Application Note

AN2360/D
Rev. 0, 10/2002

General TPU C Functions for the MPC500 Family

Randy Dees with Jeff Loeliger
TECD Applications

This application note shows examples of C level API routines to access the MPC500-family TPU registers. These routines can be used to access either ROM functions or functions stored in the DPTRAM. Examples of the use of this API are shown in the Freescale application notes shown in Table 6. All of these functions and definitions are included in the mpc500_util.c and mpc500_util.h files. These are included in the standard Freescale C header files for the MPC500 family (version 3.0.3). The header files are available on the Freescale web site.

1 Functional Overview

The MPC555 and the MPC56x devices include Timing Processor Units (TPU). The TPU included on these devices is actually the third version of the TPU, also known as TPU3. The TPU is an autonomous processor with its own memory and program control. The MPC500 device controls the startup of the TPU and can communicate to the TPU through a shared memory space (parameter RAM). The TPU has 16 separate channels, each with parameter RAM. Most of the members of the MPC500 family have 2 TPUs. This application note covers the CPU to TPU interface and some basic C routines that can be used to interface to functions programmed into the TPU channels.

The TPU can execute from either a built-in TPU ROM or from SRAM that is external to the TPU (DPTRAM). The DPTRAM is a Dual Port TPU RAM that can be accessed by the main processor or by up to two TPUs. The DPTRAM must be loaded by the main processor, but it cannot be written or read by the main CPU once the EMU bit has been set in either TPUMCR. The TPU has two limitations: it has a limited amount of address space (a total of 8K, composed of 2K banks), and it can only run from either the TPU ROM or the DPTRAM. The TPU cannot access both the ROM and the DPTRAM at the same time, and it can execute only from one or the other. The TPU ROM has a total of 4K of ROM, composed of two 2K banks. The DPTRAM on the main TPUs on the MPC555 and the MPC565 has a total of 6K of memory, composed of three 2K banks. The MPC561 and MPC563 have 8K of DPTRAM (four 2K banks). Additionally, the MPC565 has a third TPU that has it's own 4K of DPTRAM (2 banks).

The MPC500 processor sets the default TPU instruction memory source and bank in the TPUMCR register.

Each bank of the TPU instruction memory (DPTRAM or ROM) can have an entry table that points to functions that are stored in memory and each entry location is associated to a function number. Functions are assigned to channels in the channel function select register. Each channel can be assigned any function. The entry table has space for 16 functions that are

referenced by its entry table function location. Each of the 16 functions has16 sub-entries. Four of these entry points are selectable by the main processor by setting the Host Sequence Register. Typically, these consist of an initialization routine, a “no host service,” and sometimes a change or update parameter function.

Each TPU channel must also be assigned a priority. The priority determines how often that channel’s state will be executed.

