

Interrupt Processing

N. Narasimhamurthi

1 Objective

To become familiar with interrupt processing.

2 Background

In an earlier lab, you had to generate a 30.52/2 Hz square wave signal on PA4 pin. The code for generating the square wave is given below for your reference. Make sure that you run the program and verify that you get the square wave before proceeding further. Also, to understand this lab, you must connect the PA4 pin to a oscilloscope and see the square waves.

```
; Various defines go here ...
    ORG $3000 don't forget the $

ME    FCC /Your name/
      FCB 10
      FCC /ECE 372/
      FCB 10
      FCC /Date the program was last changed/
      FCB 10, 10, 4

      ORG $2000 DONT FORGET THE $
      LDX #ME
      JSR OUTSTRG ; MAKE SURE YOU HAVE EQU FOR OUTSTRG

LOOP1

; CLEAR THE FLAG. NO HARM IS DONE IF IT IS ALREADY CLEARED
      LDAA #%10000000
      STAA TFLAG2
```

```

;WAIT FOR THE FLAG TO BE SET

LOOP2
    LDAA TFLAG2
    ANDA #%10000000
    BEQ LOOP2

;NOW TOGGLE PA4
    LDAA #%00010000
    EORA PORTA
    STAA PORTA

;DO IT ALL OVER AGAIN
    BRA LOOP1

```

3 Interrupts

If you study the above code, most of the time is spent waiting for the clock to rollover (the loop at `LOOP2`). This is a lot like sitting in front of the clock and watching and waiting for the clock to rollover. Or, for that matter, sitting in front of a stove and watching and waiting for kettle to boil, or watching food being cooked in a microwave oven. A better solution would be to go about ones job and arrange matters so that one is told when an event occurs (the clock chimes, the kettle whistles, the microwave oven sounds an alarm etc.). When we are told that the event has occurred, we then take appropriate action (turn off the stove, take food out of the oven etc.). Most of the HC11 interrupts work the same way. In essence this is what happens:

1. When an event occurs, a flag is set. For example, the clock rollover sets the `TOF` flag.
2. Associated with the flag is a masking bit. The name of the bit is the same as the flag except the final `F` is replaced by `I`. The mask associated with `TOF` is `TOI`.
 - (a) If the mask is zero, then nothing much happens. The event is ignored by the interrupt processing structure.
 - (b) However, if the mask is set, then request for service is generated.
 - i. The `I` bit in the `CCR` is a master disable switch. If this is set (by using the command `SEI`), then the request for service does not

interrupt the computer and is hence ignored.

- ii. However if I bit is cleared (by using the command CLI), then the CPU is interrupted.

Note: It is extremely important that you have an SEI before any code that can turn on an interrupt and and CLI after all relevant and required initialization is performed.

3. When a CPU is interrupted, it stops its current task and starts the service.
4. When performing the service, you get a completely new set of registers. So you can not assume that the registers will have any specific value. Also, when the service terminates, the new set of registers is destroyed. So you can not assume that the rest of the code can see what you stored in the register as part of the service. In fact, the interrupted task would be oblivious to the fact that a service was provided. The only way it can find out is if the service modifies some memory location.
5. The location of the service that is associated with a particular interrupt is defined by the hardware manufacturer, and is called the *jump vector*. This would be in read only memory and can not be changed. The operating system, BUFFALO, sets the start of the service to a known location and sets aside three bytes at that location. Your service will start with JMP instruction to the actual code for the service. Your service **must end with the RTI instruction**.

The address of services to three important interrupts is given in the following table:

Interrupt	Service location
TOF	\$00D0
RTIF	\$00EB
OC2F	\$00DC

6. Your service code should, as part of the service, *turn off* the flag that generated the interrupt. If not, the request for service will still be active and will generate a new service request as soon as the current service end!

Here is a short checklist for what you should do:

1. In your main routine, enable an interrupt by turning on the associated mask.
2. Write the service routine. As part of the service make sure you turn off the flag that generated the interrupt.

3. Let HC11 know where to find the service. In other words, Link the service to the request.

With this background, we will modify the square wave generator to use the interrupt. Here is the complete code with some of the standard equates left out (you need to have them at the top of the file!). You should compare this code with the earlier one. In this code, the main program does nothing really interesting. It just prints a series of Z's to the screen

```

; Various defines go here ...
    ORG $3000 don't forget the $

ME    FCC /Your name/
      FCB 10
      FCC /ECE 372/
      FCB 10
      FCC /Date the program was last changed/
      FCB 10, 10, 4

      ORG $2000 DONT FORGET THE $
      LDX #ME
      JSR OUTSTRG ; MAKE SURE YOU HAVE EQU FOR OUTSTRG

LOOP1

; Enable TOF interrupt by setting TOI (bit#7 in TMSK2)
      SEI
      LDAA #%10000000
      STAA TMSK2
      CLI

; Now go about your business of printing Z's
      LDAA #'Z'
LOOP  JSR OUTA
      BRA LOOP

; End of main program

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; INTERRUPT SERVICE

