EE 308 – LAB 6

Using the HCS12 Timer Overflow Interrupt and Real Time Interrupt

Introduction

Enabling an interrupt on the HCS12 allows your program to respond to an external event without continually checking to see if that event has occurred. Once the event occurs, the HCS12 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 programs which enable and use interrupts. Note that it is difficult (although not impossible) to simulate interrupts on the ZAP simulator, so we will not do that for this lab.

The interrupts on the HCS12 which are easiest to use are the Timer Overflow Interrupt and the Real Time Interrupt. These interrupts allow you to interrupt the HCS12 after a specified amount of time has passed.

Pre-Lab

For the pre-lab, write the programs for Sections 6 and 7. Also, calculate the time asked for in Part 5.

The Lab

1. Connect your HCS12 to your computer. At the D-Bug12 prompt, display the contents of the TCNT register. Do this several times. How do the values compare?
2. Use D-Bug12 to modify the TSCR1 register to enable the counter. Repeat Part 1.
3. Use D-Bug12 to modify the TSCR1 register to disable the counter. Repeat Part 1.
4. Start with the follwing do-nothing program:

```
prog:   equ     $1000
stack:  equ     $3C00

CODE:   section .text    ;The stuff which follows is program code
        org     prog
        lds     #stack
loop:   wai
        jmp     loop
```

   Add code to make `PORTA` an output port. Then add a Timer Overflow Interrupt to increment the four lower bits of `PORTA`. Set the timer overflow rate to be 175 ms. You should increment the four lower bits of `PORTA` in the interrupt service routine, and leave the four upper bits

of PORTA unchanged. Connect the eight pins of PORTA to the LEDs on the breadboard and verify that the lower four bits of PORTA function as an up-counter.

Note: To use interrupts in an assembly language program you will need to add a vector.s file which contains the addresses of the interrupt vectors. Here is a vector.s file for a program which uses the TOI interrupt:

```
         xdef     toi_isr  ; toi_isr is defined in another file

VECTOR:  section  .text
         org      $3e5e    ; Address of timer overlow interrupt in D-Bug 12
         dc.w     toi_isr  ; Put address of toi_isr here
```

Because the vector.s file refers to toi_isr which is in another file, you need the following in the file where toi_isr is defined:

```
        xref    toi_isr    ; Export name so vector.s can use it
```

You then need a linker file to link the main program and the vector.s program:

```
+seg .text -b 0x1000 -n .text
+seg .text -b 0x2000 -n .data
toi.o
+seg .text -b 0x3E00 -n .text
vector.o
```

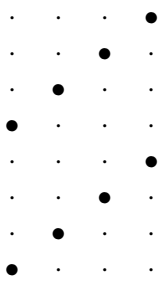
You need to assemble both the main program and the vector.s file. Here is a batch file which does this:

```
c:\cx32\ca6812 -a -l -xx -pl %1.s vector.s
c:\cx32\clnk -o %1.h12 -m %1.map %1.lkf
c:\cx32\chex -o %1.s19 %1.h12
```

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

6. 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 rotating bit on the four upper bits of PORTA, while leaving the four lower bits of PORTA unchanged. Verify that the bit takes the correct amount of time to rotate through the four upper bits.

   A rotating bit will look like this:



7. Implement the same program in C.
8. 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?
9. 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?
10. Restore your original C program from Part 7. Change your `vecdp256.c` file to put a 0 in for the address of the Timer Overlow Interrupt. Run you program. What happens now? Why?