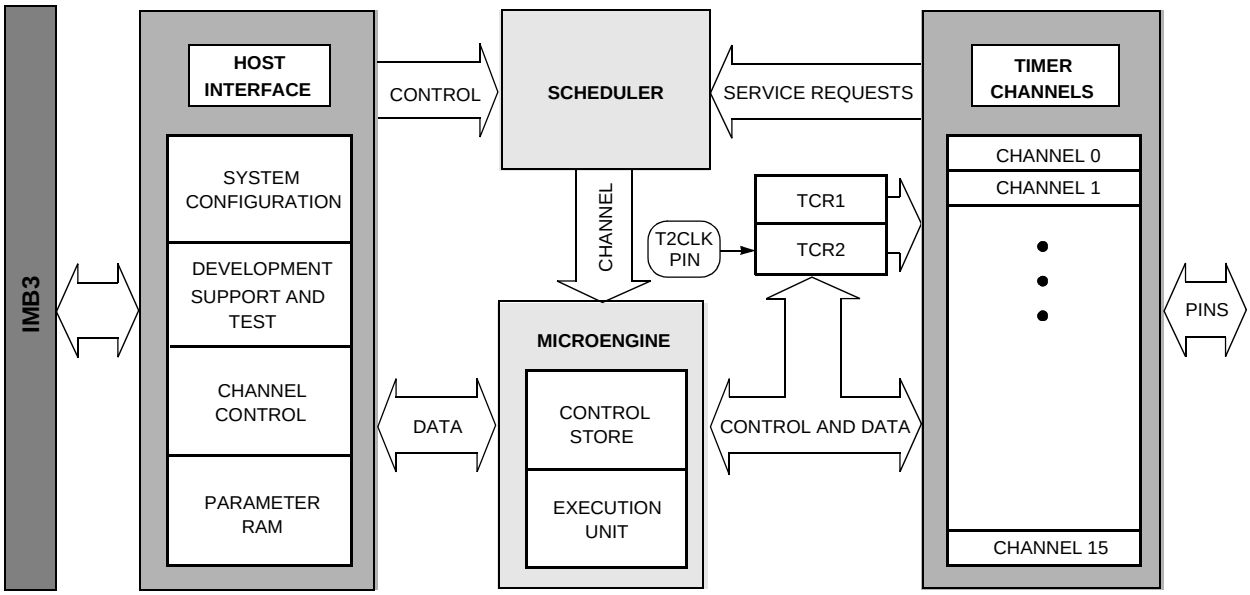

Figure 1. TPU3 Block Diagram

2 TPU Initialization Flow

1. Load TPU microcode into DPTRAM if the TPU ROM functions are not being used.
2. Initialize the TPU module registers (TPUMCR, TPUMCR2, TPUMCR3) to set the TCR1 and TCR2 clock prescalers; set the emulation (EMUL) if DPTRAM will be used. Also, the Bank should be set in the TPUMCR2 register.
3. Set the TPU3 interrupt configuration register (TICR) interrupt level. Program CIRL and ILBS to the level per Table 1.

Table 1. TPU Interrupt Level Settings

Interrupt Level	CIRL	ILBS
0-7	0-7	0b00
8-15	0-7	0b01
16-23	0-7	0b10
24-31	0-7	0b11

4. Stop all TPU channels that will be changing functions.
5. Wait for any pending HSR on the desired channel to complete. (The HSR bits are set back to 0b00 when the Host Service Routine complete.) The macro tpu_ready waits for the HSR bits to be cleared.

6. Set the TPU function for each channel in the CFSR[0:3] registers. The functions tpu_func() and get_tpu_func() can be used to access the channel function select registers.
7. Set the host sequence register (HSQR[0:1]) for each channel. The functions tpu_hsq() and get_tpu_hsq() can be used to access the host sequence registers.
8. Set the host service registers (HSRR[0:1]) for each channel. The functions tpu_hsr() and get_tpu_hsr() can be used to access the host service registers.
9. Enable interrupts for the channels that will cause interrupts to the main processor (CIER). The functions tpu_interrupt_enable() and tpu_interrupt_disable() can be used to access the channel interrupt enable register.
10. Initialize the channel's parameter RAM.
11. Set the channel priority registers for all of the channels of the TPU that will be run (CPR[0:1]). The functions tpu_enable() and tpu_disable() can be used to access the channel priority registers. See Table 5 for more information.

In addition, there are additional functions to check if the desired channel caused a TPU interrupt or can clear the interrupt.

By using the C API, this the user does not need to remember the addresses of all of the TPU registers and bit positions.

3 General TPU Routines C Level API

This section describes the Application Programming Interface (API) to some generally useful routines that can be used to interface to TPU functions. These functions are divided into functions that are used to initialize the TPU, get the status of the TPU, and return the status of the TPU.

Additional notes for the following API calls:

- *tpu can be a pointer to any of the TPU module registers on a device (either A, B, or C)
- channel is a valid channel number 0 to 15.

3.1 Initialization Functions:

Most initialization routines are specific to the specific to the TPU function, but there are some general routines for accessing the TPU itself.

- void tpu_func(struct TPU3_tag *tpu, UINT8 channel, UINT8 function_number);
- void tpu_hsr(struct TPU3_tag *tpu, UINT8 channel, UINT8 hsr);
- void tpu_hsq(struct TPU3_tag *tpu, UINT8 channel, UINT8 hsq);
- 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);

3.2 Status Operation Functions:

- UINT8 tpu_get_func(struct TPU3_tag *tpu, UINT8 channel);
- UINT8 tpu_get_hsr(struct TPU3_tag *tpu, UINT8 channel);

- UINT8 tpu_get_hsq(struct TPU3_tag *tpu, UINT8 channel);
- tpu_ready(tpu, channel);
- UINT8 tpu_check_interrupt(struct TPU3_tag *tpu, UINT8 channel)

3.3 General TPU Initialization Functions

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

3.3.1 Assign TPU Function (void tpu_func)

This function assigns a TPU function to a particular channel.

- *tpu - This is a pointer to the TPU3 module. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the function assigned to it.
- function - This is the desired function for the channel. See Table 4 for the possible values for the built-in ROM functions.

3.3.2 Assign Host Service Request Register Function (void tpu_hsr)

This function assigns the host service request register for a particular channel. The meaning of the host service request bits depends on the function specified.

- *tpu - This is a pointer to the TPU3 module. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the function assigned to it.
- hsr - This is the desired host service request for the channel. See the documentation for the individual function for the definitions for the host service routine.

3.3.3 Assign Host Sequence Register Function (void tpu_hsq)

This function assigns the host sequence (HSQ) field for the channel. The HSQ selects the mode of operation for the function selected on a given channel. The meaning of the host sequence bits depends on the function specified.

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

3.3.4 TPU Enable Function (void tpu_enable)

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

- *tpu - This is a pointer to the TPU3 module. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the function assigned to it.
- priority - This is the new channel priority. See Table 5 for possible definitions for the priority levels.

3.3.5 TPU Disable Function (void tpu_disable)

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. It is of type TPU3_tag which is defined in m_tpu3.h.
- channel - This is the channel number that has the function assigned to it.

