Application Note

AN2375/D
Rev. 0, 10/2002

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

Mario Perez II
Metrowerks

This TPU Programming Note is intended to provide simple C interface routines to the multichannel pulse width modulation TPU function (MCPWM). [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 TPU output function uses externally-gated multiple channels to produce sophisticated pulse-width modulated (PWM) signals. These signals can be used for a variety of applications including motor control. The function allows a user to select edge-aligned or center-aligned timing relationships between multiple PWM waveforms. Center-aligned relationships include dead time and inversion options to support driving H-bridges and inverters. MCPWM can also generate a programmable periodic CPU interrupt request at a programmable periodic rate.

2 Detailed Description

Standard TPU PWM functions use only one channel to produce a PWM output, but the minimum and maximum pulse widths (other than 0% or 100%) that can be obtained are limited by the latencies of TPU functions running on other channels. To produce a very short pulse, the same channel must be serviced quickly twice in succession, and this may not be possible when there is activity on other channels. Although this restriction does not present a problem in many applications, there are others, including motor control, where high resolution control of the high time is required over the full 0 to 100% duty cycle range. In motor control applications, it is also desirable to have defined timing relationships between multiple PWM signals in order to reduce ripple currents and to prevent shoot-through currents on the phase drivers.

The multichannel PWM function uses two TPU channels that are externally gated (using single EOR gates) together to form a single PWM signal. This allows a full 0 to 100% duty cycle range under a much wider range of TPU operating conditions than other PWM functions. All MCPWM timing acquisition is accomplished using the TPU timebase TCR1.

The MCPWM function can operate in edge-aligned (EA) mode or in center-aligned (CA) mode.

[1]The information in this Programming Note is based on TPUPN05. It is intended to compliment the information found in that Programming Note.

2.1 Edge-Aligned (EA) Mode

During EA operation, one TPU channel operates as a master and a number (n, in the range 1 to 15) of slave channels are externally gated with the master to generate “n” PWM outputs. External EOR gates are used. Figure 1 is an EA mode schematic.

In EA mode, the PWM outputs have aligned rising edges. This mode is more channel-efficient than CA mode and is the best choice for applications that require general-purpose, high-speed, high-resolution pulse-width modulation. Figure 2 shows EA mode output waveforms.

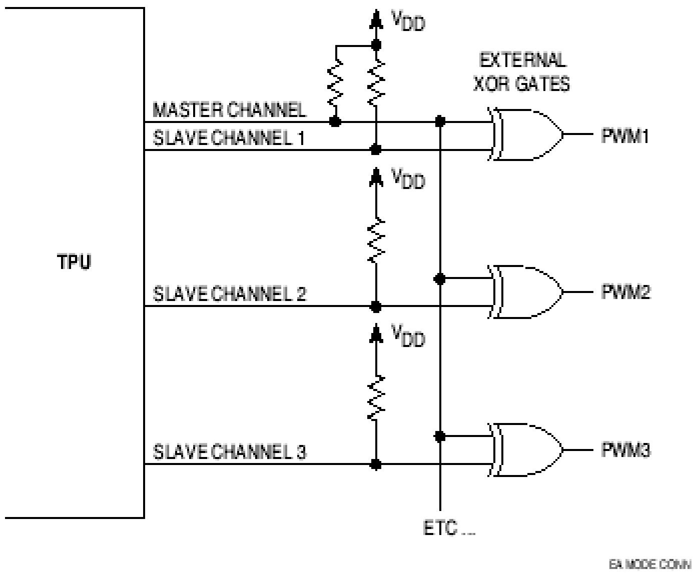

Figure 1 EA Mode External Gate Schematic

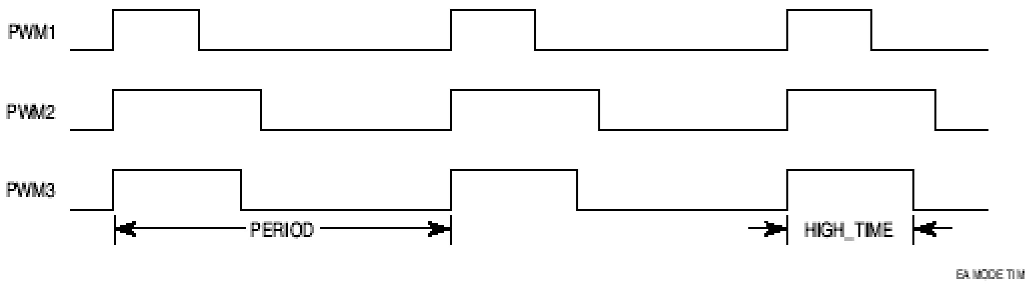

Figure 2 EA Mode Output Waveforms

2.2 Center-Aligned (CA) Mode

During center-aligned operation, one TPU channel operates as a master timing channel, providing synchronization for 1 or more pairs of slave channels. An even number (n, in the range 2 to 14) of slave channels are externally gated in pairs to generate (n / 2) PWM outputs. Figure 3 is a CA mode schematic.

In CA mode, the PWM outputs have center-aligned high times. To operate in CA mode, the function uses pairs of slave channels that perform slightly different operations. The first channel of the pair is referred to as a Type A slave, and the second channel of the pair is referred to as a Type B slave. In CA mode, the output waveform from the master channel is normally not used. Figure 4 shows CA mode PWM output waveforms.

