.

2 Description

To start a PWM waveform, the CPU configures or updates parameters PWMPER (period desired) and PWMHI (high time desired), then issues an HSR 0b10 for initialization. After CPU initialization (refer to Figure 1), the TPU generates a low-to-high transition and calculates the pulse timing (next fall time, next rise time). The time of the most recent rising edge is moved from parameter PWMRIS to parameter OLDRIS, where it can be read at any time by the CPU. Calculation of the fall time is made by adding OLDRIS to PWMHI. The next rise time is calculated by adding the period desired from PWMPER to the rise time, now in OLDRIS, and then placing the projected new rise time into PWMRIS.

[1] The information in this Programming Note is based on TPUPN17. It is intended to compliment the information found in that Programming Note.

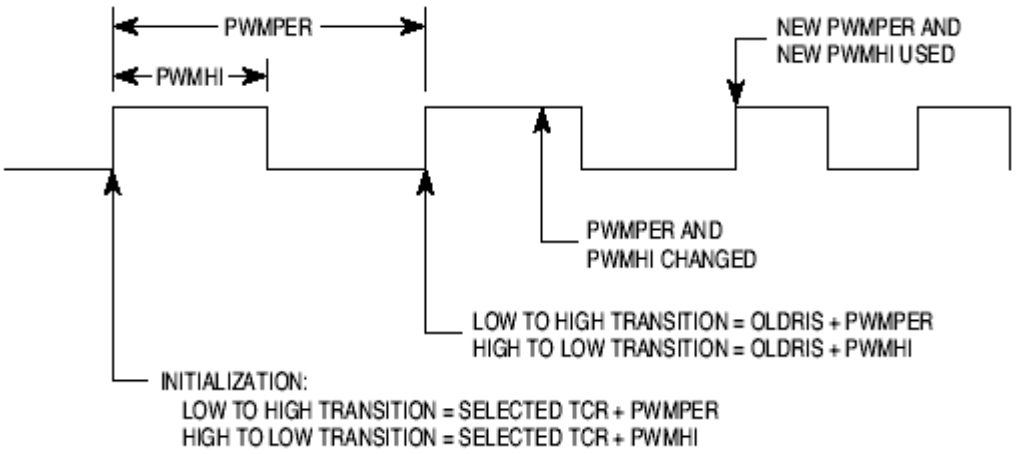

Figure 1. 50% Duty Cycle PWM Waveform

In level mode, where the high time in PWMHI is zero (indicating 0% duty cycle) or is equal to or greater than the period (indicating 100% duty cycle), a match without a pin transition is set up for the time (OLDRIS + PWMPER). In normal mode, a match and fall time is set up for the time (OLDRIS + PWMHI), and an interrupt request signal is asserted on each match event if the interrupt enable bit is set. To change the PWM parameters, the CPU coherently writes new 16-bit values to either PWMPER or PWMH. If both PWMPER and PWMH are to be changed, a coherent 32-bit write is required.

In both normal and level modes the new parameters are referenced to the next low-to-high transition. An immediate update of either or both parameters may be selected by the CPU by issuing an HSR 0b01. The immediate result to the waveform depends upon the point at which the immediate update is taken. For further information, see Section 10, “Performance and Use of the PWM Function**.”**

An optional CPU interrupt request can be made at the beginning of each pulse in any mode or after an immediate update. This allows the CPU to schedule parameter changes in relationship to a known point in the waveform.

3 FQM C Level API

- Initialization Function:
  - — void tpu_pwm_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, UINT16 period, UINT16 hightime, UINT8 timer)
- Change Operation Functions:
  - — void tpu_pwm_update(struct TPU3_tag *tpu, UINT8 channel, UINT16 period, UINT16 hightime, UINT8 immediate)
  - — void tpu_pwm_force(struct TPU3_tag *tpu, UINT8 channel, UINT8 state)
- Value Return Functions:
  - — UINT16 tpu_get_pwm_period(struct TPU3_tag *tpu, UINT8 channel)
  - — UINT16 tpu_get_pwm_hightime(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 Function (tpu_pwm_init)

void tpu_pwm_init

This function initializes the PWM function and sets the initial period and duty. To change the operating mode, this function can be called again. This function has the following parameters:

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.
- priority (UINT8) - 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

- period (UINT16) - This parameter sets the period of the PWM waveform.
- hightime (UINT16) - This parameter sets high time of the PWM waveform.
- timer (UINT8) - This parameter sets whether the TPU3 TCR1 or TCR2 clock should be used for measuring the incoming frequency.

