BIRZEIT UNIVERSITY

Electrical and Computer Engineering

Computer Design Lab – ENCS4110

Exp9: Interrupts - Timer0

Objectives:

1. To be familiar with the interrupts in LPC2138

2. To be familiar with timers in LPC2138

Interrupts

An interrupt is a signal informing a program that an event has occurred. When a program receives an interrupt signal, it takes a specified action (which can be to ignore the signal). Interrupt signals can cause a program to suspend itself temporarily to service the interrupt.

Interrupts in LPC213x are handled by Vectored Interrupt Controller (VIC). VIC , as per its design , can take 32 interrupt request inputs but only 16 requests can be assigned to Vectored IRQ interrupts in its LCP2138 Implementation. We are given a set of 16 vectored IRQ slots to which we can assign any of the 22 requests that are available in LPC2138. The slot numbering goes from 0 to 15 with slot no. 0 having highest priority and slot no. 15 having lowest priority.

Here is the complete table which says which bit corresponds to which interrupt source as given in Datasheet where the Bit number is the Source Interrupt Number:

Bit#	22	21	20	19	18	17	16	15	14	13	12	11
IRQ	USB	AD1	BOD	I2C1	AD0	EINT3	EINT2	EINT1	EINT0	RTC	PLL	SPI1/SSP
Bit#	10	9	8	7	6	5	4	3	2	1	0	
IRQ	SPI0	I2C0	PWM	UART1	UART0	TIMR1	TIMR0	ARMC1	ARMC0	N/A	WDT	

LPC2138 Interrupt Related Registers

1) VICIntSelect (R/W) :

This register is used to select an interrupt as IRQ or as FIQ. Writing a 0 at a given bit location(as given in Table 6.1) will make the corresponding interrupt as IRQ and writing a 1 will make it FIQ.. Note that here IRQ applies for both Vectored and Non-Vectored IRQs.

2) VICIntEnable (R/W) :

This is used to enable interrupts. Writing a 1 at a given bit location will make the corresponding interrupt Enabled. If this register is read then 1's will indicated enabled interrupts and 0's as disabled interrupts. Writing 0's has no effect.

3) VICIntEnClr (R/W) :

This register is used to disable interrupts. This is similar to VICIntEnable expect writing a 1 here will disabled the corresponding Interrupt. This has an effect on VICIntEnable since writing at bit given location will clear the corresponding bit in the VICIntEnable Register.

4) VICIRQStatus (R) :

This register is used for reading the current status of the enabled IRQ interrupts. If a bit location is read as 1 then it means that the corresponding interrupt is enabled and active.

5) VICFIQStatus (R) :

Same as VICIRQStatus except it applies for FIQ.

6) VICSoftInt :

This register is used to generate interrupts using software i.e. manually generating interrupts using code. If you write a 1 at any bit location then the corresponding interrupt is triggered i.e. it forces the interrupt to occur. Writing 0 here has no effect.

7) VICSoftIntClear :

This register is used to clear the interrupt request that was triggered(forced) using VICSoftInt. Writing a 1 will release(or clear) the forcing of the corresponding interrupt.

8) VICVectCntl0 to VICVectCntl15 (16 registers in all) :

These are the Vector Control registers. These are used to assign a particular interrupt source to a particular slot. As mentioned before slot 0 i.e. VICVectCntl0 has highest priority and VICVectCntl15 has the lowest. Each of this registers can be divided into 3 parts : {Bit0 to bit4} , {Bit 5} , {and rest of the bits}.The first 5 bits i.e. Bit 0 to Bit 4 contain the number of the interrupt request which is assigned to this slot. The 5th bit is used to enable the vectored IRQ slot by writing a 1. The rest of the bits are reserved.

Note:

if the vectored IRQ slot is disabled it will not disable the interrupt but will change the corresponding interrupt to Non-Vectored IRQ. Enabling the slot here means that it points to specific and dedicated interrupt handling function (ISR) and disabling it will point to the default function.

9) VICVectAddr0 to VICVectAddr15 (16 registers in all) :

For Vectored IRQs these register store the address of the function that must be called when an interrupt occurs.

10) VICVectAddr :

This must not be confused with the above set of 16 VICVecAddrX registers. When an interrupt is Triggered this register holds the address of the associated ISR i.e. the one which is currently active. Writing a value i.e. dummy write to this register indicates to the VIC that current Interrupt has finished execution. So the only place we'll use this register is at the end of the ISR to signal end of ISR execution.

11) **VICDefVectAddr** :

This register stores the address of the "default/common" ISR that must be called when a Non-Vectored IRQ occurs.

