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 its 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 has 16 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

<table>
<thead>
<tr>
<th>Interrupt Level</th>
<th>CIRL</th>
<th>ILBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0-7</td>
<td>0b00</td>
</tr>
<tr>
<td>8-15</td>
<td>0-7</td>
<td>0b01</td>
</tr>
<tr>
<td>16-23</td>
<td>0-7</td>
<td>0b10</td>
</tr>
<tr>
<td>24-31</td>
<td>0-7</td>
<td>0b11</td>
</tr>
</tbody>
</table>

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.
General TPU C Functions for the MPC500 Family

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);`

For More Information On This Product, Go to: www.freescale.com
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.
Status Functions

- 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)

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

#### Table 3. TPU Function Parameter Addresses (32-bit)

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

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.

\[ \text{tpu->PARM.R[channel][parameter_number] = new_value16;} \]

The second sets a 32 bit parameter.

\[ \text{tpu->PARM.L[channel][parameter_number] = new_value32;} \]

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. \text{tpu_func} assigns the function to a channel and \text{tpu_get_func} returns the function currently assigned to a channel.
CFSR0 — Channel Function Select Register 0

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

<table>
<thead>
<tr>
<th>MSB</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CH 15</td>
<td>CH 14</td>
<td>CH 13</td>
<td>CH 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CFSR1 — Channel Function Select Register 1

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

<table>
<thead>
<tr>
<th>MSB</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CH 11</td>
<td>CH 10</td>
<td>CH 9</td>
<td>CH 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CFSR2 — Channel Function Select Register 2

0x30 4010
0x30 4410
0x30 5C10

<table>
<thead>
<tr>
<th>MSB</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CH 7</td>
<td>CH 6</td>
<td>CH 5</td>
<td>CH 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CFSR3 — Channel Function Select Register 3

0x30 4012
0x30 4412
0x30 5C12

<table>
<thead>
<tr>
<th>MSB</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CH 3</td>
<td>CH 2</td>
<td>CH 1</td>
<td>CH 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4. TPU ROM Functions

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

General TPU C Functions for the MPC500 Family

For More Information On This Product, Go to: www.freescale.com
### 5.1.1 Assign TPU Function

```c
/*******************************************************************************
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;
    }

    return 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.

<table>
<thead>
<tr>
<th>HSRR0 — Host Service Request Register 0</th>
<th>0x30 4018</th>
<th>0x30 4418</th>
<th>0x30 5C18</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH 15</td>
<td>CH 14</td>
<td>CH 13</td>
<td>CH 12</td>
</tr>
<tr>
<td>CH 11</td>
<td>CH 10</td>
<td>CH 9</td>
<td>CH 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HSRR1 — Host Service Request Register 1</th>
<th>0x30 401A</th>
<th>0x30 441A</th>
<th>0x30 5C1A</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH 7</td>
<td>CH 6</td>
<td>CH 5</td>
<td>CH 4</td>
</tr>
<tr>
<td>CH 3</td>
<td>CH 2</td>
<td>CH 1</td>
<td>CH 0</td>
</tr>
</tbody>
</table>

5.2.1 Assign Host Service Request Register Function

```c
void tpu_hsr(struct TPU3_tag *tpu, UINT8 channel, UINT8 hsr)
```

*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.
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

<table>
<thead>
<tr>
<th>MSB</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH 15</td>
<td>CH 14</td>
<td>CH 13</td>
<td>CH 12</td>
<td>CH 11</td>
<td>CH 10</td>
<td>CH 9</td>
<td>CH 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### HSQR1 — Host Sequence Register 1

<table>
<thead>
<tr>
<th>MSB</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH 7</td>
<td>CH 6</td>
<td>CH 5</td>
<td>CH 4</td>
<td>CH 3</td>
<td>CH 2</td>
<td>CH 1</td>
<td>CH 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 5.3.1 Assign Host Sequence Register Function