3.2 Change Operation Function (tpu_pwm_update, tpu_pwm_force)

void tpu_pwm_update

This function allows the PWM parameters to be updated. The parameters are changed on the next rising edge of the PWM, unless the immediate parameter is set. void tpu_pwm_update(struct TPU3_tag *tpu, UINT8 channel, UINT16 period, UINT16 hightime, UINT8 immediate)

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.
- period (UINT16) - This parameter sets the period of the PWM waveform.

- hightime (UINT16) - This parameter sets high time of the PWM waveform.
- immediate (UINT8) - This parameter determines whether the updated parameters are effective immediately or at the beginning of the next PWM cycle.

Table 2. TPU PWM Update Immediate Definitions

Value‘	Definition
TPU_PWM_NORMAL	0x0
TPU_PWM_IMMED	0x1

void tpu_pwm_force

This function forces the state of the PWM channel to 0% (always low) and 100% (always high).

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.
- state (UINT16) - This parameter sets the period of the PWM waveform.

Table 3. TPU PWM Update Immediate Definitions

Value‘	Definition
TPU_PWM_LOW	0x0
TPU_PWM_HIGH	0x1

3.3 Value Return Value Functions (tpu_pwm_get_period, tpu_pwm_get_hightime)

UINT16 tpu_pwm_get_period

This function returns the current period of the PWM waveform.

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.

UINT16 tpu_pwm_get_hightime

This function returns the current hightime of the PWM waveform.

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - 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

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

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.
- priority (UINT8) - This is the new channel priority.

void tpu_disable

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

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.
- priority (UINT8) - This is the new channel priority.

void tpu_interrupt_enable

This function enables the interrupt bit for the specified channel.

- *tpu (struct TPU3_tag) - 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

This function disables the interrupt bit for the specified channel.

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.

void tpu_clear_interrupt

This function clears the interrupt bit for the specified channel.

- *tpu (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.

UINT8 tpu_check_interrupt

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 (struct TPU3_tag) - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel (UINT8) - This is the channel number that has the FQM function assigned to it.

4 Configuration of PWM Function

The CPU configures the PWM function as follows:

1. The appropriate channel priority bits are cleared, disabling the channel.
2. The PWM function number is written to the channel function select bits.

3. CHANNEL_CONTROL, HIGHTIME and PERIOD 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_pwm_init() function. See Section 3.1, “Initialization Function (tpu_pwm_init).”

5 Example Code

The following code shows an example program that initializes the TPU PWM ROM function using the C level API (see Section 5.1, “Code Listing”). This example initializes the PWM function on channel 0 and channel 1. A PIT interrupt is initialized to update the high time every 1 second. The first channel is updated normally (at the end of the current PWM cycle) and the second channel updates the hightime immediately. The period of the PWM is approximately 1.220 KHz with the MPC555 system clock running at 40 MHz. The