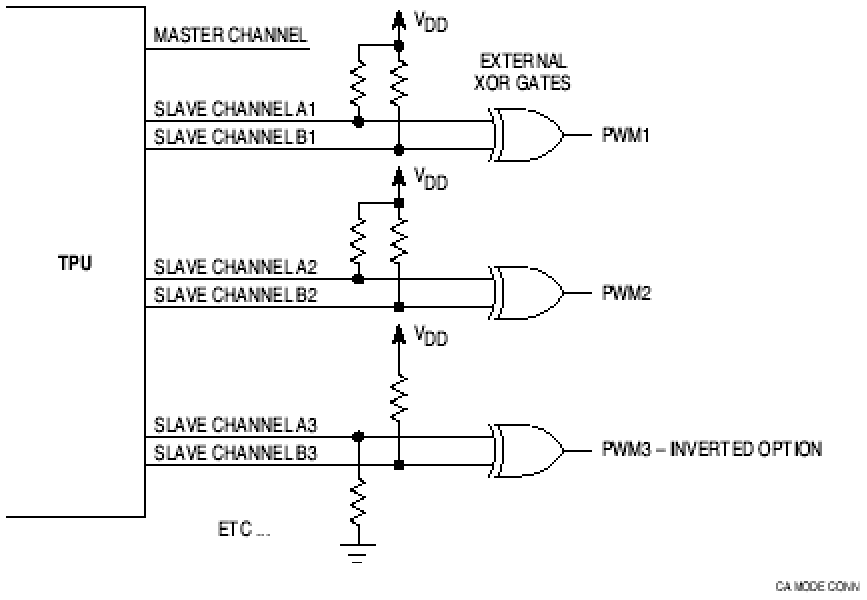

Figure 3 CA Mode External Gate Schematic

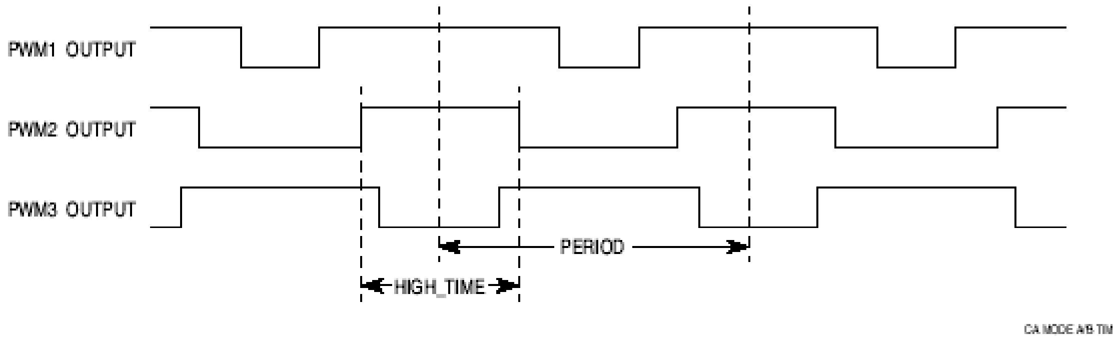

Figure 4 CA Mode Type A and Type B Output Waveforms

CA mode includes optional dead time and inversion functions. Dead time effectively reduces the PWM duty cycles slightly. If two CA mode PWM channels, one with a specified dead time and one with no specified dead time, reference the same high time, the channel with dead time specified has a slightly shorter high time and thus does not change state at the same time as the other channel. The user can specify a dead time accurate to one TPU timebase clock tick. The user can also choose to invert one or more of the PWM outputs during initialization. These features allow MCPWM to easily drive inverter-type applications. Figure 5 shows CA mode inversion and dead-time output waveforms.

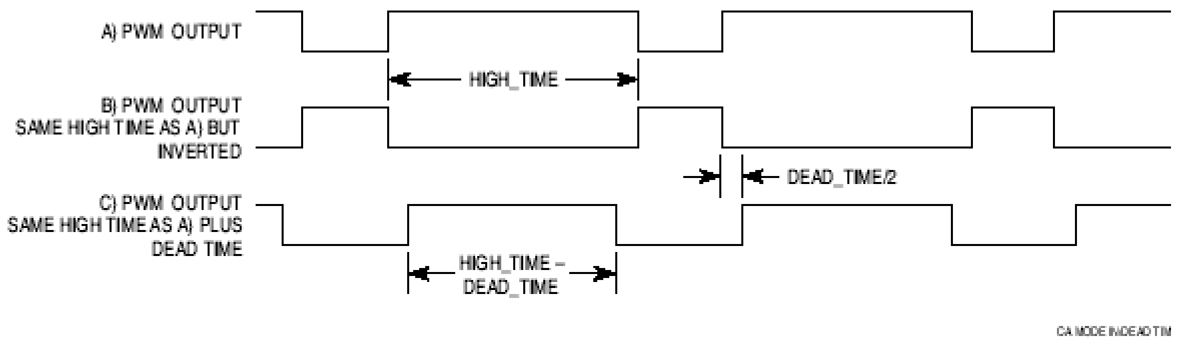
A) PWM OUTPUT
HIGH_TIME
B) PWM OUTPUT SAME HIGH TIME AS A) BUT INVERTED
DEAD_TIME/2
C) PWM OUTPUT SAME HIGH TIME AS A) PLUS DEAD TIME
HIGH_TIME – DEAD_TIME
CA MODE INDEADTIM

Figure 5 CA Mode Inverted and Dead-Time Waveforms

In both operating modes, the master MCPWM channel can be programmed to generate an interrupt service request to the host CPU when a specified number of PWM periods have elapsed. In many applications, this feature can be used to make the CPU update PWM duty cycles at a predetermined rate. The programmable interrupt feature is particularly useful for sine-wave modulated PWM production. The programmable interrupt feature can also be used as a simple periodic interrupt timer for the CPU — in this case, only one channel, programmed as a master, is needed.

2.3 PTA C Level API

Rather then controlling the TPU registers directly, the MCPWM API routines in this TPU Programming Note may be used to provide a simple and easy interface. There are 5 routines for controlling each of the MCPWM modes in 2 files (tpu_mcpwm.h and tpu_mcpwm.c). The tpu_mcpwm.h file should be included in any files that use the routines. This file contains the function prototypes and useful #defines. Each of the routines in tpu_mcpwm.c will be looked at in detail. The routines are:

- Initialization Functions:
  - — void tpu_mcpwm_master_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, UINT16 period, UINT8 irq_rate);
  - — void tpu_mcpwm_slave_edgemode_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, UINT16 period, UINT16 high_time, UINT8 high_time_ptr, UINT8 master_channel);
  - — void tpu_mcpwm_slavea_centermode_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, UINT16 period, UINT8 dead_time, UINT8 high_time_ptr, UINT8 polarity, UINT8 master_channel);
  - — void tpu_mcpwm_slaveb_centermode_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, UINT8 high_time, UINT8 polarity, UINT8 slave_a_channel);
- Update Function:
  - — void tpu_mcpwm_update_hightime(struct TPU3_tag *tpu, UINT8 channel, UINT16 high_time, UINT8 mode);

2.3.1 void tpu_mcpwm_master_init

This function is used to initialize a channel to run as a MCPWM master channel for either EA or CA mode. This function has 5 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 TPU channel number of the MCPWM master channel. The master must be assigned the lowest channel number of all MCPWM channels.
- priority - This is the priority to assign the channel. This parameter should be assigned a value of: TPU_PRIORITY_HIGH, TPU_PRIORITY_MIDDLE or TPU_PRIORITY_LOW.
- period - This defines the period of the PWM output in TCR1 timebase clocks. Period must be less than 0x4000.
- irq_rate - This determines the frequency of periodic interrupt requests generated to the host CPU. Any 8 bit value is valid. If the irq_rate = 0 then an interrupt request is generated every 256 PWM periods.

