Timers and Interrupts

We have been using the timers in the polled mode but this is not efficient since it ties up the CPU doing little but testing a timer to see if it is done. A better way to approach this problem is to use interrupts.

The interrupt system makes use of match registers. A timer such as Timer0 is started and when its value is identical to the value preset in a match register the event triggers an interrupt.

In software, all of the following items have to be set up in order to trigger an interrupt using one of the timers:

• Load the clock divider for PCLK.
• Set up the timer prescale register.
• Reset the timer and its prescale register.
• Set up the match control register. This determines how the timer generates the interrupt.
• Load the match value in the timer match register. When the timer value equals the match register value the action set up in the match control register is triggered.
• Write the interrupt service routine (ISR). This is the procedure which runs when the interrupt occurs.
• In the ISR it is necessary to do two things: 1) Clear the match register interrupt so that it is no longer pending and 2) Signal the end of the interrupt.
• Load the address of the ISR into the interrupt address register. This is how the interrupt finds the ISR.
• Enable the particular interrupt being used and set up the interrupt channel.
• Enable the global interrupt bit.
• Start the timer.

There are two timer/counters named Timer0 and Timer1. Each timer has four match registers associated with it. We will load one of these match registers with a preset value so that when the timer reaches that value an interrupt is triggered. The other registers can be used for pulse width modulation (PWM) which we will do in a later lecture. From the diagram of the timer system we note that the timer is simultaneously compared to all four match registers at one time and this compare is done in hardware.

A Match Control Register determines what happens when a match occurs. When a match register value is equal to the timer counter register you can reset the counter, trigger an interrupt, stop the counter, or set an external pin. The match registers are controlled by the match control register.
Match Control Register (MCR) actions. The left most column is the bit position in the MCR.

Table 131: Match Control Register (MCR: TIMER0 - TIMCR: 0x00002414; TIMER1 - TIMCR: 0x00008014)

<table>
<thead>
<tr>
<th>MCR</th>
<th>Function</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>interrupt on MR0</td>
<td>When one, an interrupt is generated when MR0 matches the value in the TC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Reset on MR0</td>
<td>When one, the TC will be reset if MR0 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Stop on MR0</td>
<td>When one, the TC and PC will be stopped and TC[0] will be set to 0 if MR0 matches the TC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>interrupt on MR1</td>
<td>When one, an interrupt is generated when MR1 matches the value in the TC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Reset on MR1</td>
<td>When one, the TC will be reset if MR1 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>Stop on MR1</td>
<td>When one, the TC and PC will be stopped and TC[0] will be set to 0 if MR1 matches the TC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>interrupt on MR2</td>
<td>When one, an interrupt is generated when MR2 matches the value in the TC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Reset on MR2</td>
<td>When one, the TC will be reset if MR2 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Stop on MR2</td>
<td>When one, the TC and PC will be stopped and TC[0] will be set to 0 if MR2 matches the TC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>interrupt on MR3</td>
<td>When one, an interrupt is generated when MR3 matches the value in the TC. When zero this interrupt is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>Reset on MR3</td>
<td>When one, the TC will be reset if MR3 matches it. When zero this feature is disabled.</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>Stop on MR3</td>
<td>When one, the TC and PC will be stopped and TC[0] will be set to 0 if MR3 matches the TC. When zero this feature is disabled.</td>
<td>0</td>
</tr>
</tbody>
</table>

* Note that Capture Register 3 cannot be used on TIMER0
• Load the clock divider for PCLK.  
  For example,
  VPBDIV = 0x00000002; //Set up peripheral clock for 30MHz

• Set up the timer prescale register.  
• Reset the timer and its prescale register.  
  For example,
  T0PR = 0x0000001D; //Set prescaler to 29 -> 1MHz timer clock
  T0TCR = 0x00000002; //Reset Timer0 & prescaler and
  // disable prescale counter.