5.1 Code Listing

```
#include "mpc555.h"
#include "tpu_pwm.h"
#include "mpc500_util.h"

/* Initialization parameters  */
UINT32 loopctr = 0 ;// Loop counter for main loop
UINT32 pitctr = 0;  // Counter for number of PIT timeouts
struct TPU3_tag *tpua = &TPU_A;   /* pointer for TPU routines */

/* Example specific values */
UINT8 chan_a = 0;
UINT8 chan_b = 1;
UINT16 init_period = 0x2000; /* This generates a 1.2 KHz square wave initially. */
UINT16 init_hightime = 0x1000;
UINT16 step_size = 0x100;

void init555(int freq)
{
    USIU.SYPCR.R = 0xffffff03;  /* Disable watchdog timer.                *
                                 * WARNING: this is a WRITE ONLY register. */
    USIU.PLPRCRK.R = 0x55ccaa33;  /*Unlock PLPRC with special key */
if (freq == 56){
    USIU.PLPRCR.R = 0x00d3d000; /* 14x PLL operation on normal power mode */
}else
```

```
{
    USIU.PLPRCR.R = 0x0093d000; /* 10x PLL operation on normal power mode */
}

          UIMB.UMCR.B.HSPEED = 0;// Make IMB clock = System Clock
}
void initPIT(void)
{
// STEP 1: MODULE SPECIFIC INITIALIZATION
   USIU.PITC.B.PITC = 15624;// Setup count value - 1 second.
   USIU.PISCR.B.PITF = 1;// Freeze enabled to stop PIT
   USIU.PISCR.B.PTE = 1;// PIT enabled to start counting

// STEP 2: LEVEL ASSIGNMENT
   USIU.PISCR.B.PIRQ = 0x80; // Level 0 PIT interrupt

// STEP 3: ENABLE INTERRUPT
   USIU.PISCR.B.PIE = 1 ;// Enable PIT interrupt

// STEP 4: SET APPROPRIATE SIMASK BITS
   USIU.SIMASK.R = 0x40000000; // Enable level 0; others disabled
}

void pit_handler (void)
{
UINT16 old_hightime, new_hightime, period;

/* Toggle the mios PIO bits 11/12 for activity */
          MIOS1.MPIOSM32DR.B.D11 = !MIOS1.MPIOSM32DR.B.D11;
/* do second mios pin at end to show the time difference in the interrupt. */

          pitctr++;                                /* Increment PIT counter */
          USIU.PISCR.B.PS = 1;/* Negate PIT flag by writing "1" to it */

/* get current hightime and period - assumes both channels are the same */
          old_hightime = tpu_pwm_get_hightime(tpua,0);
          period = tpu_pwm_get_period(tpua,0);

          if (old_hightime > (period - step_size))
          {
                    new_hightime = step_size;
```

```
        }
        else
        {
                new_hightime = old_hightime + step_size;
        }
/* do first channel with normal update */
        tpu_pwm_update(tpua,chan_a,period, new_hightime, TPU_PWM_NORMAL);
/* do second channel with immediate update */
        tpu_pwm_update(tpua,chan_b,period, new_hightime, TPU_PWM_IMMED);

/* Now toggle the other MIOS pin */
        MIOS1.MPIOSM32DR.B.D12 = !MIOS1.MPIOSM32DR.B.D12;

}

/*************** MIOS PIO ***************/
/* Set up the MIOS Parallel port */
/* set bits 11 and 12 to Outputs and initialize to 11 high, 12 low */
void Init_MIOS_PIO (void)
{

/* set MIOS Parallel Port bits 11 and 12 to be outputs */
    MIOS1.MPIOSM32DDR.B.DDR11 = 1; /* setup MIOS Parallel Port bit 11 output */
    MIOS1.MPIOSM32DDR.B.DDR12 = 1; /* setup MIOS Parallel Port bit 12 output */

/* set initial condition of the GPIO bits (LEDs) */
    MIOS1.MPIOSM32DR.B.D11 = 1;
    MIOS1.MPIOSM32DR.B.D12 = 0;
}

/************** TPU Set up **************/
/* Set up the TPU....                      */

void setup_tpu(struct TPU3_tag *tpu)
{
     tpu->TPUMCR.R = 0x2020; /* TCR1 prescaler divide by 2, 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; /* (but this example will not use TPU Interrupt) */
}
```

```
void main()
{
          init555(40);                              /* Perform a simple 555 initialization */
          Init_MIOS_PIO();
          initPIT();                                /* Init PIT to generate interrupts */
          setup_tpu(tpua); /* Do general TPU set up. */
          tpu_pwm_init(tpua, chan_a, TPU_PRIORITY_HIGH, init_period, init_hightime, TPU_PWM_TCR1);
          tpu_pwm_init(tpua, chan_b, TPU_PRIORITY_HIGH, init_period, init_hightime, TPU_PWM_TCR1);



#ifndef __MWERKS__
          asm(" mtspr EIE, r3");/* FINAL STEP: SET MSR[EE], MSR[RI] BITS */
#endif

#ifdef __MWERKS__

asm{
                    mtspr EIE, r3
}
#endif

          while(1)
                    {
                    loopctr++;          // Increment loopctr for something to do
                    }
}
```

