EE 308 – LAB 6

Using the HC12 Timer Overflow Interrupt and Real Time Interrupt

Introduction

Enabling an interrupt on the HC12 allows your program to respond to an external event without continually checking to see if that event has occurred. Once the event occurs, the HC12 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 HC12 which are easiest to use are the Timer Overflow Interrupt and the Real Time Interrupt. These interrupts allow you to interrupt the HC12 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 HC12 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 TSCR register to enable the counter. Repeat Part 1.
3. Use D-Bug12 to modify the TSCR register to disable the counter. Repeat Part 1.
4. Start with the follwing do-nothing program:

```
STACK:  equ     $0A00
prog:   equ     $0800

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 131.072 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 a up-counter.
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.
7. Implement the same program in C.

8. Set a breakpoint at the first instruction of the Real Time Interrupt ISR. Run your program. When the program stops when it looks at the breakpoint look at the contents of the HC12 registers and stack. Identity the saved values of the return address and the Y, X, B, A and CCR registers on the stack. Verifty that the CCR saved on the stack has the I bit cleared, and the current CCR has the I bit set, preventing interrupts while in the ISR.

9. 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?

10. 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?

11. Restore your original C program from Part 7. Change your `vec_b32.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 HC12 have different interrupt vectors, and hence use different vector files. Here is a file called `vec_b32.c` for 68HC912B32 chip:

```
/*      INTERRUPT VECTORS TABLE 68HC912B32
 */
void toi_isr();         /* define all used ISR names here */

/* Vectors at 0xFFD0 on standard HC12; remapped to 0x0B10 with D-Bug 12 */
void (* const _vectab[])() = {
        0,              /* BDLC              */
        0,              /* ATD               */
        0,              /* reserved          */
        0,              /* SCI0              */
        0,              /* SPI               */
        0,              /* Pulse acc input   */
        0,              /* Pulse acc overflow */
        toi_isr,        /* Timer overflow    */
        0,              /* Timer channel 7   */
        0,              /* Timer channel 6   */
        0,              /* Timer channel 5   */
        0,              /* Timer channel 4   */
        0,              /* Timer channel 3   */
        0,              /* Timer channel 2   */
        0,              /* Timer channel 1   */
        0,              /* Timer channel 0   */
        0,              /* Real time         */
        0,              /* IRQ               */
        0,              /* XIRQ              */
        0,              /* SWI               */
        0,              /* illegal           */
        0,              /* cop fail          */
        0,              /* cop clock fail    */
        (void *)0xff80, /* RESET             */
        };
```

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

```
cx6812 -vl -ax +debug crts.s %1.c vec_b32.c
clnk -o %1.h12 -m %1.map %1.lkf
chex -o %1.s19 %1.h12
clabs %1.h12
```

The interrupt vectors for DBug-12 start at address 0x0b10. 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 for test program
#
+seg .text -b 0x0800 -n .text # program start address
+seg .const -a .text          # constants follow code
+seg .data -b 0x0900          # data start address
crts.o                        # startup routine
lab06.o                       # application program
+seg .const -b 0x0b10         # interrupt vectors start here
vec_b32.o                     # file of interrupt vectors
+def __stack=0x0A00           # stack pointer initial value
```