3.3.6 TPU Interrupt Enable Function (void tpu_interrupt_enable)

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 function assigned to it.

3.3.7 TPU Interrupt Disable Function (void tpu_interrupt_disable)

This function disables the interrupt bit for the specified channel.

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

3.3.8 TPU Clear Interrupt Function (void tpu_clear_interrupt)

This function clears the interrupt bit for the specified channel.

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

3.4 Status Functions

The following routines get the current function or status of a TPU channel.

3.4.1 TPU Get Function (UINT8 tpu_get_func)

This function returns an 8-bit value of the function number currently running on 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 function assigned to it.

3.4.2 Get Host Service Request Register Function (UINT8 tpu_get_hsr)

This function gets the current value of the Host Service Request Register for the given channel. It returns an 8-bit value.

- *tpu - This is a pointer to the TPU3 module. It is of type TPU3_tag which is defined in m_tpu3.h.

- channel - This is the channel number that has the function assigned to it.

3.4.3 Get Host Sequence Register Function (UINT8 tpu_get_hsq)

This function gets the current value of the Host Sequence Register for a particular channel. It returns an 8-bit value.

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

3.4.4 Wait for TPU Channel Ready (tpu_ready)

This macro waits for a channel's HSR function to complete.

- *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 function assigned to it.

3.4.5 TPU Check Interrupt Function (UINT8 tpu_check_interrupt)

This function checks the interrupt bit for the specified channel to see if it is set. This function returns 0b1 if this channel caused the interrupt, 0b0 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 function assigned to it.

4 TPU Parameter RAM

Table 2 shows TPU parameter RAM addresses for each channel for each of the possible TPU modules used by the MPC500 devices. In the table, yy is equal to 0x41 for TPU_A, 0x45 for TPU_B, and 0x5d for TPU_C (MPC565 only).

Table 2. TPU Function Parameter Addresses (16-bit)[1]

	Parameter Address							
Channel Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0x30yy00	0x30yy02	0x30yy04	0x30yy06	0x30yy08	0x30yy0A	0x30yy0C	0x30yy0E
1	0x30yy10	0x30yy12	0x30yy14	0x30yy16	0x30yy18	0x30yy1A	0x30yy1C	0x30yy1E
2	0x30yy20	0x30yy22	0x30yy24	0x30yy26	0x30yy28	0x30yy2A	0x30yy2C	0x30yy2E
3	0x30yy30	0x30yy32	0x30yy34	0x30yy36	0x30yy38	0x30yy3A	0x30yy3C	0x30yy3E
4	0x30yy40	0x30yy42	0x30yy44	0x30yy46	0x30yy48	0x30yy4A	0x30yy4C	0x30yy4E
5	0x30yy50	0x30yy52	0x30yy54	0x30yy56	0x30yy58	0x30yy5A	0x30yy5C	0x30yy5E
6	0x30yy60	0x30yy62	0x30yy64	0x30yy66	0x30yy68	0x30yy6A	0x30yy6C	0x30yy6E
7	0x30yy70	0x30yy72	0x30yy74	0x30yy76	0x30yy78	0x30yy7A	0x30yy7C	0x30yy7E
8	0x30yy80	0x30yy82	0x30yy84	0x30yy86	0x30yy88	0x30yy8A	0x30yy8C	0x30yy8E
9	0x30yy90	0x30yy92	0x30yy94	0x30yy96	0x30yy98	0x30yy9A	0x30yy9C	0x30yy9E
10 (0xA)	0x30yyA0	0x30yyA2	0x30yyA4	0x30yyA6	0x30yyA8	0x30yyAA	0x30yyAC	0x30yyAE
11 (0xB)	0x30yyB0	0x30yyB2	0x30yyB4	0x30yyB6	0x30yyB8	0x30yyBA	0x30yyBC	0x30yyBE
12 (0xC)	0x30yyC0	0x30yyC2	0x30yyC4	0x30yyC6	0x30yyC8	0x30yyCA	0x30yyCC	0x30yyCE
13 (0xD)	0x30yyD0	0x30yyD2	0x30yyD4	0x30yyD6	0x30yyD8	0x30yyDA	0x30yyDC	0x30yyDE
14 (0xE)	0x30yyE0	0x30yyE2	0x30yyE4	0x30yyE6	0x30yyE8	0x30yyEA	0x30yyEC	0x30yyEE
15 (0xF)	0x30yyF0	0x30yyF2	0x30yyF4	0x30yyF6	0x30yyF8	0x30yyFA	0x30yyFC	0x30yyFE

Table 3. TPU Function Parameter Addresses (32-bit) [1]