 To define the ISR we need to explicitly tell the compiler that the function is not a normal function but an ISR. For this we'll use a special keyword called "__irq" which is a function qualifier. If you use this keyword with the function definition then compiler will automatically treat it as an ISR. Here is an example on how to define an ISR in Keil :

__irq void myISR (void){

...

}

 Consider we want to assign TIMER0 IRQ and ISR to slot X. Here is a simple steps do that :

1. First we need to enable the TIMER0 IRQ itself. Hence , from Table 1 we get the bit number to Enable TIMER0 Interrupt which is bit number 4. Hence we must make bit 4 in VICIntEnable to '1'.

2. Next , from Table 1 we get the interrupt source number for TIMER0 which is decimal 4 and OR it with (1<<5) to enables the slot and assign it to VICVectCntlX.

3. Next assign the address of the related ISR to VICVectAddrX.

Here is a simple Template to do it:

Replace X by the slot number you want .., then Replace Y by the Interrupt Source Number as given in Table 2 and finally replace myISR with your own ISR's function Name .. and you're Done!

```
VICIntEnable |= (1<<Y) ;
VICVectCntlX = (1<<5) | Y ;
VICVectAddrX = (unsigned) myISR;
```

Using above steps we can Assign TIMER0 Interrupt to Slot number say.. 0 as follows :

```
VICIntEnable |= (1<<4) ; // Enable TIMER0 IRQ
VICVectCntl4 = (1<<5) | 4 ; //5th bit must 1 to enable the slot
VICVectAddr4 = (unsigned) myISR;//Vectored-IRQ for TIMER0 has been configured
```

LPC2138 Timer Registers

The below table shows the registers associated with LPC2138 Timer module

Register		Description
IR	Interrupt Register	The IR can be read to identify which of 6(4-match, 2-Capture) possible interrupt sources are pending. Writing Logic-1 will clear the corresponding interrupt.
TCR	Timer Control Register	The TCR is used to control the Timer Counter functions(enable/disable/reset).
TC	Timer Counter	The 32-bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.
PR	Prescaler Register	This is used to specify the Prescaler value for incrementing the TC.

PC	**Prescale Counter**	The 32-bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented.
MCR	**Match Control Register**	The MCR is used to control the resetting of TC and generating of interrupt whenever a Match occurs.
MR0-MR3	**Match Registers**	The Match register values are continuously compared to the Timer Counter value. When the two values are equal, actions can be triggered automatically. The action possibilities are to generate an interrupt, reset the Timer Counter, or stop the timer. Actions are controlled by the settings in the MCR register.

Timers Registers Configuration:

TCR

Bit2-Bit31	Bit 1	Bit 0
Reserved	Counter Reset	Counter Enable

Bit 0 ? Counter Enable
This bit is used to Enable or Disable the Timer Counter and Prescalar Counter.
0- Disable the Counters
1-Enable the Counter incrementing.

Bit 1 ? Counter reset
This bit is used to clear the Timer counter.
0- Do not Clear.
1-The Timer Counter and the Prescaler Counter are synchronously reset on the next positive edge of PCLK.

MCR

Bit12-Bit31	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Reserved	MR3S	MR3R	MR3I	MR2S	MR2R	MR2I	MR1S	MR1R	MR1I	MR0S	MR0R	MR0I

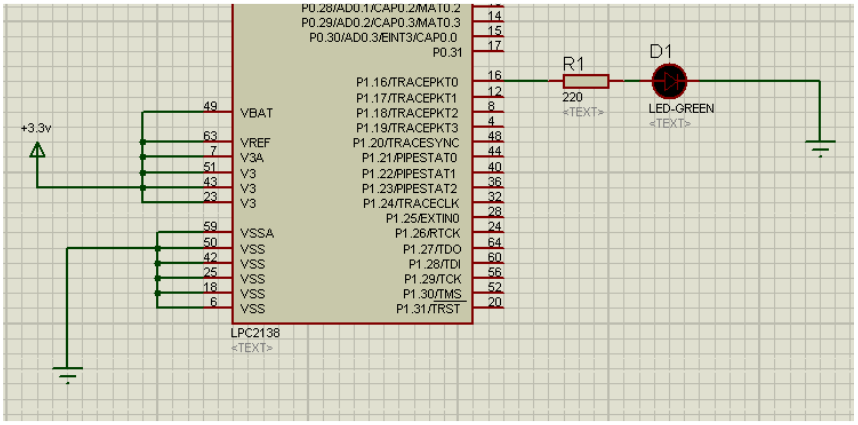
MRxI
This bit is used to Enable or Disable the Timer interrupts when the TC matches MRx (x:0-3)
0- Disable the Timer Match interrupt
1-Enable the Timer Match interrupt.