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 100 ns. This example sets the period to 0x2000 (8192) TCR1 clock periods which equals 819 microseconds or 1220 Hetrz.The high pulse width starts at 409.6 microseconds and is incremented by 25.6 microseconds until it reaches 640 microseconds at which time it is reset to 25.6 microseconds.

$$\text{PWMFrequency} = \frac{1}{\text{TCR1 period x PERIOD}}$$

$$\text{PWMFrequency} = \frac{1}{\text{100ns x 32768}}$$

$$\text{PWMFrequency} = \frac{1}{\text{819.2microS}}$$

$$\text{PWMFrequency} = 1220.7\text{Hz}$$

5.3 Example Exception Code

```
        .name "exceptions.s"

        .section .abs.00000100
                b _start            ; System reset exception, per crt0 file

        .section .abs.00000500
                b external_interrupt_exception

        .text
external_interrupt_exception:

        .equ    SIVEC,              0x2fc01c  ;Register addresses
                                              ; STEP 1:  SAVE "MACHINE CONTEXT"
        stwu    sp, -80 (sp); Create stack frame and store back chain
        stw             r3, 36 (sp); Save working register
        mfsrr0  r3                            ; Get SRR0
        stw             r3, 12 (sp); and save SRR0
        mfsrr1  r3                            ; Get SRR1
        stw             r3, 16 (sp); and save SRR1



                                    ; STEP 2:  MAKE MSR[RI] RECOVERABLE
        mtspr   EID, r3             ; Set recoverable bit
                                    ; Now debugger breakpoints can be set



                                    ; STEP 3: SAVE OTHER APPROPRIATE CONTEXT
        mflr    r3                            ; Get LR
        stw             r3, 8 (sp); and save LR
        mfxer   r3                            ; Get XER
        stw             r3, 20 (sp); and save XER
        mfspr   r3, CTR             ; Get CTR
        stw             r3, 24 (sp); and save CTR
        mfcr    r3                            ; Get CR
        stw             r3, 28 (sp); and save CR
        stw             r0, 32 (sp); Save R0
        stw             r4, 40 (sp); Save R4 to R12
        stw             r5, 44 (sp)
        stw             r6, 48 (sp)
```

```
stw             r7, 52 (sp)
stw             r8, 56 (sp)
stw             r9, 60 (sp)
stw             r10, 64 (sp)
stw             r11, 68 (sp)
stw             r12, 72 (sp)

                        ; STEP 4: DETERMINE INTERRUPT SOURCE
lis             r3, SIVEC@ha ; Load higher 16 bits of SIVEC address
lbz             r3, SIVEC@l (r3) ; Load Interrupt Code byte from SIVEC
                ; Interrupt Code will be jump table index

lis             r4, IRQ_table@h ; Load interrupt jump table base address
ori             r4, r4, IRQ_table@l
add             r4, r3, r4          ; Add index to table base address
mtlr    r4                          ; Load result address to link register


                        ; STEP 5: BRANCH TO INTERRUPT HANDLER
blrl                    ; Jump to Execution Routine (subroutine)
                        ; (After returning here, restore context)


                        ; STEP 6: RESTORE CONTEXT
lwz             r0, 32 (sp); Restore gprs except R3
lwz             r4, 40 (sp)
lwz             r5, 44 (sp)
lwz             r6, 48 (sp)
lwz             r7, 52 (sp)
lwz             r8, 56 (sp)
lwz             r9, 60 (sp)
lwz             r10, 64 (sp)
lwz             r11, 68 (sp)
lwz             r12, 72 (sp)
lwz             r3, 20 (sp); Get XER
mtxer   r3                          ; and restore XER
lwz             r3, 24 (sp); Get CTR
mtctr   r3                          ; and restore CTR
lwz             r3, 28 (sp); Get CR
mtcrf   0xff, r3  ; and restore CR
lwz             r3, 8 (sp); Get LR
```