	Parameter Address			
Channel Number	**0**	**1**	**2**	**3**
0	0x30yy00	0x30yy04	0x30yy08	0x30yy0C
1	0x30yy10	0x30yy14	0x30yy18	0x30yy1C
2	0x30yy20	0x30yy24	0x30yy28	0x30yy2C
3	0x30yy30	0x30yy34	0x30yy38	0x30yy3C
4	0x30yy40	0x30yy44	0x30yy48	0x30yy4C
5	0x30yy50	0x30yy54	0x30yy58	0x30yy5C
6	0x30yy60	0x30yy64	0x30yy68	0x30yy6C
7	0x30yy70	0x30yy74	0x30yy78	0x30yy7C
8	0x30yy80	0x30yy84	0x30yy88	0x30yy8C
9	0x30yy90	0x30yy94	0x30yy98	0x30yy9C
10 (0xA)	0x30yyA0	0x30yyA4	0x30yyA8	0x30yyAC
11 (0xB)	0x30yyB0	0x30yyB4	0x30yyB8	0x30yyBC
12 (0xC)	0x30yyC0	0x30yyC4	0x30yyC8	0x30yyCC
13 (0xD)	0x30yyD0	0x30yyD4	0x30yyD8	0x30yyDC
14 (0xE)	0x30yyE0	0x30yyE4	0x30yyE8	0x30yyEC
15 (0xF)	0x30yyF0	0x30yyF4	0x30yyF8	0x30yyFC

[1] yy = 0x41 for TPU_A,
yy = 0x45 for TPU_B, and
yy = 0x5D for TPU_C (MPC565 only)

The parameter RAM provides a means of passing arguments from the main processor to the TPU function. Each function defines the use of each of the parameter RAM locations. Typically, each function includes a parameter RAM map similar to the one shown in Figure 2. Some of these parameter locations can be defined to be written by the CPU, by the TPU channel, or by both the CPU and the TPU channel. They can also be unused by the function.

When using the standard MPC500 header files, the TPU Parameter RAM can be set using the following C statements. The first sets a 16 bit parameter.

```
tpu->PARM.R[channel][parameter_number] = new_value16;
```

The second sets a 32 bit parameter.

```
tpu->PARM.L[channel][parameter_number] = new_value32;
```

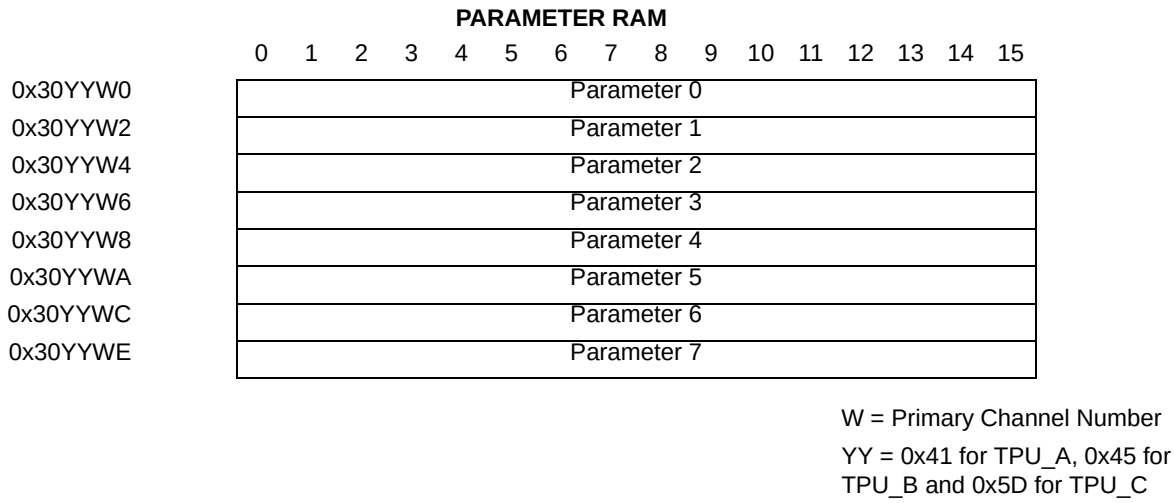

Figure 2. Parameter RAM Layout

5 Host Interface to TPU Function

This section provides information about how the CPU interfaces to the TPU though a set of registers that exist in the CPU address space.

5.1 Channel Function Select Registers

Each channel of the TPU can have any function assigned to it. The Channel Function Select Register is used to select the function for each of the channels. Table 4 lists the functions that are defined in the TPU3 ROM used in the MPC500 devices. Two routines are provided to read and write a channels Channel Select Function. *tpu_func* assigns the function to a channel and *tpu_get_func* returns the function currently assigned to a channel.