2.3.2 void tpu_mcpwm_slave_edgemode_init

This function is used to initialize a channel to run as a MCPWM slave edge-aligned channel. This function has 7 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 TPU channel number of the MCPWM edge mode slave channel. This parameter must be assigned a higher value than the channel number of the associated master channel.
- priority - This is the priority to assign the channel. This parameter should be assigned the same value as the priority of the associated master channel.
- period - This defines the period of the PWM output in TCR1 clocks. Period must be less than 0x4000. The PERIOD parameters of all slaves must have the same value as that of the master that they are referencing.
- high_time - This defines the high time of the PWM output waveform and is specified in TCR1 clocks over the range 0 to PERIOD, representing a 0 to 100% duty cycle.
- high_time_ptr - This contains the TPU memory address of the value to be used for the high time of the PWM output waveform. Any TPU memory location can be used, but in order to use the high time value specified by the previous parameter, this pointer should be set to (channel * 0x10) + 2. For example, if this slave is initialized to run on TPU channel 7, then high_time_ptr should be initialized to 0x72 in order to use the high_time value specified above.
- master_channel - This is the TPU channel number of the MCPWM master channel.

2.3.3 void tpu_mcpwm_slavea_centermode_init

This function is used to initialize a channel to run as a MCPWM slave A center-aligned channel with either inverted or non-inverted polarity. This function has 8 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 TPU channel number of the MCPWM slave A center-aligned channel. This parameter must be assigned a higher value than the channel number of the associated master channel. In CA mode, the slave A channel of a pair must be assigned a lower channel number than

the slave B channel. When two CA mode pairs are used to form an inverter driver (the second pair has dead time and references the CURRENT_HIGH_TIME parameter of the first pair), the pair without dead time must be assigned lower channel numbers than the pair with dead time.

- priority - This is the priority to assign the channel. This parameter should be assigned the same value as the priority of the associated master channel.
- period - This defines the period of the PWM output in TCR1 clocks. Period must be less than 0x4000. The PERIOD parameters of all slaves must have the same value as that of the master that they are referencing.
- dead_time - This contains the total dead time to be subtracted from the high time of the PWM, specified in TCR1 clocks. The range of dead_time is between 0 and 0xFF.
- high_time_ptr - This contains the TPU memory address of the value to be used for the high time of the PWM output waveform. In order to use the value specified by the high time parameter in the slave B center-aligned mode initialization call, this pointer should be set to (slave B channel * 0x10) + 1. For example, if this channel's slave B pair is initialized to run on TPU channel 5, then high_time_ptr should be initialized to 0x51 in order to use the high_time value specified in the slave B initialization. When a second CA mode slave pair is being used to generate a PWM with dead time, high_time_ptr should be initialized to (1st slave B channel * 0x10) + 3. For example, if the 1st slave B channel is initialized to run on TPU channel 5, then high_time_ptr should be initialized to 0x53.
- polarity - This is the polarity of the slave A center-aligned channel. This parameter should be assigned a value of: INVERTED or NONINVERTED.
- master_channel - This is the TPU channel number of the MCPWM master channel.

2.3.4 void tpu_mcpwm_slaveb_centermode_init

This function is used to initialize a channel to run as a MCPWM slave B center-aligned channel with either inverted or non-Inverted polarity. This function has 6 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 TPU channel number of the MCPWM slave B center-aligned channel. This parameter must be assigned a higher value than the channel number of the associated master channel. In CA mode, the slave A channel of a pair must be assigned a lower channel number than the slave B channel.
- priority - This is the priority to assign the channel. This parameter should be assigned the same value as the priority of the associated master channel.
- high_time - This is specified in TCR1 clocks over the range 0 to PERIOD, representing 0 to 100% duty cycle.
- polarity - This is the polarity of the slave B center-aligned channel. This parameter should be assigned a value of: INVERTED or NONINVERTED. This value should be set to the same polarity as that of the slave A channel they are referencing.
- slave_a_channel - This is the TPU channel number of the MCPWM slave A center-aligned channel that the slave B is referencing.

2.3.5 void tpu_mcpwm_update_hightime

This function is used to update the high time of a channel running the MCPWM channel for either EA or CA mode. This function has 4 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 TPU channel number of the MCPWM update hightime channel.
- high_time - This is specified in TCR1 clocks over the range 0 to PERIOD, representing 0 to 100% duty cycle.
- mode - This is the mode of the update hightime function. This parameter should be assigned a value of: EDGE or CENTER

Care should be taken when initializing TPU channels. The TPU's behavior may be unpredictable if a channel is reconfigured while it is running. The channels should be disabled before they are configured. TPU channels can be disabled by using the *tpu_disable* function in the mpc500_utils.c file. For example, disabling channel 0 is done like this: tpu_disable(tpu, 0); If the channel is currently being serviced when the priority is set to disable, the TPU will continue to service the channel until the state ends. To make sure the channel is not being serviced, you need to wait for the longest state execution time after disabling the channel. All channels are disabled out of reset so the channels can be configured immediately from reset.

The *tpu_mcpwm_init* functions attempt to wait between the disabling of the channels before they start configuring them, however the actual execution speed of the code will be depend on the specific system.

3 Multichannel PWM Examples

The following examples show configuration of the multichannel pulse-width modulation (MCPWM) function for both EA and CA modes. Each example is a C program that shows how to configure and use the MCPWM interface routines.

3.1 Example 1

3.1.1 Description

This sample program shows a simple MCPWM example in Edge-Aligned mode that generates two PWM signals with a period of 256 TCR1 clocks, one with a duty cycle of 50%, the other 25%. It will use channel 0 as the master and channels 1 and 2 as the slaves. It also generates a periodic interrupt every five PWM periods.

