Lab 05: *Pardon the Interruption…* Using Hardware Interrupts
Jonathan Gusler
2/19/2019

Overview
This lab's primary purpose was to introduce the use of hardware interrupts. This included both setting up the PORTS and multiple layers of Enable Interrupt registers to define explicitly what kind of interrupts are allowed.

Background
This lab requires knowledge of Atmel assembly language, what a hardware interrupt is, a working knowledge of what happens in the program when a hardware interrupt occurs, how to initialize interrupts in the code, and binary numbers.

Discussion
This lab involves three primary hardware components:

1) A joystick
2) A blinking LED
3) A set of four LED's to represent 4-bit numbers

In the lab, a joystick controlled the blink frequency of the BOOT LED and a group of four LEDs which displayed frequency using a 4-bit number. The frequency was limited to 1 to 15 Hz. Whenever the joystick was pushed up or down, one interrupt of two, based on the direction of the push, was generated in the blink loop. This interrupt caused the frequency to increase or decrease, and the 4-bit number frequency readout to change accordingly. These interrupts also checked the current frequency and would not change the frequency if the new frequency was greater than 15 or less than one. A secondary feature in the interrupts was a loop that prevented the interrupt from triggering multiple times, when the user only pushed once, due to the bouncing of the joystick, a problem inherent in the mechanical system that was the circuit completed by the joystick being pushed. The loop lasted long enough such that the blink frequency was only updated after the bouncing stopped. Table 1 shows the ports and relevant bits that were used and what they were used for.

PORTA.7	Output, The blinking LED
PORTB.1 and PORTB.3	Input. Joystick Up and Down, respectively
PORTC3:0	Output, the 4-bit frequency readout
PORTD.1 and PORTD.3	Input, connected to PORTB to generate interrupt

Table 1: List of ports being used and their purpose.

A point of explanation, PORTB cannot generate interrupts, but PORTD can. The signals from PORTB were wired to PORTD so that the joystick could generate interrupts indirectly.

From a code perspective, the interrupts needed to be initialized such that the system only allowed interrupts from PORTD.1 and PORTD.3 and only on the falling signal edge. The code for the whole program is shown in Appendix A.

Analysis and Results
The code did perform the intended actions. The blink frequency of the BOOT LED increased by 1 Hz when the joystick was pushed up but did not exceed 15 Hz. The blink frequency of the BOOT LED decreased by 1 Hz when the joystick was pushed down but did not go below 1 Hz. The 4-bit readout always showed the current frequency. Doing anything else with the joystick did not cause anything to occur.

Summary and Conclusions

The lab was successful in introducing the use of hardware interrupts and the code performed as intended.