Example Exception Code

```
        mtlr    r3                          ; and restore LR
        mtspr   NRI, r3             ; Clear recoverable bit, MSR[RI]
                                    ; Note: breakpoints CANNOT be set
                                    ; from now thru the rfi instruction
        lwz             r3, 12 (sp); Get SRR0 from stack
        mtsrr0  r3                          ; and restore SRR0
        lwz             r3, 16 (sp); Get SRR1 from stack
        mtsrr1  r3                          ; and restore SRR1
        lwz             r3, 36 (sp); Restore R3
        addi    sp, sp, 80; Clean up stack


                                    ; STEP 7:  Return to Program
        rfi                         ; End of Interrupt

; ====================================================================
; Branch tablefor the different SIVEC Interrupt Code values:

IRQ_table:                          ; Branch forever if routine is not written

irq_0:          b       irq_0
level_0:  b     pit_handler; Branch to PIT C routine
irq_1:          b       irq_1
level_1:  b     level_1
irq_2:          b       irq_2
level_2:  b     level_2
irq_3:          b       irq_3
level_3:  b     level_3
irq_4:          b       irq_4
level_4:  b     level_4
irq_5:          b       irq_5
level_5:  b     level_5
irq_6:          b       irq_6
level_6:  b     level_6
irq_7:          b       irq_7
level_7:  b     level_7
```

6 FQM C Level API Code

6.1 PWM Initialization Function

The *tpu_pwm_init* initialization routine initializes the channel to run the PWM function.

```
void tpu_pwm_init(struct TPU3_tag *tpu, UINT8 channel,
                  UINT8 priority, UINT16 period, UINT16 hightime, UINT8 timer)
{
UINT16 channel_control;
UINT16 tbs;
UINT8 hsq;

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

    /*  PWM is function 0x3                 */
    tpu_func( tpu, channel, TPU_FUNCTION_PWM);

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

   /* mask off illegal values */

    tbs = ( timer & TPU_PWM_TBS_MASK ) << 5;

    /* Initialize Parameter RAM                                        */
    channel_control = ( tbs | TPU_PWM_PAC | TPU_PWM_PSC );

    tpu->PARM.R[channel][TPU_PWM_CHANEL_CONTROL] = channel_control;
    tpu->PARM.R[channel][TPU_PWM_PWMHI] = hightime;
    tpu->PARM.R[channel][TPU_PWM_PWMPER] = period;


/***********************************************/
/*    Configure the Channels.                  */
/***********************************************/
   tpu_hsr(tpu, channel, TPU_PWM_INIT);


    /* Enable channel by assigning a priority to them. */
```

```
    tpu_enable(tpu, channel, priority);

}  /* End tpu_pwm_init */
```

6.2 PWM Update Function

The *tpu_pwm_update* routine should be called to change either the high time and the period of the sample window.

```
void tpu_pwm_update(struct TPU3_tag *tpu, UINT8 channel,
                    UINT16 period, UINT16 hightime, UINT8 immediate)
{
UINT32 update_val32;


          update_val32 = ((hightime << 16 ) | (period));


          tpu->PARM.L[channel][TPU_PWM_HIPER32] = update_val32;
          if (immediate)
                    {
                    tpu_hsr(tpu, channel, TPU_PWM_IMMED_UPDATE);
           }
}  /* End tpu_pwm_update */
```

6.3 PWM Get Value Return Functions

The function *tpu_pwm_get_hightime* returns the number of TCR clocks of the current high time of the PWM channel.

```
UINT16    tpu_pwm_get_hightime(struct TPU3_tag *tpu, UINT8 channel)
{
          UINT16 hightime;


          hightime = tpu->PARM.R[channel][TPU_PWM_PWMHI];
    return (hightime);
}  /* End tpu_pwm_get_hightime */
```

6.4 PWM Get Value Return Functions

The function *tpu_pwm_get_period* returns the number of TCR clocks of the current period of the PWM channel.

```
UINT16    tpu_pwm_get_period(struct TPU3_tag *tpu, UINT8 channel)
```

```
{
          UINT16 period;


          period = tpu->PARM.R[channel][TPU_PWM_PWMPER];
    return (period);
}  /* End tpu_pwm_get_hightime */
```

7 PWM Function Parameters

This section provides detailed descriptions of PWM function parameters stored in channel parameter RAM. Figure 2 shows the parameter RAM assignment used by the PWM function.