3.1.2 Program

```
/**************************************************************************/
/* FILE NAME: tpu_mcpwm_example1.c            COPYRIGHT (c)  2002 */
/* VERSION: 1.1                                    All Rights Reserved     */
/*                                                                        */
/* DESCRIPTION: This sample program shows a MCPWM example in Edge-Aligned */
/* mode that generates two PWM pulses with a period of 256 TCR1 clocks,   */
              duty cycle of 50%, the other 25%. It will use channel 0 as */
```

```
/* the master and channels 1 and 2 as the slaves. It also generates a    */
/* periodic interrupt every five PWM periods.                             */
/*                                                                        */
/* The program is targeted for the MPC555 but should work on any MPC500   */
/* device with a TPU. For other devices the setup routines will also need */
/* to be changed.                                                         */
/*========================================================================*/
/* HISTORY           ORIGINAL AUTHOR: Mario Ramiro Perez II               */
/*                                                                        */
/* REV      AUTHOR          DATE       DESCRIPTION OF CHANGE              */
/* ---   -----------     ---------    ---------------------               */
/* 1.0   Mario R. Perez  25 Sep 02    Initial version of function.        */
/**************************************************************************/

#include "mpc555.h"         /* Define MPC555 registers, this needs to be  */
                            /* changed if other MPC500 devices are used.  */
#include "mpc500_util.h"    /* Utility routines for using MPC500 devices  */
#include "tpu_mcpwm.h"      /* TPU MCPWM functions                        */



void main ()
{
           struct TPU3_tag *tpua = &TPU_A;        /* pointer for TPU routines */


           setup_555();



           /* Initialize MCPWM Master Init function with:  */
          /*     - TPU A Channel 0 set                    */
          /*     - Priority set to High                   */
          /*     - Period set to 0x100                    */
          /*     - IRQ Rate set to 0x5                    */
tpu_mcpwm_master_init(tpua, 0, TPU_PRIORITY_HIGH, 0x100, 0x5);



           /* Initialize Slave Edge Mode function with:    */
          /*     - TPU A Channel 1 set                    */
          /*     - Priority set to High                   */
          /*     - Period set to 0x100                    */
          /*     - High Time set to 0x80                  */
          /*    - High Time Pointer set to 0x12           */
          /*    - Master Channel is on TPU ch. 0          */
tpu_mcpwm_slave_edgemode_init (tpua, 1, TPU_PRIORITY_HIGH, 0x100, 0x80, 0x12, 0);


           /* Initialize Slave Edge Mode function with:    */
          /*    - TPU A Channel 2 set                     */
          /*    - Priority set to High                    */
          /*    - Period set to 0x100                     */
          /*    - High Time set to 0x40                   */
          /*    - High Time Pointer set to 0x22           */
          /*    - Master Channel is on TPU ch. 0          */
tpu_mcpwm_slave_edgemode_init (tpua, 2, TPU_PRIORITY_HIGH, 0x100, 0x40, 0x22, 0);


tpu_interrupt_enable(tpua, 0);          /* enable MCPWM interrupts */


          while (1) {                            /* loop and measure */
          }
}
```

3.1.3 Example 1 Schematic

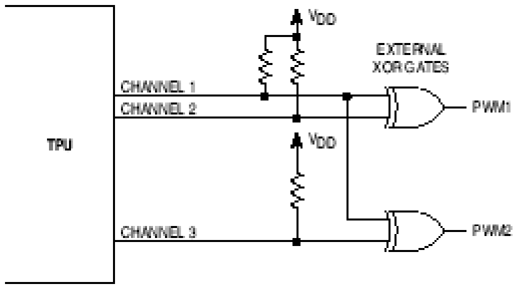

Figure 1. EA Mode Schematic

3.1.4 Example 1 Output Waveforms

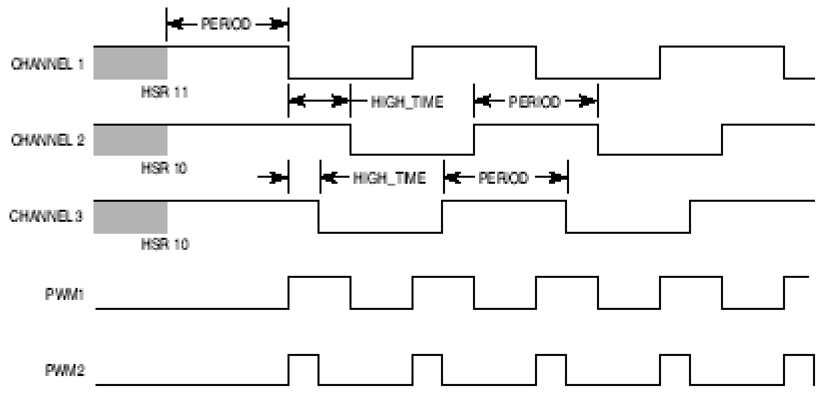

Figure 2. EA Mode Output Waveforms

3.2 Example 2

3.2.1 Description

This sample program shows a simple MCPWM example in Center-Aligned mode that generates two PWM signals with a period of 256 TCR1 clocks and a duty cycle of 50%. They share a common HIGH_TIME and are suitable for driving an inverter. The required DEAD_TIME = 4 and the master channel is on channel 0 while channels 1, 2, 3 and 4 are slaves. Periodic interrupts are generated every 10 PWM periods.