CFSR0 — Channel Function Select Register 0

0x30 400C
0x30 440C
0x30 5C0C

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 15				CH 14				CH 13				CH 12			

CFSR1 — Channel Function Select Register 1

0x30 400E
0x30 440E
0x30 5C0E

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 11				CH 10				CH 9				CH 8			

CFSR2 — Channel Function Select Register 2

0x30 4010
0x30 4410
0x30 5C10

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 7				CH 6				CH 5				CH 4			

CFSR3 — Channel Function Select Register 3

0x30 4012
0x30 4412
0x30 5C12

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 3				CH 2				CH 1				CH 0			

Table 4. TPU ROM Functions

Function Number	Bank 0		Bank 1	
	Function	Function Name	Function	Function Name
0xF	TPU_FUNCTION_PTA	Programmable Time Accumulator	TPU_FUNCTION_PTA	Programmable Time Accumulator
0xE	TPU_FUNCTION_QOM	Queued Output Match	TPU_FUNCTION_QOM	Queued Output Match
0xD	TPU_FUNCTION_TSM	Table Stepper Motor	TPU_FUNCTION_TSM	Table Stepper Motor
0xC	TPU_FUNCTION_FQM	Frequency Measurement	TPU_FUNCTION_FQM	Frequency Measurement
0xB	TPU_FUNCTION_UART	Universal Asynchronous Receiver/Transmitter	TPU_FUNCTION_UART	Universal Asynchronous Receiver/Transmitter
0xA	TPU_FUNCTION_NITC	New Input Capture/ Input Transition Counter	TPU_FUNCTION_NITC	New Input Capture/ Input Transition Counter

Table 4. TPU ROM Functions (continued)

Function Number	Bank 0		Bank 1	
	Function	Function Name	Function	Function Name
0x9	TPU_FUNCTION_COMM	Multiphase Motor Commutation	TPU_FUNCTION_COMM	Multiphase Motor Commutation
0x8	TPU_FUNCTION_HALLD	Hall Effect Decode	TPU_FUNCTION_HALLD	Hall Effect Decode
0x7	TPU_FUNCTION_MCPWM	Multi-Channel Pulse Width Modulation	—	Reserved
0x6	TPU_FUNCTION_FQD	Fast Quadrature Decode	TPU_FUNCTION_FQD	Fast Quadrature Decode
0x5	TPU_FUNCTION_PPWA	Period/Pulse Width Accumulator	TPU_FUNCTION_ID	Identification
0x4	TPU_FUNCTION_OC	Output Compare	TPU_FUNCTION_OC	Output Compare
0x3	TPU_FUNCTION_PWM	Pulse Width Modulation	TPU_FUNCTION_PWM	Pulse Width Modulation
0x2	TPU_FUNCTION_DIO	Discrete Input/Output	TPU_FUNCTION_DIO	Discrete Input/Output
0x1	TPU_FUNCTION_SPWM	Synchronized Pulse Width Modulation	TPU_FUNCTION_RWTPIN	Read/Write Timers and Pin
0x0	TPU_FUNCTION_SIOP	Serial Input/Output Port	TPU_FUNCTION_SIOP	Serial Input/Output Port

5.1.1 Assign TPU Function

```
/*********************************************************************
FUNCTION      : tpu_func
PURPOSE       : To assign a function to a given channel
INPUTS NOTES  : This function has 3 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 number of the channel
          function_number - Function number to be assigned
*********************************************************************/

void tpu_func(struct TPU3_tag *tpu, UINT8 channel, UINT8 function_number)

{

    UINT16 reg;


    if (channel < 4) {

        reg = tpu->CFSR3.R;

        reg &= ~(TPU_CHANNEL_MASK << (channel * 4));

        reg |= (function_number << (channel * 4));

        tpu->CFSR3.R = reg;

    }

    else if (channel < 8) {

        reg = tpu->CFSR2.R;

        reg &= ~(TPU_CHANNEL_MASK << ((channel-4) * 4));

        reg |= (function_number << ((channel-4) * 4));

        tpu->CFSR2.R = reg;

    }
```

```
    else if (channel < 12) {
        reg = tpu->CFSR1.R;
        reg &= ~(TPU_CHANNEL_MASK << ((channel-8) * 4));
        reg |= (function_number << ((channel-8) * 4));
        tpu->CFSR1.R = reg;
    }
    else {
        reg = tpu->CFSR0.R;
        reg &= ~(TPU_CHANNEL_MASK << ((channel-12) * 4));
        reg |= (function_number << ((channel-12) * 4));
        tpu->CFSR0.R = reg;
    }
}
```

5.1.2 TPU Get Function

```
/*********************************************************************
FUNCTION      : tpu_get_func
PURPOSE       : To get the function number running on a given channel
INPUTS NOTES  : This function has 2 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 number of the channel
RETURNS NOTES: This function will return the function number running on
          the channel.
*********************************************************************/
UINT8 tpu_get_func(struct TPU3_tag *tpu, UINT8 channel)
{
    UINT16 function;

    if (channel < 4) {
        function = tpu->CFSR3.R;
    }
    else if (channel < 8) {
        channel -= 4;
        function = tpu->CFSR2.R;
    }
    else if (channel < 12) {
        channel -= 8;
        function = tpu->CFSR1.R;
    }
    else {
        channel -= 12;
        function = tpu->CFSR0.R;
```

```
    }

    function=(function & (TPU_CHANNEL_MASK << (channel * 4)) >> (channel * 4));

    return ((UINT8)function);
}
```

5.2 Host Service Request Registers

The host service request field selects the type of host service request for the function selected on a given channel. The meaning of the host service request bits is determined by function microcode. Three routines are provided to read and write a channel's host service register *tpu_hsr* assigns the host service request to a channel. *tpu_get_hsr* will return the host service request setting currently assigned to a channel or zero if the current state has completed. A third macro is provided that waits for a host service routine to complete.