To add interrupt vectors in C you will need a file listing the addresses of the interrupt service routines. Note that different versions of the HCS12 have different interrupt vectors, and hence use different vector files. Here is a file called `vecdp256.c` for 68HC912B32 chip:

```
/*      INTERRUPT VECTORS TABLE FOR MC9S12DP256B
 */
void toi_isr();         /* define all used ISR names here */

/* Vectors start at 0xFF80 on standard MC9S12DP256B;
 * remapped to RAM by D-Bug 12, starting at 0x3E00 */
void (* const _vectab[])() = {
                        /*                                Flash   DB12  */
        0,              /* Reserved                       (FF80) (3E00) */
        0,              /* Reserved                       (FF82) (3E02) */
        0,              /* Reserved                       (FF84) (3E04) */
        0,              /* Reserved                       (FF86) (3E06) */
        0,              /* Reserved                       (FF88) (3E08) */
        0,              /* Reserved                       (FF8A) (3E0A) */
        0,              /* PWM Emergency Shutdown         (FF8C) (3E0C) */
        0,              /* Port P                         (FF8E) (3E0E) */
        0,              /* MSCAN4 XMT                     (FF90) (3E10) */
        0,              /* MSCAN4 RCV                     (FF92) (3E12) */
        0,              /* MSCAN4 ERR                     (FF94) (3E14) */
        0,              /* MSCAN4 WAKE                    (FF96) (3E16) */
        0,              /* MSCAN3 XMT                     (FF98) (3E18) */
        0,              /* MSCAN3 RCV                     (FF9A) (3E1A) */
        0,              /* MSCAN3 ERR                     (FF9C) (3E1C) */
        0,              /* MSCAN3 WAKE                    (FF9E) (3E1E) */
        0,              /* MSCAN2 XMT                     (FFA0) (3E20) */
        0,              /* MSCAN2 RCV                     (FFA2) (3E22) */
        0,              /* MSCAN2 ERR                     (FFA4) (3E24) */
        0,              /* MSCAN2 WAKE                    (FFA6) (3E26) */
        0,              /* MSCAN1 XMT                     (FFA8) (3E28) */
        0,              /* MSCAN1 RCV                     (FFAA) (3E2A) */
        0,              /* MSCAN1 ERR                     (FFAC) (3E2C) */
        0,              /* MSCAN1 WAKE                    (FFAE) (3E2E) */
```

```
0,              /* MSCAN0 XMT                      (FFB0) (3E30) */
0,              /* MSCAN0 RCV                      (FFB2) (3E32) */
0,              /* MSCAN0 ERR                      (FFB4) (3E34) */
0,              /* MSCAN0 WAKE                     (FFB6) (3E36) */
0,              /* Flash                           (FFB8) (3E38) */
0,              /* EEPROM                          (FFBA) (3E3A) */
0,              /* SPI2                            (FFBC) (3E3C) */
0,              /* SPI1                            (FFBE) (3E3E) */
0,              /* IIC                             (FFC0) (3E40) */
0,              /* DLC                             (FFC2) (3E42) */
0,              /* SCME                            (FFC4) (3E44) */
0,              /* CRG Lock                        (FFC6) (3E46) */
0,              /* PACTLB Overflow                 (FFC8) (3E48) */
0,              /* Modulus Down Counter Underflow (FFCA) (3E4A) */
0,              /* PORTH                           (FFCC) (3E4C) */
0,              /* PORTJ                           (FFCE) (3E4E) */
0,              /* ATD1                            (FFD0) (3E50) */
0,              /* ATD0                            (FFD2) (3E52) */
0,              /* SCI1                            (FFD4) (3E54) */
0,              /* SCI0                            (FFD6) (3E56) */
0,              /* SPI0                            (FFD8) (3E58) */
0,              /* PACTLA Input Edge               (FFDA) (3E5A) */
0,              /* PACTLA Overflow                 (FFDC) (3E5C) */
toi_isr,        /* Timer Overflow                  (FFDE) (3E5E) */
0,              /* Timer channel 7                 (FFE0) (3E60) */
0,              /* Timer channel 6                 (FFE2) (3E62) */
0,              /* Timer channel 5                 (FFE4) (3E64) */
0,              /* Timer channel 4                 (FFE6) (3E66) */
0,              /* Timer channel 3                 (FFE8) (3E68) */
0,              /* Timer channel 2                 (FFEA) (3E6A) */
0,              /* Timer channel 1                 (FFEC) (3E6C) */
0,              /* Timer channel 0                 (FFEE) (3E6E) */
0,              /* Real time                       (FFF0) (3E70) */
0,              /* IRQ                             (FFF2) (3E72) */
0,              /* XIRQ                            (FFF4) (3E74) */
0,              /* SWI                             (FFF6) (3E76) */
0,              /* Illegal Instruction             (FFF8) (3E78) */
0,              /* Cop Fail                        (FFFA) (3E7A) */
0,              /* Clock Monitor Fail              (FFFC) (3E7C) */
(void *)0xff80, /* RESET                           (FFFE) (3E7E) */
};
```

To compile your program with interrupts, change you cc.bat program to look like this:

```
c:\cx32\cx6812 -vl -ax +debug crts.s %1.c vecdp256.c
c:\cx32\clnk -o %1.h12 -m %1.map %1.lkf
c:\cx32\chex -o %1.s19 %1.h12
c:\cx32\clabs %1.h12
```

The interrupt vectors for DBug-12 start at address 0x3E00. To tell the linker to put the interrupt vector file at the right place, change your `lkf` file to look like this:

```
# link command file
#
+seg .text -b  0x1000 -n .text  # program start address
+seg .const -a .text            # constants follow code
+seg .data  -b 0x2000           # data start address
crts.o                          # startup routine
lab06.o                         # application program
+seg .const -b 0x3E00
vecdp256.o
+def __memory=@.bss             # symbol used by library
+def __stack=0x3C00             # stack pointer initial value
```