MRxR
This bit is used to Reset TC whenever it Matches MRx(x:0-3)
0- Do not Clear.
1-Reset TC counter value whenever it matches MRx.

MRxS
This bit is used to Stop TC and PC whenever the TC matches MRx(x:0-3).
0- Disable the Timer stop on match feature
1-Enable the Timer Stop feature. This will stop the Timer whenever the TC reaches the Match register value.

Lab Work 1: Write a program that blinks a led every 3 seconds using timer0 interrupt.

Proteus:

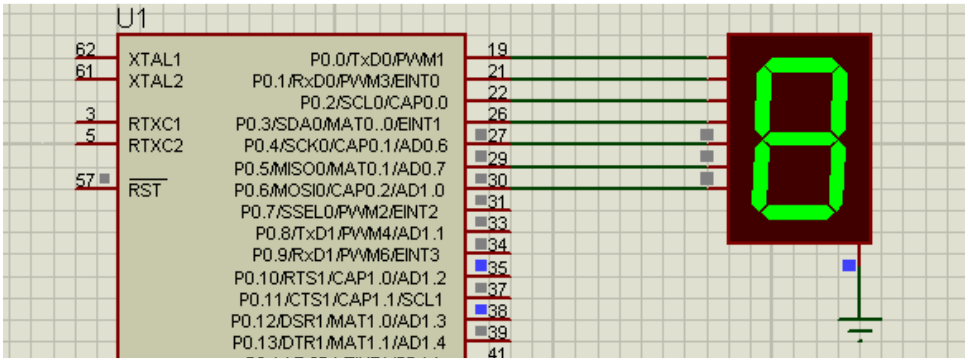



```
#include<lpc213x.h>

__irq void Timer0_IRQ(void);
void Timer0_Init(void);

int counter;
int main(void) {
        IO0DIR |= (1<<16);
        IO0CLR = 1<<16;

        Timer0_Init(); //initialize timer0 with 1 sec delay
        T0TCR = 0x01; //start timer0

        while(1)
            {
              if(counter ==3){ // when counter reachs 3 seconds
                        IO0PIN ^= (1<<16); //toggle the led
                        counter=0;
                    }
            }
      }


__irq void Timer0_IRQ(void)
{
      //IO0PIN ^=(1<<16);
      counter++;
      T0IR = ( T0IR | (0x01) );
      VICVectAddr = 0x00;
}
```

```
void Timer0_Init(void)
{
        VPBDIV = 0x1;  //make pclk = cclk (=12MHz)
        //timer0 configuration//
        T0CTCR = 0x00;     // select timer mode
        T0PR = 12000-1; // pre-scale value 12MHz/12000 = 1000 = 1ms
        T0MR0 = 1000-1; // for 1000 ms = 1 sec

        T0MCR = 3;
        T0TC = 0x00;

        VICVectAddr4 = (unsigned) Timer0_IRQ;
        VICVectCntl4 = (0x20 | 4);
        VICIntEnable |= (1<<4);
}
```

Lab Work 2: Write a program that counts from 0 to 9 using Timer0 interrupt and display the result on 7-segment. Counting rate every one second.

Proteus:



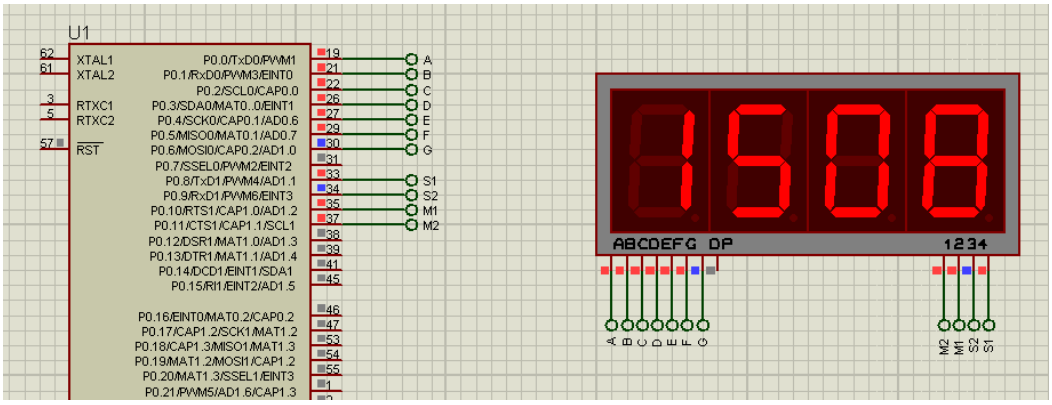
Keil:

```
#include <lpc213x.h>
__irq void T0_ISR(void);
int dec_to_7seg(int number);

int counter = 0 ;

int main(void){

  IO0DIR |= 0x7F; //Configure pins 0..6 on port 0 output

  TMR0_init(T0_ISR);
  TMR0_start();

  while(1); // run forever

}
```

```
__irq void Timer0_IRQ(void)
{
      IO0CLR = 0x7F;  //clear 7-seg
      IO0SET = dec_to_7seg(counter);  // load next number
      Counter++;
      If(counter >9) counter = 0;
       T0IR = ( T0IR | (0x01) );
       VICVectAddr = 0x00;
}
```