```

SERVICE

```
; TOGGLE PA4
    LDAA #%00010000
    EORA PORTA
    STAA PORTA

; TURN OFF THE FLAG!
    LDAA #%10000000
    STAA TFLG2

; END WITH AN RTI
    RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Connect the service to the interrupt ;

    ORG $00D0 ; $00D0 WHERE THE SERVICE STARTS
    JMP SERVICE ; JUMP TO WHERE THE SERVICE CODE ACTUALLY IS
```

How does this code differ from the previous one? We don't wait for the clock to rollover. Instead, the main program goes about its task (in this case not a very interesting one!). Note that although the service routine uses the **A** register, this does not affect the value the main routine has stored in the **A** register (the character Z).

4 The Real time interrupt

The real time interrupt, RTI (not to be confused with the RTI instruction) acts exactly like the timer overflow interrupt, except you can control the time between interrupts using the last two bits (0 and 1) of PACTL at location \$1026. If the both the bits are set, the time between the interrupts is 32.768 ms, i.e. same as timer overflow. However, you can decrease the time between the interrupts (increase the rate) by changing the last two bits as PACTL as shown below:

Last two bits of PACTL	Time between interrupts
00	4.096 ms (244.1 Hz)
01	8.192 ms (122 Hz)
10	16.384 ms (61 Hz)
11	32.768 ms (30.5 Hz)

Thus if we want to use the RTI interrupt, we have to change the above code as shown below. Note the crucial differences:

```
; Various defines go here ...
    ORG $3000 don't forget the $

ME    FCC /Your name/
      FCB 10
      FCC /ECE 372/
      FCB 10
      FCC /Date the program was last changed/
      FCB 10, 10, 4

      ORG $2000 DONT FORGET THE $
      LDX #ME
      JSR OUTSTRG ; MAKE SURE YOU HAVE EQU FOR OUTSTRG

LOOP1

; Enable RTIF interrupt by setting RTII (bit#6 in TMSK2)
    SEI
    LDAA #%01000000 <= This is different
    STAA TMSK2

    LDAA #%00000011 <= THIS IS NEW
    STAA PACTL

    CLI

; Now go about your business of printing Z's
    LDAA #'Z'
LOOP  JSR OUTA
      BRA LOOP

; End of main program

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; INTERRUPT SERVICE

SERVICE
```

```

; TOGGLE PA4
    LDAA #%00010000
    EORA PORTA
    STAA PORTA

; TURN OFF THE FLAG!
    LDAA #%01000000 <= This is different
    STAA TFLG2

; END WITH AN RTI
    RTI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Connect the service to the interrupt ;

    ORG $00EB ; $00EBO WHERE THE SERVICE STARTS
    JMP SERVICE ; JUMP TO WHERE THE SERVICE CODE ACTUALLY IS

```

4.1 Exercises

1. Modify the above program to generate a 30.5 Hz square wave by setting the RTI interrupt rate to 61 Hz.
2. After you made the modification, toggle PA4 every 61 interrupts (Hint, set up a counter and initialize it to 61 in the main program. In the interrupt service, decrement the counter. When the counter reaches zero, toggle the pin and reset the counter back to 61). Verify that the signal you generate is a 0.5 Hz square wave
3. Create a simple clock. In addition to toggling the pin, increment an 8-bit variable called TIME. In the main loop, instead of printing Z's, print the variable using OUT1BSP.

5 The output compare interrupt

The HC11 has 5 OCx interrupts. These are like alarm clocks. You set a desired 'alarm' time and when the clock matches the alarm setting, the OCxF flag will be turned on and could then generate a request for service. Note that if you do not change the alarm setting, you will still get an interrupt every 32.768 ms. However,

having the alarm gives you greater flexibility. First a code that does not change the alarm setting and hence generates 30.5 Hz square wave.

```
; Various defines go here ...
    ORG $3000 don't forget the $

ME    FCC /Your name/
      FCB 10
      FCC /ECE 372/
      FCB 10
      FCC /Date the program was last changed/
      FCB 10, 10, 4

      ORG $2000 DONT FORGET THE $
      LDX #ME
      JSR OUTSTRG ; MAKE SURE YOU HAVE EQU FOR OUTSTRG

LOOP1

; Enable OC2 interrupt by setting OC2I (bit#6 in TMSK1)
    SEI
    LDAA #%01000000 <= This is different
    STAA TMSK1

    CLI

; Now go about your business of printing Z's
    LDAA #'Z'
LOOP  JSR OUTA
      BRA LOOP

; End of main program

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; INTERRUPT SERVICE

SERVICE

; TOGGLE PA4
    LDAA #%00010000
    EORA PORTA
```


; END WITH AN RTI
RTI