```c
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));
    }
}
```

Freescale Semiconductor, Inc.

For More Information On This Product, Go to: www.freescale.com
5.3.2 Get Host Sequence Register Function

/* ************************************************************************
 * FUNCTION      : tpu_get_hsq
 * PURPOSE       : To get the current state of the HSQ field
 * INPUTS NOTES  :
 * RETURNS NOTES :
 * **************************************************************************/

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

<table>
<thead>
<tr>
<th>MSB</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH 15</td>
<td>CH 14</td>
<td>CH 13</td>
<td>CH 12</td>
<td>CH 11</td>
<td>CH 10</td>
<td>CH 9</td>
<td>CH 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RESET:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

**CPR1**— Channel Priority Register 1

<table>
<thead>
<tr>
<th>MSB</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH 7</td>
<td>CH 6</td>
<td>CH 5</td>
<td>CH 4</td>
<td>CH 3</td>
<td>CH 2</td>
<td>CH 1</td>
<td>CH 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Service Priority</th>
<th>Define</th>
<th>Value</th>
<th>Guaranteed Times Slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>TPU_PRIORITY_HIGH</td>
<td>3</td>
<td>4 out of 7</td>
</tr>
<tr>
<td>Medium</td>
<td>TPU_PRIORITY_MIDDLE</td>
<td>2</td>
<td>2 out of 7</td>
</tr>
<tr>
<td>Low</td>
<td>TPU_PRIORITY_LOW</td>
<td>1</td>
<td>1 out of 7</td>
</tr>
<tr>
<td>Disable</td>
<td>TPU_PRIORITY_DISABLE</td>
<td>0</td>
<td>None</td>
</tr>
</tbody>
</table>

**Figure 3. TPU Hardware Scheduler**
### 5.4.1 TPU Enable Function

```c
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_PRIORITY_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.

```c
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_interrupt_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

<table>
<thead>
<tr>
<th></th>
<th>CH 15</th>
<th>CH 14</th>
<th>CH 13</th>
<th>CH 12</th>
<th>CH 11</th>
<th>CH 10</th>
<th>CH 9</th>
<th>CH 8</th>
<th>CH 7</th>
<th>CH 6</th>
<th>CH 5</th>
<th>CH 4</th>
<th>CH 3</th>
<th>CH 2</th>
<th>CH 1</th>
<th>CH 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

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.

```c
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**

<table>
<thead>
<tr>
<th>MSB</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH 15</td>
<td>CH 14</td>
<td>CH 13</td>
<td>CH 12</td>
<td>CH 11</td>
<td>CH 10</td>
<td>CH 9</td>
<td>CH 8</td>
<td>CH 7</td>
<td>CH 6</td>
<td>CH 5</td>
<td>CH 4</td>
<td>CH 3</td>
<td>CH 2</td>
<td>CH 1</td>
<td>CH 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**0x30 4020**

**0x30 4420**

**0x30 5C20**

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.

```c
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 : Disabling interrupts does not stop the interrupt flag
                from being set.
*******************************************************************************/
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>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN2360/D</td>
<td>General TPU C Functions for the MPC500</td>
</tr>
<tr>
<td>AN2361/D</td>
<td>Comparison Between the M68332 TPU1 and the MPC500-Family TPU3</td>
</tr>
<tr>
<td>AN2362/D</td>
<td>Using the Fast Quadrature Decode TPU Function (FQD) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2363/D</td>
<td>Using the Frequency Measurement TPU Function (FQM) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2364/D</td>
<td>Using the Table Stepper Motor TPU Function (TSM) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2365/D</td>
<td>Using the Programmable Time Accumulator TPU Function (PTA) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2366/D</td>
<td>Using the New Input Transition / Input Capture TPU Function (NITC) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2367/D</td>
<td>Using the Multiphase Motor Commutation TPU Function (COMM) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2368/D</td>
<td>Using the Hall Effect Decode (HALLD) TPU Function with the MPC500 Family</td>
</tr>
<tr>
<td>AN2369/D</td>
<td>Using the Discrete Input / Output TPU Function (DIO) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2370/D</td>
<td>Using the Quadrature Decode TPU Function (QDEC) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2371/D</td>
<td>Using the Universal Asynchronous Receiver Transmitter TPU Function (UART) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2372/D</td>
<td>Using the Output Compare TPU Function (OC) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2373/D</td>
<td>Using the Pulse Width Modulation TPU Function (PWM) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2374/D</td>
<td>Using the Queued Output Match TPU Function (QOM) with the MPC500 Family</td>
</tr>
<tr>
<td>AN2375/D</td>
<td>Using the Multichannel Pulse Width Modulation TPU Function (MCPWM) with the MPC500 Family</td>
</tr>
</tbody>
</table>
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 Habbleheim Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 5220080 (English)
+49 89 92103 559 (German)
+33 1 69 36 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 8808
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
LDCFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.