3.2.2 Program

```
/****************************************************************************/
/* FILE NAME: tpu_mcpwm_example2.c            COPYRIGHT (c)  2002   */
/* VERSION: 1.0                                    All Rights Reserved      */
/*                                                                          */
/* DESCRIPTION: This sample program shows a MCPWM example in center-aligned */
/* mode that generates two PWM pulses with a period of 256 TCR1 clocks,     */
/* sharing a common HIGH_TIME and suitable for driving an inverter. The    */
/* required DEAD_TIME = 4 and the master channel is on channel 0 while      */
/* channels 1, 2, 3 and 4 are slaves.  Periodic interrupts every 10 PWM     */
/* periods.  Initialize with a duty cycle of 50%.                           */
/*                                                                          */
/* The program is targeted for the MPC555 but should work on any MPC500     */
/* device with a TPU. For other devices the setup routines will also need   */
/* to be changed.                                                           */
/*======================================================================== */
/* HISTORY           ORIGINAL AUTHOR: Mario Ramiro Perez II                 */
/*                                                                          */
/* REV      AUTHOR          DATE       DESCRIPTION OF CHANGE                  */
/* ---   -----------     ---------    ---------------------                  */
/* 1.0   Mario R. Perez  25 Sep 02    Initial version of function.           */
/*****************************************************************************/

#include "mpc555.h"         /* Define MPC555 registers, this needs to be    */
                            /* changed if other MPC500 devices are used.    */
#include "mpc500_util.h"    /* Utility routines for using MPC500 devices    */
#include "tpu_mcpwm.h"      /* TPU MCPWM functions                          */


void main ()
{
    struct TPU3_tag *tpua = &TPU_A;         /* pointer for TPU routines     */

    setup_555();


    /* Initialize MCPWM Master Init function with:  */
    /*     - TPU A Channel 0 set                    */
    /*     - Priority set to High                   */
    /*     - Period set to 0x100                    */
    /*     - IRQ Rate set to 0xA                    */
tpu_mcpwm_master_init(tpua, 0, TPU_PRIORITY_HIGH, 0x100, 0xA);

    /* Initialize SlaveA Center Mode (Inverted):    */
    /*     - TPU A Channel 1 set                    */
    /*     - Priority set to High                   */
    /*     - Period set to 0x100                    */
    /*     - Dead Time set to 0x0                   */
    /*     - High Time Pointer set to 0x21          */
    /*     - Polarity is set to Inverted            */
    /*     - Master channel is running on TPU ch 0  */
tpu_mcpwm_slavea_centermode_init(tpua, 1, TPU_PRIORITY_HIGH, 0x100, 0x0, 0x21, INVERTED, 0);


    /* Initialize SlaveB Center Mode function with: */
    /*     - TPU A Channel 2 set                    */
    /*     - Priority set to High                   */
    /*     - High Time set to 0x80                  */
             larity is set to Inverted              */
    /*    - Slave A channel is running on TPU ch 1  */
tpu_mcpwm_slaveb_centermode_init(tpua, 2, TPU_PRIORITY_HIGH, 0x80, INVERTED, 1);



    /* Initialize SlaveA Center Mode (Non-Inverted):*/
    /*    - TPU A Channel 3 set                     */
    /*    - Priority set to High                    */
    /*    - Period set to 0x100                     */
    /*    - Dead Time set to 0x4                    */
    /*    - High Time Pointer set to 0x23           */
    /*    - Polarity is set to Non-Inverted         */
    /*    - Master channel is running on TPU ch 0   */
tpu_mcpwm_slavea_centermode_init(tpua, 3, TPU_PRIORITY_HIGH, 0x100, 0x4, 0x23, NONINVERTED, 0);



    /* Initialize SlaveB Center Mode function with: */
    /*    - TPU A Channel 4 set                     */
    /*    - Priority set to High                    */
    /*    - High Time set to 0 (don't care)         */
    /*    - Polarity is set to Non-Inverted         */
    /*    - Slave A channel is running on TPU ch 3  */
tpu_mcpwm_slaveb_centermode_init(tpua, 4, TPU_PRIORITY_HIGH, 0, NONINVERTED, 3);



tpu_interrupt_enable(tpua, 0);          /* enable MCPWM interrupts */

    while (1) {                             /* loop and measure */
    }
}
```

3.2.3 Example 2 Schematic

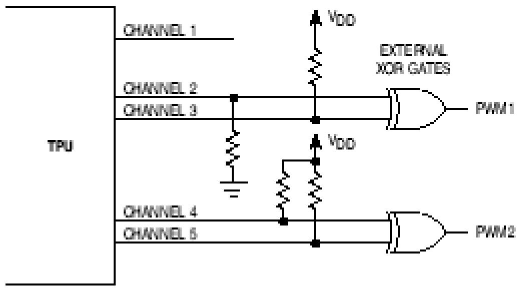

Figure 3. CA Mode Schematic

3.2.4 Example 2 Output Waveform

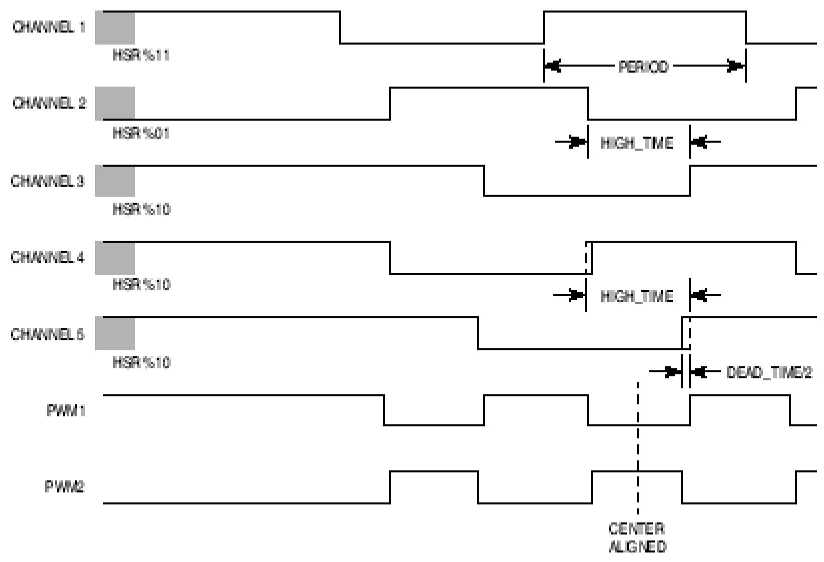

Figure 4. CA Mode Output Waveforms

4 Function State Timing

When calculating the worst case latency for the TPU, the execution time of each state of the TPU is needed. The state timings for each of the three modes of the SPWM function are shown below in Table 1.

Table 1. MCPWM Function State Timing

State Number & Name	Max. CPU Clock Cycles	RAM Accesses by TPU
S1 MINIT_MCPWM	20	6
S2 SINIT_MCPWM		
EA Mode slave	18	6
CA Mode slave A	38	8
CA Mode slave B	8	2
S3 S_INV_INIT_MCPWM	40	8
S4 MLH_MCPWM	18	6
S5 MHL_MCPWM	18	6
S6 SHL_MCPWM		
EA Mode slave	16	6
CA Mode slave A	36	8
CA Mode slave B	6	2
S7 SLH_MCPWM		
EA Mode slave	16	6
CA Mode slave A	36	8
CA Mode slave B	6	2

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

5 Function Code Size

Total TPU function code size determines what combination of functions can fit into a given ROM or emulation memory microcode space. MCPWM function code size is:

30µ instructions + 8 entries = 38 long words

6 Notes on Use and Performance of the MCPWM Function

6.1 Performance

Like all TPU functions, the performance limit of the MCPWM function in a given application depends to some extent on the activity on other TPU channels. This is due to the operation of the TPU scheduler.