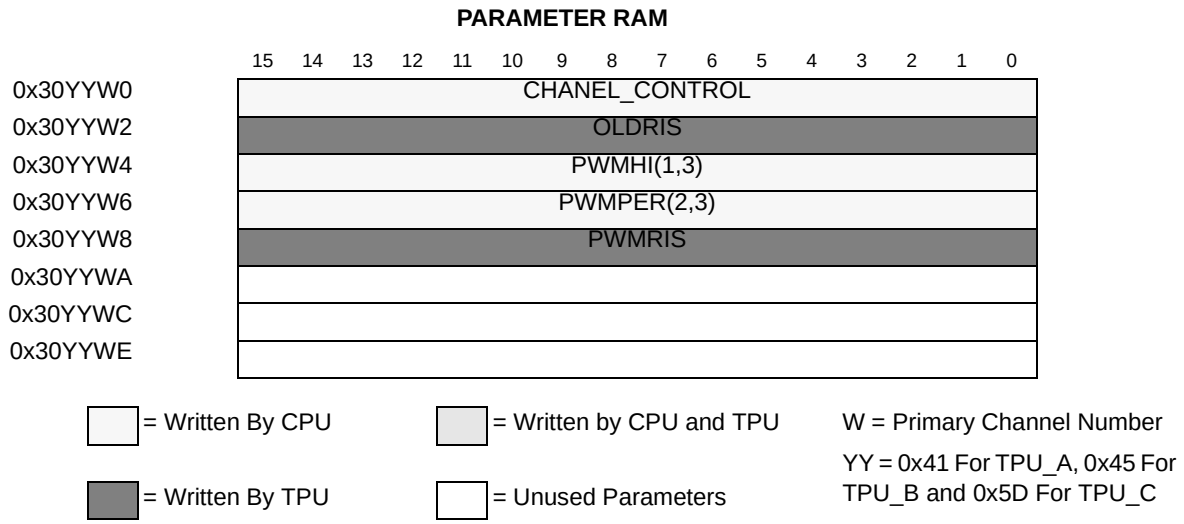

Figure 2. PWM Parameters

7.1 Channel Control

CHANNEL_CONTROL contains the channel latch controls and configures the PSC, PAC, and TBS fields. The PSC field forces the output level of the pin directly without affecting the PAC latches, or forces the output level to the state specified by the PAC latches. The PAC field specifies the pin logic response as either a timer channel input or output. The TBS field configures a channel pin as input or output and configures the time base for output match/input capture events.

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 2. Any other values cause indeterminate operation. Bits 9–15 are not used and are ignored.

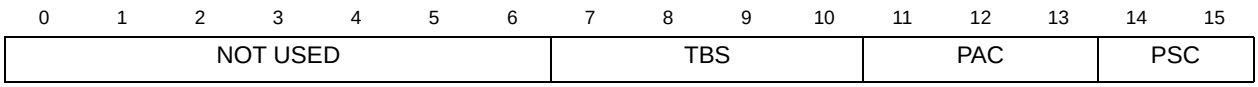

Figure 3. Channel Control Bit Encoding

NOTE:

This channel must be configured as an output because the PWM function is indeterminate when programmed as an input.

Table 4. Channel Control Bit Definitions

Function	TBS				PAC			PSC	
Force Pin as Specified by PAC Latches								0	0
Force Pin High								0	1
Force Pin Low								1	0
Do Not Force Any State								1	1
Do Not Change PAC					1	x	x		
Output Channel	0	1	x	x					
Capture TCR1, Compare TCR1	0	1	0	0					
Capture TCR2, Compare TCR2	0	1	1	1					
Do Not Change TBS	1	1	x	x					

7.1.1 PSC

The PSC field determines the setting of the pin after initialization. In normal mode, PSC is set to force the pin high. In level mode, where a 0% duty cycle is desired, PSC should be set to force the pin low at initialization. The PAC field specifies the pin logic response as a timer channel output; however, the PWM function does not use the PAC field, but uses direct control by the microcode. CHANNEL_CONTROL must be written by the CPU before initialization.