Appendix A

```
/*************************************************************************
 *         File: Lab05.asm
 *  Lab Name: Pardon the Interruption...
 *    Author: Dr. Greg Nordstrom
 *   Created: 02/14/2019
 * Processor: ATmega128A (on the ReadyAVR board)
 *
 * Modified by: Jonathan Gusler
 * Modified on: 2/15/2019
 *
 * This program blink the BOOT LED at a rate ( 1-15 Hz) that is changed
 * in increments of 1 Hz by the joystick on
 * the ATmega128A microcontroller and also uses another set of 4 LEDs
 * to display the blink rate in binary. The changes to rate are executed using
 * ISRs which are triggered by interrupts from the joystick
 *
 *************************************************************************/

.org 0x0000                  ; next instruction address is 0x0000
                             ; (the location of the reset vector)


.def BlinkRateReadout = R17  ; stores binary readout
.def BlinkFreq                = R20  ; holds current blink rate (1-15 Hz)
.equ BlinkFreqMin      = 1
.equ BlinkFreqMax      = 15
.equ InitialBlinkFreq = BlinkFreqMin

rjmp main                   ; allow reset to run this program

; Define two interrupts
.org 0x0004                                ; Joystick up
       jmp ISRJoystickDown

.org 0x0008                                ; Joystick down
       jmp ISRJoystickUp


;---------------------------------Initialization stuff
main:                       ; jump here on reset
    ldi R16, HIGH(RAMEND)   ; initialize stack (default RAMEND = 0x10FF)
    out SPH, R16
    ldi R16, LOW(RAMEND)
    out SPL, R16

       ldi R20, 1                ; Init BreakFreq to 1 Hz
       ldi R17, $0E           ; Init Readout to 1

       ; Set the BOOT LED register status as output and turn it off initially
    ldi R16, (1<<DDA7)      ; set PORTA.7 pin as output via bit 7
    out DDRA, R16           ; in PORTA's data direction register

    sbi PORTA, PORTA7       ; turn BOOT LED off (active low) by setting PORTA.7

       ; Set Joystick up and down as inputs
```

```
       ldi R16, (1<<DDB1) | (1<<DDB3)
       out DDRB, R16

       ; Set up internal pull-ups for PORTB on pins 1 and 3
       ldi R16, (1<<PB1) | (1<<PB3)
       out PORTB, R16

       ; Set PORTC.3:0 as output for the 4-bit frequency indicator and set to 1 initally
       ldi R16, (1<<DDC0) | (1<<DDC1) | (1<<DDC2) | (1<<DDC3) ; R16 = 00001111
       out DDRC, R16

       out PORTC, BlinkRateReadout

       ; Set portD as input to trigger ISRs
       ldi R16, (1<<DDD1) | (1<<DDD3)
       out DDRD, R16

       ; define the doors for the interrupts
       ldi R16, (1<<ISC11) | (1<<ISC31); set interrupts to occur when signal goes from high to low
       sts EICRA, R16
       ldi R16, (1<<INT1) | (1<<INT3)    ; only allow interrupts from INT1 and INT3
       out EIMSK, R16



;------------------------------ MAIN LOOP
mainLoop:
       sei
    cbi PORTA, PORTA7       ; turn BOOT LED on (active low) by clearing PORTA.7

    ; kill some time
    ldi R16, 16                     ; R16 is outer loop counter
       sub R16, BlinkFreq
outer_loop1:
    ldi R24, low($FFFF)     ; load low and high parts of R25:R24 pair with
    ldi R25, high($FFFF)    ; loop count by loading registers separately
    inner_loop1:
        sbiw R24, 1         ; decrement inner loop counter (R25:R24 pair)
        brne inner_loop1    ; loop back if R25:R24 isn't zero
    dec R16                 ; decrement the outer loop counter (R16)
    brne outer_loop1        ; loop back if R16 isn't zero

    sbi PORTA, PORTA7       ; turn LED off (active low) by setting PORTC.3

    ; kill some more time
    ldi R16, 16                     ; R16 is outer loop counter
       sub R16, BlinkFreq
outer_loop2:
    ldi R24, low($FFFF)     ; load low and high parts of R25:R24 pair with
    ldi R25, high($FFFF)    ; loop count by loading registers separately
    inner_loop2:
        sbiw R24, 1         ; decrement inner loop counter (R25:R24 pair)
        brne inner_loop2    ; loop back if R25:R24 isn't zero
    dec R16                 ; decrement the outer loop counter (R16)
    brne outer_loop2        ; loop back if R16 isn't zero

    rjmp mainLoop           ; play it again, Sam...
```

```
;----------------------------ISR for joystick up
ISRJoystickUP:
	ldi R18, 5
Interrupt_delay_loop1:                                  ; delay to compensate for bounce
	ldi R26, low($FFFF)
	ldi R27, high($FFFF)
	Interrupt_delay_loop2:
		sbiw R26, 1
		brne Interrupt_delay_loop2
	dec R18
	brne Interrupt_delay_loop1

	cpi BlinkFreq, BlinkFreqMax
	breq ExitJoystickUp                     ; if at max, don't do anything
	inc BlinkFreq                           ; increment BlinkFreq
	dec    BlinkRateReadout                 ; increment binary readout
	out PORTC, BlinkRateReadout
	reti

ExitJoystickUp:
	reti
;----------------------------ISR for joystick down
ISRJoystickDown:
	ldi R18, 5
Interrupt_delay_loop3:                                  ; delay to compensate for bounce
	ldi R26, low($FFFF)
	ldi R27, high($FFFF)
	Interrupt_delay_loop4:
		sbiw R26, 1
		brne Interrupt_delay_loop4
	dec R18
	brne Interrupt_delay_loop3

	cpi BlinkFreq, BlinkFreqMin
	breq ExitJoystickDown                           ; if at min, don't do anything
	dec BlinkFreq
	inc    BlinkRateReadout                 ; increment binary readout
	out PORTC, BlinkRateReadout
	reti

ExitJoystickDown:
	reti
```