Under steady state conditions, all MCPWM channels generate 50% duty cycle output waveforms (the phase delay between the signals generates the variable high time on the output of the EOR gates). This provides maximum service time for all transitions and lets the MCPWM function generate high and low duty cycle PWM consistently under a wide range of TPU operating conditions. Worst case service conditions for the MCPWM function occur when a large change in duty cycle is requested — in the extreme cases of a switch from 0% to 100% or 100% to 0%, the slave channels must produce a single pulse of half the normal period in order to establish the new timing relationship. Thus, in systems where these large duty cycle changes are possible, absolute worst-case timing analysis must be carried out using PERIOD divided by two as the maximum allowable service latency for each channel.

Taking this worst-case condition into account, and assuming a PWM PERIOD of 255 TCR1 clocks with the TCR1 prescaler set to divide by 4 (the fastest possible 8-bit PWM), the following is an approximate TPU loading formula for MCPWM generation:

$$\text{TPU loading (\%)} = 6 + (a \times 5.1) + (b \times 12.2)$$

where:

a = number of edge-aligned PWM

b = number of center-aligned PWM

The additional 6% is for the master channel.

This formula assumes a very low parameter RAM collision rate between the CPU and TPU, which is the normal operating condition. However, in applications where the CPU makes very frequent accesses to parameter RAM, these loading figures do not apply. In this case, and in applications where other TPU functions are running, detailed timing analysis of the TPU system is required.

Since the scheduler assures that the worst-case latencies in any TPU application can be closely estimated, a detailed timing analysis can be performed by following the guidelines given in the TPU reference manual. To perform an analysis, use the MCPWM state timing information in Table 1 above along with the state timing information for any other active TPU functions.

7 Usage Notes and Restrictions

7.1 Channel Assignment Restrictions

Due to coherency issues and the operation of the TPU scheduler, the following rules must be adhered to when assigning channels and priorities to ensure correct operation of the MCPWM function:

1. 1. All channels associated with a master channel must be assigned the same priority as that master.

2. 2. The master must be assigned the lowest channel number of all MCPWM channels. In CA mode, the slave A channel of a pair must be assigned a lower channel number than the slave B channel.
3. 3. When two CA mode pairs are used to form an inverter driver (the second pair has dead time and references the CURRENT_HIGH_TIME parameter of the first pair), the pair without dead time must be assigned lower channel numbers than the pair with dead time.

The following example shows application of these rules to a 3-phase H-bridge drive configuration using thirteen TPU channels (six CA mode PWM plus master).

Table 2 Example Multichannel PWM Function

Channel Number	Operation Mode	Function
0	Master	Reference timing channel for all slaves
1	Slave A	PWM1
2	Slave B	PWM1
3	Slave A	PWM1' — as PWM1 but with dead time
4	Slave B	PWM1' — as PWM1 but with dead time
5	Slave A	PWM2
6	Slave B	PWM2
7	Slave A	PWM2' — as PWM2 but with dead time
8	Slave B	PWM2' — as PWM2 but with dead time
9	Slave A	PWM3
10	Slave B	PWM3
11	Slave A	PWM3' — as PWM3 but with dead time
12	Slave B	PWM3' — as PWM3 but with dead time

7.2 Resolution/Frequency Relationship

Unlike most hardware PWM peripherals, the resolution and frequency of the MCPWM function is not fixed and can be tailored to meet application requirements. In general, the two qualities are inversely proportional, i.e., the higher the resolution, the lower the frequency. Also, since PERIOD is freely programmable in TCR1 clocks, the number of bits of resolution does not have to be an integer number. The user could program PERIOD to be 0x180, to produce an' 8.5 bit ' PWM with 384 resolution states. With a 40 MHz system clock, the minimum TCR1 clock period is 100 ns and the resulting frequency of the PWM output is given by 1 § (PERIOD*100 ns). This gives frequencies of approximately 40 kHz, 20 kHz, and 10 kHz for 8-, 9-, and 10-bit resolution PWM respectively.

7.3 Hardware Requirements

The MCPWM function requires one external exclusive OR gate for each PWM output. In addition, it is recommended that pull-up resistors be added to all MCPWM channels except for slave A channels of inverted PWM, which should have pull down resistors. The resistors insure a predictable output from the EOR gates during the time between power up or reset (when all TPU channel pins are in a high- impedance state) and initialization of the MCPWM function. In some circumstances, an external buffer with three-state capability may be required.

7.4 Initialization Timing

The following timing assumes initialization from reset.

In edge-aligned mode, the first rising edge of the PWM outputs from the EOR gates occurs approximately one PWM period after completion of the API call for the master channel initialization. Figure 5 shows these relationships.

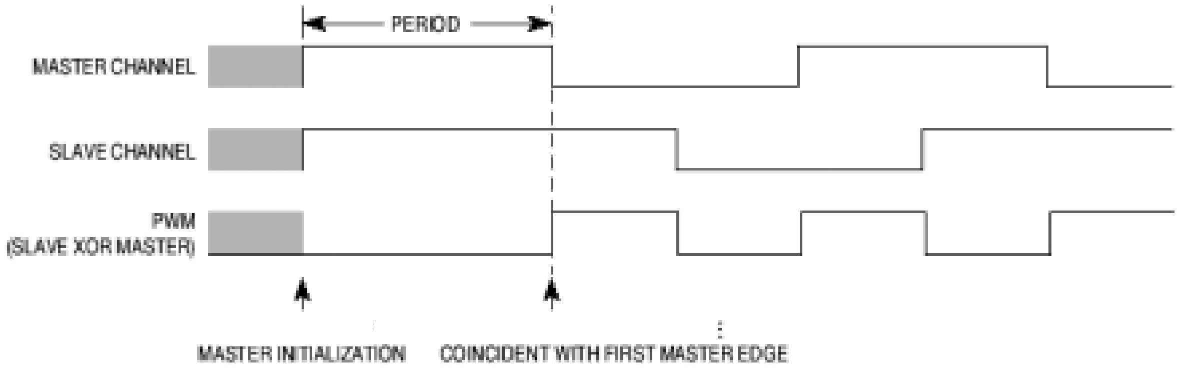

Figure 5. EA Mode Initialization Timing

In center-aligned mode, the first rising edge occurs between one and one-and-a-half PWM periods after completion of the API call for the master channel initialization, depending on the programmed duty cycle. Figure 6 shows these relationships.

