CpE 390 Microprocessor Systems

Lab 4: Interrupts

1. Introduction

Interrupts allow a processor to perform multiple tasks simultaneously. An interrupt is typically generated by a hardware source such as a UART, a timer or peripheral that is ready to transfer data. When an interrupt is triggered, the main program is suspended and the processor examines the Interrupt Vector Table (IVT) to find the memory address of the Interrupt Service Routine (ISR) that has been designated to service this interrupt. Each interrupt source is allocated a specific location in the IVT. Once the address of the ISR has been determined, the processor will jump to that location.

An ISR is much like a regular subroutine, except that context switching is handled automatically by the processor. Context switching is the process of saving all the registers on the stack so that the operation of the main program can continue once the ISR has completed its task. A "return from interrupt" instruction (**rti**) at the end of the ISR restores the CPU registers to values they had before the interrupt and then returns control to the main program.

The other difference between an interrupt and a regular subroutine call is that an interrupt can happen at any time, whereas a subroutine call only happens at well-defined places in the user code. When writing an ISR, it's important to remember that you do not know where in the main program an interrupt will occur, and to make sure that unexpected behavior does not occur. For this reason, you cannot use registers to communicate between an ISR and the main program.

In this lab session, we will program the processor to do two tasks simultaneously. The main program will read characters from the serial input (which comes from your keyboard) and echo them to the serial output (which displays on the monitor output window). At the same time, the Real-Time Interrupt system (RTI) will be set up to interrupt the processor at regular intervals and toggle the outputs of the one of the processor's parallel ports.

2. Main Program

(a) Enter the following code into the MiniIDE, assemble and load it into the processor:

```
SCOSR:     EQU   $00CC              ; Serial Communications Status Register
SCODR:     EQU   $00CF              ; Serial Communications Data Register
ESC:       EQU   $1B                ; ASCII ESC character
CR:        EQU   $0D                ; ASCII CR character (ENTER key)
LF:        EQU   $0A                ; ASCII LF (line feed)
```

```
            ORG     $4000
getch:      brclr   SCOSR, #$20, getch     ; wait for RDRF bit to be set
            ldaa    SCODR                  ; read character from serial input into A
            cmpa    #ESC                   ; test for escape character
            beq     done                   ; if found, return to monitor
            cmpa    #CR                    ; if character is CR
            bne     putch
            ldaa    #LF                    ; change echo character to LF
putch:      brclr   SCOSR, #$80, putch     ; wait for TDRE bit to be set
            staa    SCODR                  ; echo character back to serial output
            bra     getch                  ; get next character
done:       swi
```

This main program is an (almost) infinite loop that echoes characters sent to the microcontroller by the MiniIDE serial terminal. We have not yet covered the serial communication interface in lectures. But briefly, the program waits for the Read Data Register Full flag (RDRF) in the Serial Communications Status Register (SCOSR) to be set. This indicates that an input character has been received by the interface. The program loads this character into Accumulator A by reading the Serial Communications Data Register (SCODR). Its then tests the character to see if it is an Escape (ESC) character. If it is an ESC character, the program exits and returns to the monitor via an **swi** instruction. If it is a CR character, it changes the echo character to be LF. This is done so that when you hit the ENTER key, the output display will go to the next line. The program then checks the SCOSR and waits for the Transmit Data Register Empty flag (TDRE) to be set indicating that the output interface is ready to receive another character. The input character is then output (echoed) to the serial terminal by writing to the SCODR. The program then loops back to get the next character.

(b) Run the program. With the cursor in the monitor output window, type some characters and verify that they are echoed back by the microcontroller. Also check that you can exit the program by typing an ESC character.

