Lab 3 – Part 2
Using the MC9S12 Timer Overflow Interrupt and Real Time Interrupt

Introduction and Objectives

Enabling an interrupt on the MC9S12 allows your program to respond to an external event without continually checking to see if that event has occurred. Once the event occurs, the MC9S12 interrupt subsystem will transfer control of your program to an interrupt service routine (ISR) to handle the event, and then return control to your original code sequence. In this week's lab you will write assembly and C language program which enable and use interrupts.

The interrupts on the MC9S12 which are easiest to use are the Timer Overflow Interrupt and the Real Time Interrupt. These interrupts allow you to interrupt the microcontroller after a specified amount of time has passed.

1. Prelab

For the prelab write the program for Parts 6 and 7. Also, calculate the time asked for in 5.

2. The Lab

1. Connect your MC9S12 to your computer. At the DBug 12 prompt, display the contents of the TCNT register. Do this several times. How do the values compare?

2. Use DBug12 to modify the TSCR1 register to enable the counter. Repeat Part 1.

3. Use DBug12 to modify the TSCR1 register to disable the counter. Repeat Part 1.

4. Start with Program 1: The file "vectors12.inc" is available at http://www.ee.nmt.edu/~erives/308L_13/EE308L.html. This file contains the DBug12 RAM-based interrupt vectors.

Add code to make PORTB an output port, to enable the eight individual LEDs, and disable the seven segment LEDs. Then add a Timer Overflow Interrupt to increment the four lower bits for PORTB. Set the timer overflow rate to be 175 ms. You should increment the four lower bits of PORTB in the interrupt service routine, and leave the four upper bits of PORTB unchanged. Verify that the lower four bits of PORTB function as an up-counter.

Program 1 Program snippet.

```
            INCLUDE 'derivative.inc'
            INCLUDE 'vectors12.inc'

prog:       equ  $2000
stack:      equ  $2000

            org  prog
            lds  #stack
            movw #tof_isr,UserTimerOvf ; Set interrupt vector
            movb #$ff,DDRB  ; Port B output
            movb #$80,TSCR1 ; Turn on timer
            movb #$86,TSCR2 ; Enable timer overflow interrupt, set prescaler
                            ; so interrupt period is 175 ms
            movb #$80,TFLG2 ; Clear timer interrupt flag
            cli             ; Enable interrupts
l1:         wai             ; Do nothing - go into low power mode
            bra l1          ; Infinite loop

tof_isr:    inc PORTB       ; Increment PORTB
            movb #$80,TFLG2 ; Clear timer overflow interrupt flag
            rti
```

Suppose you want to use the Timer Overflow interrupt in your program. You would need to write a Timer Overflow ISR, with a label (say, tof isr) at the first instruction of the ISR. To set the Timer Overflow interrupt vector in your program, you would include the following instruction in your program, before the instruction which enables interrupts:

movw #tof_isr, UserTimerOvf

5. To add interrupt vector in C you will need a file listing the addresses of the interrupt service routines. Note that the different versions of the HCS12 have different interrupt vectors, and hence use different vector files. The file vectors12.h (available on the EE 308 homepage) has the appropriate vectors for the MC9S12DP256 chip. Here is part of vectors12.h:

```
#define VECTOR BASE 0x3E00
#define INTERRUPT __attribute__(interrupt))
#define VEC16(off) *(volatile unsigned short *) (VECTOR BASE + off*2)
#define UserTimerOvf _VEC16 (47)
#define UserRTI _VEC16 (56)
```

6. This tells the compiler that, for example, UserTimerOvf is a pointer to a short (16bit) number at address 0x3E5E (0x3E00 + 2*47).

7. To compile your program with interrupts, include the vectors12.h file in your program. Then, before enabling interrupts, set the interrupt vectors for those interrupts you are using. For example, if you are using the Timer Overflow Interrupt, and the name of the Timer Overflow ISR is toi isr(), put the following line in your program:

UserTimerOvf = (unsigned short)&toi_isr;

8. Calculate how long it should take for the lower bits of PORTB to count from 0x0 to 0xF and roll over to 0x0. Use a watch to measure the time. How do the two times agree?

9. Add a Real Time Interrupt to your assembly language program. Set up the RTI to generate an interrupt every 65.536 ms. In the RTI interrupt service routine, implement a johnson counter on the four upper bits of PORTB, while leaving the four lower bits of PORTB unchanged. Verify that the bit takes the correct amount of time to rotate through the four upper bits.

10. Implement the above program in C. (What you need to do to use interrupts in C is discussed in the textbook and lecture notes, and described briefly in 11 below.)

11. Change your C program so that you do not reset the Timer Overflow Flag in the Timer Overflow ISR. Does your up-counter work? Does your rotating bit work? Why?

12. Restore your original C program from Part 7. Now change your C program so that you do not reset the Real Time Interrupt Flag in the Real Time Interrupt ISR. Does your up-counter work? Does your rotating bit work? Why?

13. Restore your original C program from Part 7. Change your C program so that you do not set the address for the Timer Overflow Interrupt. Run your program. What happens now? Why?