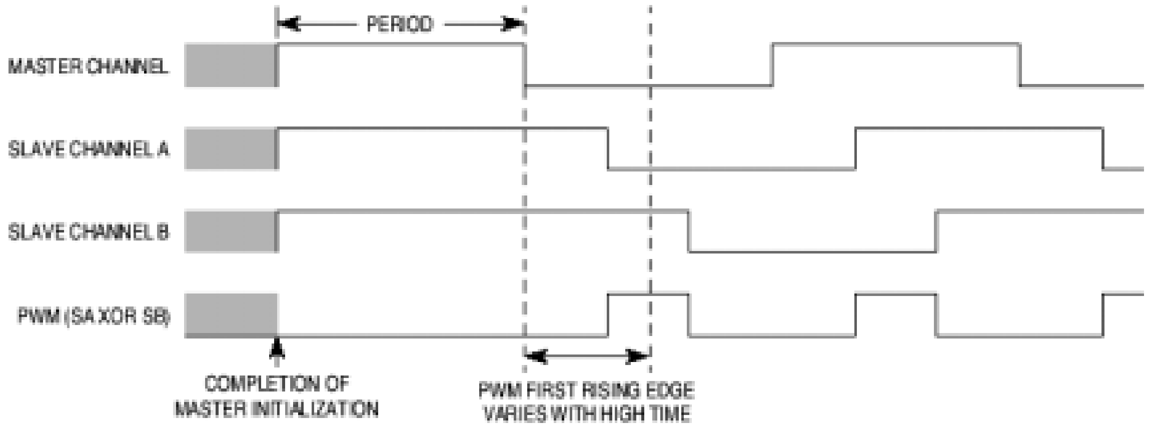
PERIOD
MASTER CHANNEL
SLAVE CHANNEL A
SLAVE CHANNEL B
PWM (SA XOR SB)
COMPLETION OF MASTER INITIALIZATION
PWM FIRST RISING EDGE VARIES WITH HIGH TIME

Figure 6. CA Mode Initialization Timing

7.5 High Time Update Timing

In both edge-aligned and center-aligned modes, when a new HIGH_TIME value is written by the CPU, it takes effect either at the beginning of the next period or the beginning of the period following that, depending on the exact timing of the write as explained below. There is no provision for updating the high time of the period in progress.

When a new high time value is written in edge-aligned mode, if the write takes place before the falling edge of the PWM, the new high time value is used in the next period. If the write takes place after the falling edge of the PWM, the new value is used no later than the second period after the write.

When a new high time value is written in center-aligned mode, if the write takes place before the rising edge of the PWM (non-inverted), the new high time value is used in the next period. If the write takes place after the rising edge, the new value is used no later than the second period after the write.

7.6 Coherent High Time Updates

There are two possible cases when a HIGH_TIME parameter is updated asynchronously in relation to PWM output waveform timing and TPU channel servicing. Using the periodic master channel interrupt does not guarantee either case, since the master edge that results in an interrupt is coincident with the start of the PWM period — a CPU write in response to the interrupt could result in either case depending on the current high time and CPU and TPU latencies.

There may be applications where this high time uncertainty is unacceptable. For example, an application may demand that multiple PWM outputs be updated in the same period. MCPWM can support such a requirement in center-aligned mode by using a second master channel solely for periodic interrupt generation. The second master channel runs at twice the speed of the master channel used for actual PWM generation.

To support this capability, set up the second master channel as follows:

1. Second master channel number > last PWM slave channel number
2. PERIOD = (PWM period)/2
3. Interrupts enabled
4. IRQ_RATE = (desired number of PWM periods between interrupts) $*$ 2
5. PERIOD_COUNT initialized to 0x00
6. Priority same as other MCPWM channels

The initialization API call for the second master channel should be made at the same time as that for other MCPWM channels. Channel order is important because the second master channel must be serviced last when multiple MCPWM channels request service simultaneously (see details of scheduler operation in the TPU reference manual). This scheme causes an interrupt request to be made to the CPU in the second half of the PWM period, close to the period mid-point but after other pending MCPWM channels have been serviced. The CPU has the remainder of the PWM period in which to update HIGH_TIME parameters before the next PWM rising edge, thus guaranteeing use of a new high time in the following cycle. Figure 7 shows an example of this technique where the high times of two PWMs are coherently updated every three periods (IRQ_RATE of second master = 6).

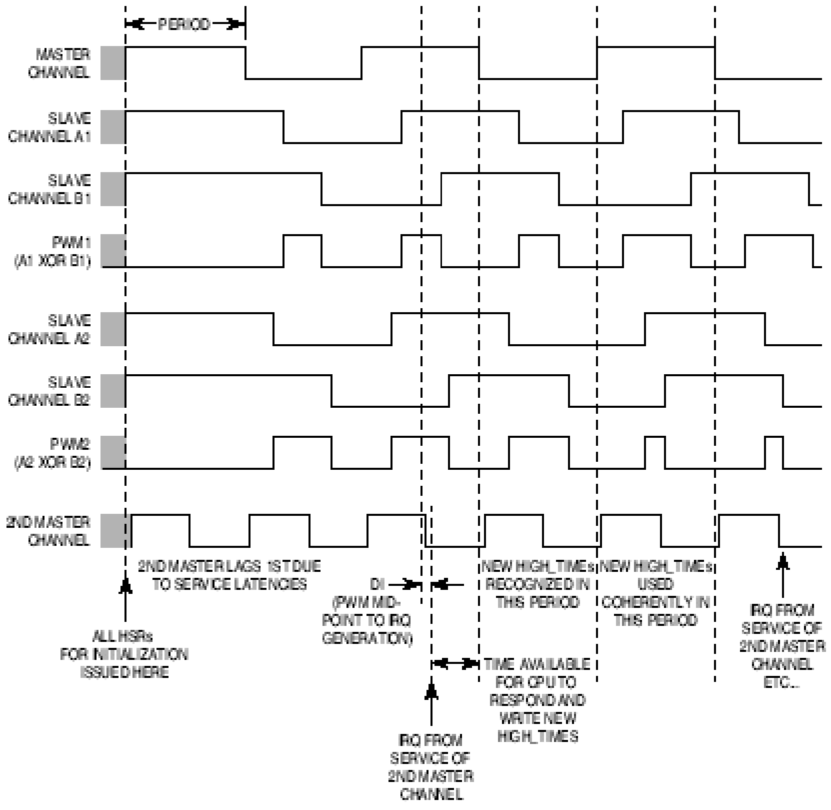
PERIOD
MASTER CHANNEL
SLAVE CHANNEL A1
SLAVE CHANNEL B1
PWM1 (A1 XOR B1)
SLAVE CHANNEL A2
SLAVE CHANNEL B2
PWM2 (A2 XOR B2)
2ND MASTER CHANNEL
2ND MASTER LAGS 1ST DUE TO SERVICE LATENCIES
ALL HSRs FOR INITIALIZATION ISSUED HERE
DI (PWM MID-POINT TO IRQ GENERATION)
IRQ FROM SERVICE OF 2ND MASTER CHANNEL
NEW HIGH_TIMEs RECOGNIZED IN THIS PERIOD
TIME AVAILABLE FOR CPU TO RESPOND AND WRITE NEW HIGH_TIMES
NEW HIGH_TIMEs USED COHERENTLY IN THIS PERIOD
IRQ FROM SERVICE OF 2ND MASTER CHANNEL ETC...