7.1.2 PAC

Since the PWM function is an output function, the PAC bits are not used and are therefore set to the value of 0x4.

7.1.3 TBS

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

Table 5. Channel Control TBS Definitions

TBS Setting		Value
Capture TCR1, Match TCR1	TPU_FQM_TCR1	0x4
Capture TCR2, Match TCR2	TPU_FQM_TCR2	0x7

7.1.4 OLDRIS

OLDRIS is the time of the previous low-to-high transition. The PWM microcode uses this value when calculating the next rise time. The TPU updates this parameter and should not be changed by the user.

7.1.5 PWMRIS

PWMRIS is the current calculated rise time calculated at the beginning of the pulse (on the low-to-high transition) by adding OLDRIS to PWMPER. The TPU updates this parameter.

7.1.6 PWMHI

PWMHI, which is updated by the CPU, is the current pulse high time that may be updated at any time. Estimate for best-case minimum value for PWMHI is greater than 32 system clocks, assuming a single channel operating. When more than one channel is operating, the minimum value for PWMHI depends on TPU configuration (the variables are described in Section 10, “Performance and Use of the PWM Function”). The maximum value is 0x8000. The user should calculate case timing to ensure proper execution of this function.

7.1.7 PWMPER

PWMPER, which is updated by the CPU, is the current PWM period and is used by the TPU to calculate the next low-to-high transition time. Estimate for best-case minimum value for PWMPER is greater than 32 system clocks, assuming that a single channel is operating. When more than one channel is operating, the minimum value for PWMPER depends on TPU configuration (the variables are described in Section 10, “Performance and Use of the PWM Function”). The maximum usable value is that which satisfies the condition: (PWMPER – PWMHI) is less than or equal to 0x8000. PWMHI and PWMPER must be accessed coherently. The user should calculate the case timing to ensure proper execution of this function. Normal, 100%, and 0% duty cycles are defined as follows.

0% → PWMHI = 0
100% → PWMPER ≤ PWMHI, AND PWMHI ≠ 0
Else normal → PWMPER > PWMHI, AND PWMHI ≠ 0

8 PWM Header File

The following header file listing provides common definitions for the PWM functions.

```
#include "m_tpu3.h"


#ifdef  __cplusplus
extern "C" {
#endif


/************************************************************/
/*            Definition of terms and initial settings      */
/************************************************************/


/* Define HSQ values (mode) */
/* HSQ values are not used by the PWM function. */


/* Define the Immediate values */
```

```
#define TPU_PWM_NORMAL 0x0/* Update normally at end of next cycle */
#define TPU_PWM_IMMED 0x1/* Update values immediately */
#define TPU_PWM_IMMED_MASK 0x1 /* MASK for illegal values */

/* State Definitions */
#define TPU_PWM_LOW0x0
#define TPU_PWM_HIGH0x1

/* Define HSR values */
#define TPU_PWM_NO_HOST 0x0/* No Host Service (Reset Condition) */
#define TPU_PWM_IMMED_UPDATE0x1/* Update PWM parameters immediately */
#define TPU_PWM_INIT 0x2      /* Initialize  */
#define TPU_PWM_NOT_USED_30x3/* Not Used */



/* Define test result values */
#define TPU_PWM_TRUE  1
#define TPU_PWM_FALSE 0

/* Define TPU_Channel Control. PSC is always 0b1 (start high) */
#define TPU_PWM_PSC0x1
#define TPU_PWM_PSC_MASK 0x3
/* PSC is always 0b1xx (shift by 2 for channel control location */
#define TPU_PWM_PAC 0x10
#define TPU_PWM_PAC_MASK 0x7

/* Define the Timer values */
#define TPU_PWM_TCR14/* Capture TCR1, Match TCR1 */
#define TPU_PWM_TCR27/* Capture TCR2, Match TCR2 */
#define TPU_PWM_TBS_MASK 0xF



/* Define parameter RAM locations */
#define TPU_PWM_CHANEL_CONTROL0/* Channel Control TCR1 or 2 */
#define TPU_PWM_OLDRIS        1         /* Old Rise time - used by TPU */
#define TPU_PWM_PWMHI         2         /* High time of the period */
#define TPU_PWM_PWMPER        3         /* Period */
#define TPU_PWM_PWMRIS        4         /* Next rise time */
#define TPU_PWM_PRM_5         5         /* Not Used */
#define TPU_PWM_PRM_6         6         /* Not Used */
#define TPU_PWM_PRM_7         7         /* Not Used */
```

```
/* define 32-bit parameter RAM Location for accessing */
#define TPU_PWM_HIPER321 /* the high time and period together */

/* Prototype of functions */

void tpu_pwm_init(struct TPU3_tag *tpu, UINT8 channel,
                  UINT8 priority, UINT16 period, UINT16 hightime, UINT8 timer);

void tpu_pwm_update(struct TPU3_tag *tpu, UINT8 channel,
                  UINT16 period, UINT16 hightime, UINT8 immediate);

void tpu_pwm_force(struct TPU3_tag *tpu, UINT8 channel, UINT8 state);

UINT16 tpu_pwm_get_period(struct TPU3_tag *tpu, UINT8 channel);

UINT16 tpu_pwm_get_hightime(struct TPU3_tag *tpu, UINT8 channel);

void setup_tpu(struct TPU3_tag *tpu);

#ifdef  __cplusplus
}
#endif
```