• Set up the match control register.  This determines how the timer generates the interrupt.  
  For example, according to the match control register table above:
  T0MCR = 0x00000003; //Generate int & reset Timer0 on match to MR0

  will reset Timer0 on a match and generate an interrupt since this line sets both bit 0 and bit 1.

• Load the match value in the timer match register.  When the timer value equals the match
  register value the action set up in the match control register is triggered.  
  For example, if we want to have a match when the timer gets to 1000 counts we would write:
  T0MR0 = 0x000003E8; //Match value = 1000 or 3E8h

The interrupt must also be set up.  The interrupt source table gives the sources for all interrupts. 
Timer 0 is on Interrupt Channel 4 and this is for both the match register and the capture register.

The interrupts are enabled using the VICVectCntl register.  Each interrupt channel has its own 
VICVectCntl register.  Since timer0 is on channel 4 we want to operate on VICVectCntl4.  Bit 5 
in this register enables the interrupt request (IRQ) channel and bits 0-4 specify the particular 
channel.

• Enable the particular interrupt being used and set up the interrupt channel.  
  This means we need to set up the vector control register for this particular interrupt.  The 
table on the following page (at the bottom) shows that Bit 5 in this register is set to 1 for the 
ablenable and bits 0-4 must be set to the slot number.

  For example, timer0 interrupt comes in on channel 4 and bits 0-4 must be set to specify the 
channel.  Bit 5 enables the IRQ channel to work.  So we need a bit pattern that looks like 
this: 00 ... 0010 0100 which is 24 in hex.
  Each channel has its own vector control register.  For channel 4 this is VICVectCntl4.
  
  VICVectCntl4 = 0x00000024; //bit 5 enables interrupt, bits 0-4 set ch.
This table gives channel numbers for particular interrupts. Note that Timer0 is on Ch. 4 and Timer1 is on Ch. 5.

This is the Vector Control Register or VICVectCntl. Bit 5 enables the slot (channel) and bits 0-4 determine the channel number. Every channel has its own VICVectCntl register.
• Enable the global interrupt bit.

Table 37: Interrupt Enable Register (VICIntEnable - 0xFFFFF010, Read/Write)

<table>
<thead>
<tr>
<th>VICIntEnable</th>
<th>Function</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>When this register is read, 1s indicate interrupt requests or software interrupts that are enabled to contribute to FIQ or IRQ. When this register is written, ones enable interrupt requests or software interrupts to contribute to FIQ or IRQ, zeroes have no effect. See the VICIntEnable register (Table 38 below), for how to disable interrupts.</td>
<td>0</td>
</tr>
</tbody>
</table>

This register VICIntEnable is the global interrupt enable register.

For example, we want to enable channel 4 so we need a bit pattern of 00 … 0001 0000 or 0x00000010 in hexadecimal. Note that we use the |= (or equals) operator so as not to change the status of other interrupts.

VICIntEnable |= 0x00000010;  //Enable the interrupts

• Write the interrupt service routine (ISR). This is the procedure which runs when the interrupt occurs.

• In the ISR it is necessary to do two things: 1) Clear the match register interrupt so that it is no longer pending and 2) Signal the end of the interrupt.

Table 128: Interrupt Register (IR: TIMER0 - T0IR: 0xE0040000; TIMER1 - T1IR: 0xE0080000)

<table>
<thead>
<tr>
<th>IR</th>
<th>Function</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MR0 interupt</td>
<td>interrupt flag for match channel 0.</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>MR1 interupt</td>
<td>interrupt flag for match channel 1.</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>MR2 interupt</td>
<td>interrupt flag for match channel 2.</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>MR3 interupt</td>
<td>interrupt flag for match channel 3.</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>CR0 interupt</td>
<td>interrupt flag for capture channel 0 event.</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>CR1 interupt</td>
<td>interrupt flag for capture channel 1 event.</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>CR2 interupt</td>
<td>interrupt flag for capture channel 2 event.</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>CR3 interupt</td>
<td>interrupt flag for capture channel 3 event.</td>
<td>0</td>
</tr>
</tbody>
</table>

This table gives the bit that is used to clear the interrupt after it has been serviced. If we use Match Register 0 (MR0) then we need to operate on bit 0 of this register (T0IR).