Figure 7. HIGH_TIME Update Timing

Worst-case delay (DI) from the period mid-point to interrupt generation using this technique occurs when all PWM are set to 0% (all slave channels request service at PWM mid-point). DI can be closely estimated:

DI = Total Worst Case Service Time (WCST) of all MCPWM slave channels +
Worst case service time of second master channel.

For example, if MCPWM is used to generate two centered-aligned PWM outputs (four slaves), with a second master channel for coherent high time updates and a time slot transition time of 10 CPU clocks for each channel serviced, then:

DI = WCSTA1 + WCSTA2 + WCSTB1 + WCSTB2 + WCSTM2 =
46 + 46 + 16 + 16 + 28 = 152 CPU Clocks

The CPU has the remainder of the PWM period to respond to the interrupt and update the high times.

7.6.1 Effects of DEAD_TIME

When DEAD_TIME has a non-zero value, the programmed HIGH_TIME is reduced by an amount equal to DEAD_TIME TCR1 clocks. This means that the minimum HIGH_TIME value that can cause a nonzero PWM duty cycle is (DEAD_TIME + 1). Similarly, at the upper end of the scale, 100% duty cycle is not possible with a non-zero DEAD_TIME value. If HIGH_TIME is programmed to equal PERIOD, the output has a non-zero low time equal to DEAD_TIME.

7.6.2 Odd PERIOD/HIGH_TIME/DEAD_TIME Values in CA Mode

In center-aligned mode, the MCPWM function attempts to center ‘real’ high time (HIGH_TIME – DEAD_TIME) within the PWM period. Obviously, with an even PERIOD and an odd high time or with an odd PERIOD and an even high time, exact centering is not possible. In these two cases, full resolution is maintained by making the portion of the high time before period mid-point one TCR1 clock longer than the portion following the period mid-point.

7.7 Changing Period

The MCPWM function is designed primarily for applications where the period of the PWM output is not variable. Since the PWM output is the result of the combination of two independently-running channels, when the CPU changes the PERIOD parameters, one channel must use the new period before the other.

This results in temporary loss of synchronization and indeterminate output from the EOR gate. When HIGH_TIME is less than or equal to the new PERIOD, the function returns to normal operation after a maximum of two times the original PERIOD value.

7.8 Stopping the Function

The MCPWM function can be disabled by using the *tpu_disable* function in the mpc500_utils.c file. For example. disabling channel 0 is done like this: tpu_disable (tpu, 0). However, this procedure leaves EOR gate output at an indeterminate level, even when HIGH_TIME is programmed to 0 or 100% before the bits are cleared. This is due to the phase relationship between the two channels forming the PWM output and to independent servicing of each channel. In most applications the best method of switching off the PWM is to program HIGH_TIME to 0 or 100% and leave the function running. When the function must be stopped cleanly there are two solutions:

1. Place an external buffer with three-state capability on the output of the EOR gate and control it with an MCU output pin.
2. Reset the MCU — this returns all TPU pins to their original high-impedance condition.

7.9 Interrupts

Interrupt service requests are generated by the MCPWM master channel every IRQ_RATE periods. Interrupts are enabled or disabled by using the *tpu_interrupt_enable* or *tpu_interrupt_disable* functions in the mpc500_utils.c file. The slave A channels can also generate interrupt requests, but these are meaningless to the user.

Since interrupt request generation is performed by microcode software when the relevant master channel edge is serviced, there is a delay between when the master channel edge transition occurs and the interrupt request signal is asserted. This delay is variable — it depends on the service latency of the master channel in a particular application. The time between successive interrupt requests can vary and does not exactly equal the IRQ_RATE periods. Worst-case variation (WCV) about IRQ_RATE periods are derived as follows:

WCV = (Worst case master channel service latency) – (Best case master channel service latency)

It follows that there is a spread of interrupt periods from

{(IRQ_RATE ∗ PERIOD) – WCV} to {(IRQ_RATE ∗ PERIOD) + WCV}.

7.10 Combining EA and CA Modes

Since master channel operation is identical for both edge-aligned and center-aligned modes, it is possible to generate both types of PWM in a system using only one master channel. However, the periods of all PWMs that refer to a common master must be identical.

7.11 Using MCPWM as a Periodic Timer

The periodic update interrupt feature of the MCPWM function makes it suitable for use as a periodic interrupt timer (PIT) for the CPU. To use the function as a PIT, only one channel, programmed as a master, is required. When no slave channels are referencing the master channel, the maximum PERIOD parameter value can be increased to 0x8000 TCR1 clocks. The combination of PERIOD and the 8-bit IRQ_RATE parameters allows a wide variety of interrupt periods to be programmed. The maximum interrupt period can be greater than one minute; the minimum period is obtained by programming IRQ_RATE to one and setting PERIOD to a minimum value determined from the state timing table and other TPU system activity.

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

AN2375/D