3. Interrupt Service Routine

(a) Add the following constant declarations to the beginning of your program:

```
PTT:        EQU     $240        ; Port T Data
DDRT:       EQU     $242        ; Port T Direction Register
CRGFLG:     EQU     $37         ; CRG Flag Register
```

```
CRGINT:    EQU        $38          ; CRG Interrupt Register
RTICTL:    EQU        $3B          ; RTI Control Register
```

(b) Add the following initialization code to the beginning of your main program (i.e., right before the *getch* label):

```
main:      bset       DDRT, $80                 ; set up PT7 to be output
           bclr       PTT, $80                  ; set PT7 data to be '0'
           movw       #rtisr, $0FF0             ; set up RTI interrupt vector
           movb       #$4B, RTICTL              ; set RTI timeout to 98,304 cycles
           bset       CRGINT, $80               ; enable rti (local) interrupt
           cli                                  ; enable (global) interrupts
```

This code sets bit 7 of Port T (PT7) to be an output pin and sets its value to zero. It then loads the ISR address *rtisr* into the RTI user interrupt vector table location $0FF0. The code then sets the time out period to be $12 \times 2^{13} = 98{,}304$ crystal oscillator clock cycles by loading $4B into the RTI Control register. Interrupts are then enabled by first setting the local RTI Interrupt Enable bit in the CRG Interrupt Register and then turning on maskable interrupts by clearing the I bit in the CSR Register.

(c) Add the following Interrupt Service Routine to the end of your program:

```
rtisr:     bset       CRGFLG, $80               ; clear RTI interrupt flag
           brset      PTT, $80, ptclr           ; if PT7='1', then clear
           bset       PTT, $80                  ; if not, then set PT7
           rti                                  ; return from interrupt
ptclr:     bclr       PTT, $80                  ; clear PT7
           rti                                  ; return from interrupt
```

This is the code that runs whenever an interrupt occurs. It first clears the RTI interrupt flag by setting the MSB of the CRG Flag register. This ensures that the RTI will not interrupt again until it next "times out". It then tests the current value of PT7, If it is currently a '1', then it clears it to '0'. Otherwise, it sets it to '1'. In other words, every time the interrupt service routine runs, it toggles (complements) the output value PT7. Finally, an **rti** instruction returns control to the main program.

(d) Assemble, load and run the program. Check that is still echoes characters typed into the Monitor output window.

(e) The RTI module has been set up to interrupt the main program once every 98,304 crystal oscillator clock cycles. The EVB boards use a 16 MHz crystal oscillator. This means that an interrupt will occur every 6.144 ms. Each interrupt should toggle PT7. Connect the oscilloscope to pin PT7. You should see a square wave with a period of (2 x 6.144) ≈ 12.3

ms.. Note that the microcontroller is generating this square wave and echoing characters "simultaneously". Sketch the waveform you see showing time and voltage values.

4. Switching the on-board LED

(a) On the EVB board, you will see two light emitting diodes, marked LED1 and LED2. LED1 is connected to PS2 (pin 2 of Port S) and LED2 is connected to PS3. Modify your program so that in addition to toggling PT7, it also toggles PS2. You will need the following constants:

```
DDRS:       EQU   $24A        ; Port S data direction register
PTS:        EQU   $248        ; Port S data register
```

Be careful not to disturb bits PS0 and PS1 – they are used to maintain the serial communication between the EVP and your host PC.
You should see the LED1 now glowing dimly. It is actually turning on and off every 6.4 ms – too fast for your eye to see – so it looks like it is "half on".

(b) Now slow down the flash rate by only changing the state of the LED once every 100 interrupts. You can do this by making a global variable to count the number of interrupts. At the beginning of the main program, initialize its value to zero. Then every time an interrupt occurs, increment the variable and check to see if it is equal to 100. If not, just return from interrupt. If it is interrupt #100, then change the state of the LED and reset the interrupt count to zero.

5. Communicating between the Main Program

Modify your program so that when you type the letter **f** (meaning fast), it speeds up the flashing by changing the divide ratio to 50 and when you type the letter **s** (meaning slow) , it restores the divide ratio to 100.
You can do this by setting up another named global variable which will hold the divide ratio (50 or 100). Initialize this to be 100. In the ISR, use this value to test the interrupt count. In the main program, change this value whenever an 's' or 'f' is typed.

6. Lab Report

Include your program(s) and execution results. Sketch the waveform you obtained from PT7 in part 3, showing voltage and time values.