9 Host Interface to PWM Function

This section provides information concerning the TPU host interface to the PWM 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 0x3 to select the PWM ROM function.

9.2 Host Sequence Registers

The host sequence register is not used by the PWM function.

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 PWM function.

Table 6. Host Service Bit Definitions

Bit Setting	Definition
0b00	No Host Service (Reset Condition
0b01	Immediate Update
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 PWM Function

10.1 Performance

Like all TPU functions, the PWM function performance in an application is to some extent dependent upon the service time (latency) of other active TPU channels. This is due to the operational nature of the scheduler. When a single PWM channel is in use and no other TPU channels are active, the minimum time between any two pulse edges is greater than 32 CPU clocks. When more TPU channels are active, performance decreases. However, worst-case latency in any TPU application can be closely estimated. 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 the PWM state timing table below.

10.2 Changing Duty Cycle

The CPU can change the duty cycle at any time once the TPU has completed the initialization state (indicated by HSR 0b00 or a CPU interrupt request). Changes are made by writing a new high time value to PWMHI in the channel's parameter RAM.

The minimum duty cycle (and the maximum non-100% duty cycle) is dependent on the number of active TPU channels and the maximum channel latency as discussed above. A 0% duty cycle is generated by setting PWMHI = 0. A 100% duty cycle is scheduled by setting PWMPER less than or equal to PWMHI when PWMHI is not equal to zero.

Duty cycle changes take effect at the completion of the current period unless an immediate update (HSR 0b01) is also requested. Immediate updates may be requested for any duty cycle including 0% and 100%. A new PWMHI value with an immediate update HSR causes the TPU to change the currently scheduled high-to-low time. This can cause the undesired side effect of an improper duty cycle for one period.

10.3 Changing Period

Once the TPU has completed the initialization state the CPU may at any time specify a new period by writing to the PWMPER parameter. Unless the CPU also generates an immediate update service request the new period takes effect at the beginning of the next period, as shown in Figure 1. That is, a new rise time is calculated at the next low-to-high transition. Thus, the current period is allowed to complete before the new one begins.

10.4 Counting Periods

The TPU generates a CPU interrupt service request during the channel service at the beginning of each period. The CPU can respond to these requests to keep track of how many periods have elapsed. In this way, new pulse widths can be scheduled at a known position in time.

10.5 Stopping the Function

Once PWM operation is initialized on a channel, it runs without CPU intervention until a reset occurs. If it is necessary to turn off a PWM channel, the CPU can write zeros to the channel function select bits in registers CFSR[0:3]. This disables the function on the channel. Another way to disable output is to select 0% or 100% duty cycle in the channel parameter RAM. In this case the PWM continues to run and receive channel service but no transitions are seen on the pin.

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