Lab Work 3: Write a program that Implements a clock with MM:SS timing using timer 0 and multiplexing.

Proteus:



Keil:

```
#include<lpc213x.h>

__irq void Timer0_IRQ(void);
void Timer0_Init(void);
int dec_to_7seg(int number);
void delay(void);

int s1, s2, m1, m2 =0;

int main(void) {

        IO0DIR |= 0x7F;  //P0.0 ... P0.6 output
        IO0DIR |= 0xF00;  // P0.8 ... P0.11 output

        Timer0_Init(); //initialize timer0 with 1 sec delay
        T0TCR = 0x01; //start timer0

      while(1){
            IO0SET |= (1<<9) | (1<<10) | (1<<11); //disable s2,m1,m2
            IO0CLR |= (1<<8); //Enable s1
            IO0CLR = 0x7F; //clear 7-seg
            IO0SET = dec_to_7seg(s1); //load s1 on 7-seg
            delay();  //display s1 for short time

            IO0SET |= (1<<8) | (1<<10) | (1<<11); //disable s1,m1,m2
            IO0CLR |= (1<<9); //Enable s2
            IO0CLR = 0x7F; //clear 7-seg
```

```
            IO0SET = dec_to_7seg(s2); //load s2 on 7-seg
            delay();  //display s2 for short time

            IO0SET |= (1<<8) | (1<<9) | (1<<11); //disable s1,s2,m2
            IO0CLR |= (1<<10); //Enable m1
            IO0CLR = 0x7F; //clear 7-seg
            IO0SET = dec_to_7seg(m1); //load s1 on 7-seg
            delay();  //display m1 for short time

            IO0SET |= (1<<8) | (1<<9) | (1<<10); //disable s1,s2,m1
            IO0CLR |= (1<<11); //Enable m2
            IO0CLR = 0x7F; //clear 7-seg
            IO0SET = dec_to_7seg(m2); //load m2 on 7-seg
            delay();  //display m2 for short time

      }
}

__irq void Timer0_IRQ(void){
       s1++;
       if(s1 == 10){
             s1 = 0;
             s2 += 1;
       }

       if(s2 == 6){
             s2 = 0;
             m1 += 1;
       }

       if(m1 == 10) {
             m1 = 0;
             m2 += 1;
       }

      T0IR = ( T0IR | (0x01) );
      VICVectAddr = 0x00;
 }

 void delay(){
       int i,j;
       long c = 0;
       for(i=0;i<10;i++){
             for(j=0;j<1000;j++){
                   c++;
             }
       }
 }

int dec_to_7seg(int number)
 {
        switch(number){
                  case 0 : return(0x3F);
                  case 1 : return(0x06);
                  case 2 : return(0x5B);
                  case 3 : return(0x4F);
                  case 4 : return(0x66);
```

```
                case 5 : return(0x6D);
                case 6 : return(0x7D);
                case 7 : return(0x07);
                case 8 : return(0x7F);
                case 9 : return(0x6F);
                default : return(0x00);
            }
      }

      void Timer0_Init(void)
{
        VPBDIV = 0x1;  //make pclk = cclk (=12MHz)
        //timer0 configuration//
        T0CTCR = 0x00;    // select timer mode
        T0PR = 12000-1; // pre-scale value 12MHz/12000 = 1000 = 1ms
        T0MR0 = 1000-1; // for 1000 ms = 1 sec

        T0MCR = 3;
        T0TC = 0x00;

        VICVectAddr4 = (unsigned) Timer0_IRQ;
        VICVectCntl4 = (0x20 | 4);
        VICIntEnable |= (1<<4);
}
```

ToDo Task:
Write a program/simulation that contains initially: LCD with initial message "System is On" and shining led.

- After 1 minutes pass, LCD message must be "System is Off".
- After 5 minutes led must be turned off.