In addition the VICVectAddr register must be written to in order to clear the interrupt priorities. This is done in the ISR as a dummy write.

Table 45: Vector Address Register (VIC/ VectAddr - 0xFFFFF030, Read/Write)

<table>
<thead>
<tr>
<th>VICVectAddr</th>
<th>Function</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>if any of the interrupt requests or software interrupts that are assigned to a vectored IRQ slot is (are) enabled, classified as IRQ, and asserted, reading from this register returns the address in the Vector Address Register for the highest-priority such slot (lowest-numbered) such slot. Otherwise it returns the address in the Default Vector Address Register. Writing to this register does not set the value for future reads from it. Rather, this register should be written near the end of an ISR, to update the priority hardware.</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: Philips must have just run out of names to call the registers. A check of LPC213X.H shows that there is indeed a unique address for VICVectAddr as well as VICVectAddr0 through VICVectAddr15.
For example, a typical ISR looks like this:

```c
void Timer0ISR(void) __irq
{//Perform the task required when the interrupt happens.
 //  This might include updating a global variable or possibly
 //  setting or clearing an I/O bit.
 //  ...
 T0IR |= 0x00000001;     //Clear match 0 interrupt
 VICVectAddr = 0x00000000;  //Dummy write to signal end of interrupt
}
```

• Load the address of the ISR into the interrupt address register. This is how the interrupt finds the ISR.

To set up the interrupt service routine (ISR) we must set the vectored interrupt address register for that channel to the address of the timer 0 ISR. For example, if our interrupt service routine is named Timer0ISR and we are operating on channel 4 we would write this address to VICVectAddr4.

```
VICVectAddr4 = (unsigned)Timer0ISR; //Assign addr of ISR to int addr reg
```

• Start the timer.

For example, to start timer0 we load a 1 into its control register.

```
T0TCR = 0x00000001; //Start counter 0
```

The program TimerInt.c shown below generates a 500Hz square wave on P1.16 using timer 0 in an interrupt mode to keep track of real time. The timer forces an interrupt every millisecond

**Class exercise:** Modify TimerInt.c to use a timer0 and an interrupt routine to output a sinusoid of a given frequency of say 1,000Hz?

**Class exercise:** If the peripheral clock divider and the timer prescaler are set to the their maximum values what is the longest time period that timer 0 can count before an interrupt in the above program?
/* TimerInt.c                                             July 5, 2007  
This program outputs a 0.5 KHz square wave on pin P1.16 using 
    Timer 0. The timer forces an interrupt. 
*/
#include <LPC213X.H>

void Timer0ISR(void) __irq; 
int main(void)
{
    VPBDIV = 0x00000002; //Set up peripheral clock for 30MHz
    PINSEL2 = 0x00000000; //Sets all pins on Port 1 as GPIO
    T0PR = 0x0000001D; //Set prescaler to 30 -> 1MHz timer clock
    T0TCR = 0x00000002; //Reset Timer0 & prescaler; disable prescale counter
    T0MCR = 0x00000003; //Generate interrupt and reset Timer on match to MR0
    T0MR0 = 0x000003E8; //Match value = 1000 or 3E8h
    IODIR1 = 0x00010000; //Set P1.16 to output. All others are input
    T0TCR = 0x00000001; //Start counter 0
    VICVectAddr4 = (unsigned)Timer0ISR; //Assign addr of ISR to int addr reg
    VICVectCntl4 = 0x00000024; //bit 5 enables interrupt, bits 0-4 set ch.
    VICIntEnable |= 0x00000010; //Enable the interrupts
    while(1);
}

//This is the interrupt service routine.
void Timer0ISR(void) __irq
{
    IOPIN1 = ~IOPIN1; //Complements output bit on P1.16
    T0IR |= 0x00000001; //Clear match 0 interrupt
    VICVectAddr = 0x00000000; //Dummy write to signal end of interrupt
}