Lab 6: Timer Subsystem -Flag Polling and Interrupts
ECE-3731

HCS12 INTERRUPT SYSTEM

The interrupt system on the HCS12 is used for communicating with external hardware as well as for timing procedures. When an interrupt line is activated, the interrupt system stops the main program, stores it registers on the stack and then services the interrupt (runs the interrupt subroutine). Once an interrupt has been served, it will return back to the main program. You can think of the interrupt system like an alarm clock – when some condition is met (a certain time, for instance), the alarm sounds, and you immediately attend to the alarm clock. Because the interrupt system is automatic and immediate, it results in better timing control and less software work.

When an interrupt occurs, it is serviced with an interrupt service routine. An ISR is a separate piece of code, similar to a subroutine, which executes when an interrupt occurred. The ISR performs any work that needs to be done to service the interrupt, such as reading data from a parallel port. After the ISR is finished, the main program resumes.

Here's what actually happens during an interrupt. First, the HCS12 saves all the registers (A, B, X, Y, the CCR, and the PC) to the stack, so we don't need to worry about disturbing them during the ISR. This makes sense, because we don't know exactly when an interrupt is going to happen. Second, the HCS12 prevents any more interrupts from happening until the ISR is finished. This is good, because we don't want a second interrupt happening until the first one is completed. Third, the ISR executes. Finally, when the ISR is finished, the HCS12 restores the registers from the stack, and resumes the main program. The instruction that signifies the end of the ISR is the RTI instruction. *Every interrupt service routine must end with the RTI instruction.*

How does the HCS12 know where to find the ISR? When an interrupt occurs, the HCS12 looks to the jump vector table for that specific interrupt line. A jump vector is simply a space in memory that contains the address of the ISR.

Naturally, we don't want any random interrupt going off during our program, so we can control which interrupts are active or enabled. The I (interrupt) bit in the CCR serves as a master control switch for the interrupt system. When the I bit is one, all interrupts are disabled. When the I bit is zero, the interrupt system is enabled. By default, the I bit is set to one, so we have to set the I bit to zero when we want to use any interrupt. The CLI instruction clears the I bit to zero. If we want to later disable the interrupt system, the SEI instruction sets the I bit to one. Each separate interrupt line also has its own individual control bit that enables or disables it. In this way, we can control whether any interrupts are enabled, and then selectively enable only those interrupt lines that we need. This control system is similar to a circuit breaker box in a lab. There is usually a master on/off

switch that needs to be turned on before anything can get power, and then each light or area has its own individual circuit breaker.

Please see the class notes and textbook for more in-depth information and examples.

HCS12 TIMER SUBSYSTEM

The HCS12 has a standard timer module that is built around a 16-bit timer counter which is clocked by submultiple of the bus clock, using the prescaler.
It provides 8 channels of input capture or output compare. It also has a pulse accumulator and can do pulse width modulation.
In this lab we are mainly concerned with output compare and timer overflow.

Output Compare
-waits for the 16-bit timer (TCNT register) to be equal to a value in a register (TOC time of compare register) -an output compare flag is set when this happens
-optionally generates an interrupt

Timer Overflow
-a flag is set (TOF timer overflow flag) whenever the 16-bit TCNT register overflows.
-optionally generates an interrupt

Flag Polling
-the flags are associated with hardware events are always being set to indicate the status. When interrupts are enabled, which is done by setting interrupt mask bits and interrupts are globally enabled (cli instruction) then an interrupt can occur in response to the hardware event.
We may, if we wish, choose to not enable interrupts, but instead have the program repeatedly query the status of a flag in a "flag polling loop". In this way we can generate a time delay. (The program cannot move forward until the flag becomes set causing the flag polling loop to end.)

(See assignment on next page)

ASSIGNMENT

To be done in Assembly

This program is to have a main program loop and one interrupt service routine.

(i) Main Program Loop:

This generates a tone (any frequency) on the speaker. The speaker is on Port T pin 5 (PTT5).
This is to be done by toggling the speaker followed by a delay (done repeatedly in an endless loop.
The delay is to be created by using a delay subroutine. This subroutine must use the timer overflow flag (TOF**) in flag polling mode**. (Doing this any other way will lose points.)
NOTE: If your speaker is not working, check the jumpering of your board. It may be that your board is jumpered to have the speaker on port P pin 5.

(ii) Interrupt Service Routine:

You will maintain a 4-bit counter on the Port B LED's, as was done in the previous lab.
You will use an output compare channel in interrupt mode (e.g. OC2) to update the value of the counter.
You will program **an action to occur once every second using an output compare interrupt**. The interrupt service routine will update the counter by incrementing it by one. (This is happening once every second). Your 4-bit counter overflows to zero after being incremented to all ones, as in the previous lab.

HINT:
You will need to set up an interrupt vector for your ISR (interrupt service routine).
This is done in the Interrupt Vectors section at the end of your Codewarrior Project.

```
;*****************************************************************
;*              Interrupt Vectors                *
;*****************************************************************
        ORG   Vreset
        DC.W  Entry        ; Reset Vector

; PUT YOUR OUTPUT COMPARE VECTOR HERE
```