WARNING

Only the TPU can clear the HSR bits for a channel. If the HSR bits are written without waiting for the TPU to clear them, the requests will be ORed together. The host CPU can only set bits in the HSR registers so the values can be written directly with masks.

HSRR0— Host Service Request Register 0 **0x30 4018 0x30 4418 0x30 5C18**

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 15		CH 14		CH 13		CH 12		CH 11		CH 10		CH 9		CH 8	

HSRR1— Host Service Request Register 1 **0x30 401A 0x30 441A 0x30 5C1A**

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 7		CH 6		CH 5		CH 4		CH 3		CH 2		CH 1		CH 0	

5.2.1 Assign Host Service Request Register Function

```
/*********************************************************************

FUNCTION      : tpu_hsr

PURPOSE       : To issue a host service request (HSR) to a channel
INPUTS NOTES  : This function has 3 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 number of the channel
          hsr - service request value
GENERAL NOTES : WARNING the HSR bits for a channel should be cleared
          before writing another HSR otherwise the requests will be
          ORed together. The host CPU can only set bits in the HSR
          registers so the values can be written directly with masks.
*********************************************************************/
void tpu_hsr(struct TPU3_tag *tpu, UINT8 channel, UINT8 hsr)
```

```
{
    if (channel < 8) {
        tpu->HSRR1.R = (hsr << (channel * 2));
    }
    else {
        tpu->HSRR0.R = (hsr << ((channel - 8) * 2));
    }
}
```

5.2.2 Get Host Service Request Register Function

```
/*********************************************************************
FUNCTION      : tpu_get_hsr
PURPOSE       : To get the current state of the HSR field
INPUTS NOTES  : This function has 2 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 number of the channel
RETURN NOTES : This function will return the pending HSR or zero
*********************************************************************/

UINT8 tpu_get_hsr(struct TPU3_tag *tpu, UINT8 channel)
{
    UINT16 hsr;


    if (channel < 8) {
        hsr = tpu->HSRR1.R;
    }
    else {
        channel -= 8;
        hsr = tpu->HSRR0.R;
    }


    hsr = ((hsr & (TPU_HSR_MASK << (channel * 2))) >> (channel * 2));


    return ((UINT8)hsr);
}
```

5.2.3 TPU Ready Macro Function

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

5.3 Host Sequence Registers

The host sequence field selects the mode of operation for the function selected on a given channel. The meaning of the host sequence bits depends on the function specified. Two routines are provided to read and write a channel's host sequence register. *tpu_hsq assigns* the host sequence to a channel and *tpu_get_hsq* returns the host sequence currently assigned to a channel.

HSQR0 — Host Sequence Register 0

0x30 4014
0x30 4414
0x30 5C14

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 15		CH 14		CH 13		CH 12		CH 11		CH 10		CH 9		CH 8	

HSQR1 — Host Sequence Register 1

0x30 4016
0x30 4416
0x30 5C16

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 7		CH 6		CH 5		CH 4		CH 3		CH 2		CH 1		CH 0	

5.3.1 Assign Host Sequence Register Function

```
/*****************************************************************************************
FUNCTION      : tpu_hsq
PURPOSE       : To set the host sequence bits (HSQ)
INPUTS NOTES  : This function has 3 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 number of the channel
          hsq - host sequence value
*****************************************************************************/
void tpu_hsq(struct TPU3_tag *tpu, UINT8 channel, UINT8 hsq)
{
    UINT16 reg;


    if (channel < 8) {
        reg = tpu->HSQR1.R;
        reg &= ~(TPU_HSQ_MASK << ((channel) * 2));
        reg |= (hsq << ((channel) * 2));
        tpu->HSQR1.R = reg;
    }
    else {
        channel -= 8;
        reg = tpu->HSQR0.R;
        reg &= ~(TPU_HSQ_MASK << ((channel) * 2));
```

```
        reg |= (hsq << ((channel) * 2));
        tpu->HSQR0.R = reg;
    }
}
```

5.3.2 Get Host Sequence Register Function

```
/*****************************************************************
FUNCTION      : tpu_get_hsq
PURPOSE       : To get the current state of the HSQ field
INPUTS NOTES  : This function has 2 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 number of the channel
RETURNS NOTES : This function will return the current HSQ field
*****************************************************/

UINT8 tpu_get_hsq(struct TPU3_tag *tpu, UINT8 channel)
{
    UINT16 hsq;


    if (channel < 8) {
        hsq = tpu->HSQR1.R;
    }
    else {
        channel -= 8;
        hsq = tpu->HSQR0.R;
    }


    hsq = (hsq & (TPU_HSQ_MASK << (channel * 2))) >> (channel * 2);


    return ((UINT8)hsq);
}
```

5.4 Channel Priority Registers

The channel priority registers (CPR0, CPR1) assign one of three priority levels to a channel or disable the channel. The definition of the values for the channel priority register are shown in Table 5. The TPU hardware scheduler allocates execution time based on the priority. See Figure for a graphic representation of the time allocation. Two routines are provided to read and write a channel's channel priority register. *tpu_enable* assigns the channel priority to a channel (which enables the function) and *tpu_disable* sets the channel priority for a channel to 0b00.

CPR0— Channel Priority Register 0

0x30 401C
0x30 441C
0x30 5C1C

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 15		CH 14		CH 13		CH 12		CH 11		CH 10		CH 9		CH 8	
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CPR1— Channel Priority Register 1

0x30 401E
0x30 441E
0x30 5C1E

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 7		CH 6		CH 5		CH 4		CH 3		CH 2		CH 1		CH 0	
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5 lists the values for the defined priorities for TPU functions.

Table 5. TPU Scheduler Priorities

Service Priority	Define	Value	Guaranteed Times Slots
High	TPU_PRIORITY_HIGH	3	4 out of 7
Medium	TPU_PRIORITY_MIDDLE	2	2 out of 7
Low	TPU_PRIORITY_LOW	1	1 out of 7
Disable	TPU_PRIORITY_DISABLE	0	None

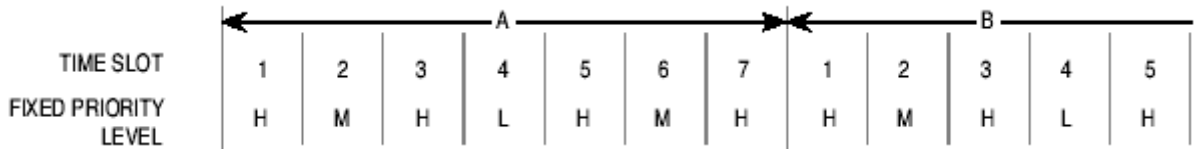

Figure 3. TPU Hardware Scheduler

5.4.1 TPU Enable Function

```
/*******************************************************************************
FUNCTION      : tpu_enable
PURPOSE       : To enable a TPU channel by assigning a priority to the channel
INPUTS NOTES  : This function has 3 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 number of the channel
          priority - This is the priority to assign to the channel.
          This parameter should be assigned a value of:
          TPU_PRIORITY_HIGH, TPU_PRIORITY_MIDDLE or TPU_PRIORITY_LOW.
*******************************************************************/

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

{

    UINT16 reg;


    if (channel < 8) {

        reg = tpu->CPR1.R;

        reg &= ~(TPU_PRIORITY_MASK << ((channel) * 2));

        reg |= (priority << ((channel) * 2));

        tpu->CPR1.R = reg;

    }

    else {

        reg = tpu->CPR0.R;

        reg &= ~(TPU_PRIORITY_MASK << ((channel-8) * 2));

        reg |= (priority << ((channel-8) * 2));

        tpu->CPR0.R = reg;

    }

}
```

5.4.2 TPU Disable Function

The TPU channels are disabled by setting the channel priority to TPU_PRIOTRITY_DISABLE (0b00).

NOTE

Disabling a channel does not actually stop the execution of that channel. It stops the channel after the current HSR routine completes.

```
/*******************************************************************************
FUNCTION      : tpu_disable
PURPOSE       : To disable a TPU channel by setting the priority bits to
                  0b00
INPUTS NOTES  : This function has 2 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 number of the channel
GENERAL NOTES : Disabling a channel will not stop a channel that is
          currently being executed by the TPU.
*******************************************************************/

void tpu_disable(struct TPU3_tag *tpu, UINT8 channel)
```

```
{
    UINT16 reg;


    if (channel < 8) {
        tpu->CPR1.R &= ~(TPU_PRIORITY_MASK << ((channel) * 2));
    }
    else {
        tpu->CPR0.R &= ~(TPU_PRIORITY_MASK << ((channel-8) * 2));
    }
}
```

5.5 Channel Interrupt Enable Register

The channel interrupt enable register (CIER) allows the CPU to enable or disable the ability of individual TPU3 channels to request interrupt service. Setting the appropriate bit in the register enables a channel to make an interrupt service request; clearing a bit disables the interrupt. Two routines are provided to enable and disable the channel's interrupt. *tpu_interupt_enable* sets the interrupt enable of a channel (which enables interrupts) and *tpu_interrupt_disable* clears the channel's interrupt bit, which disables the channels interrupt.

CIER— Channel Interrupt Enable Register

0x30 400A
0x30 440A
0x30 5C0A

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 15	CH 14	CH 13	CH 12	CH 11	CH 10	CH 9	CH 8	CH 7	CH 6	CH 5	CH 4	CH 3	CH 2	CH 1	CH 0

5.5.1 TPU Interrupt Enable Function

The interrupt for a channel is enabled by setting the channel's bit in the CIER register. The function *tpu_interrupt_enable* performs this action.

```
/*****************************************************************************
FUNCTION      : tpu_interrupt_enable
PURPOSE       : To enable interrupts on a channel
INPUTS NOTES  : This function has 2 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 number of the channel
*****************************************************************/

void tpu_interrupt_enable(struct TPU3_tag *tpu, UINT8 channel)

{


    tpu->CIER.R |= (1 << channel);


}
```

5.5.2 TPU Interrupt Disable Function

The interrupt for a channel is disabled by clearing the channel's bit in the CIER register. The function *tpu_interrupt_disable* performs this action.

```
/*************************************************************************************
FUNCTION      : tpu_interrupt_disable
PURPOSE       : To enable interrupts on a channel
INPUTS NOTES  : This function has 2 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 number of the channel
GENERAL NOTES : Disabling interrupts does not stop the interrupt flag
          from being set.
*************************************************************************/

void tpu_interrupt_disable(struct TPU3_tag *tpu, UINT8 channel)

{


    tpu->CIER.R &= ~(1 << channel);


}
```

5.6 Channel Interrupt Status Register

The channel interrupt status register (CISR) contains one interrupt status flag per channel. Functions specify via microcode when an interrupt flag is set. Setting a flag causes the TPU3 to make an interrupt service request if the corresponding CIER bit is set. To clear a status flag, read CISR, then write a zero to the appropriate bit. CISR is the only TPU3 register that can be accessed on a byte basis. Two routines are provided to clear and check a channel's Interrupt. *tpu_check_interrupt* checks the interrupt status bit to determine if the desired channel caused the interrupt and *tpu_clear_interrupt* clears the channel's interrupt status bit.

CISR — Channel Interrupt Status Register

0x30 4020
0x30 4420
0x30 5C20

MSB 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	LSB 15
CH 15	CH 14	CH 13	CH 12	CH 11	CH 10	CH 9	CH 8	CH 7	CH 6	CH 5	CH 4	CH 3	CH 2	CH 1	CH 0

5.6.1 TPU Clear Interrupt Function

The interrupt for a channel is cleared by reading the CISR and then clearing the channel's interrupt status bit in the CISR register. The function *tpu_clear_interrupt* performs this action.

```
/*************************************************************************************
FUNCTION      : tpu_clear_interrupt
PURPOSE       : To clear an interrupts on a channel
INPUTS NOTES  : This function has 2 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 number of the channel
GENERAL NOTES :
*************************************************************************/
```

```
void tpu_clear_interrupt(struct TPU3_tag *tpu, UINT8 channel)
{
    UINT16 dummy;

    dummy = tpu->CISR.R;

    tpu->CISR.R = ~(1 << channel);

}
```

5.6.2 TPU Check Interrupt Function

Since the main CPU only knows that a particular TPU caused an interrupt, based on the interrupt level set in the TICR, user code must determine which channel actually caused the interrupt. The function *tpu_check_interrupt* checks to see if a particular channel's interrupt status bit is set.

```
/*****************************************************************************
FUNCTION      : tpu_check_interrupt
PURPOSE       : To check interrupts on a channel
INPUTS NOTES  : This function has 2 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 number of the channel
GENERAL NOTES :
*****************************************************************/

UINT8 tpu_check_interrupt(struct TPU3_tag *tpu, UINT8 channel)
{
    UINT16 intstat;

    intstat = ((tpu->CISR.R) >> channel ) & 1;

    return ((UINT8) intstat);
}
```

6 References

For more information on TPU functions for the MPC500 family of devices, see the application notes shown in Table 6.

Table 6. MPC500 TPU Application Notes

Number	Title
AN2360/D	General TPU C Functions for the MPC500
AN2361/D	Comparison Between the M68332 TPU1 and the MPC500-Family TPU3
AN2362/D	Using the Fast Quadrature Decode TPU Function (FQD) with the MPC500 Family
AN2363/D	Using the Frequency Measurement TPU Function (FQM) with the MPC500 Family
AN2364/D	Using the Table Stepper Motor TPU Function (TSM) with the MPC500 Family
AN2365/D	Using the Programmable Time Accumulator TPU Function (PTA) with the MPC500 Family
AN2366/D	Using the New Input Transition / Input Capture TPU Function (NITC) with the MPC500 Family
AN2367/D	Using the Multiphase Motor Commutation TPU Function (COMM) with the MPC500 Family
AN2368/D	Using the Hall Effect Decode (HALLD) TPU Function with the MPC500 Family
AN2369/D	Using the Discrete Input / Output TPU Function (DIO) with the MPC500 Family
AN2370/D	Using the Quadrature Decode TPU Function (QDEC) with the MPC500 Family
AN2371/D	Using the Universal Asynchronous Receiver Transmitter TPU Function (UART) with the MPC500 Family
AN2372/D	Using the Output Compare TPU Function (OC) with the MPC500 Family
AN2373/D	Using the Pulse Width Modulation TPU Function (PWM) with the MPC500 Family
AN2374/D	Using the Queued Output Match TPU Function (QOM) with the MPC500 Family
AN2375/D	Using the Multichannel Pulse Width Modulation TPU Function (MCPWM) with the MPC500 Family

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