Lab Assignment 5—*Pardon the Interruption…*
Using Hardware Interrupts

Overview:

In this lab you'll write assembly code that uses the ReadyAVR's **joystick** to control the blink rate of an on-board LED. Specifically, you'll enable the joystick to generate *hardware interrupts* that trigger the ATmega128A processor to run an interrupt service routine (ISR) which updates a register used by the main program to control the LED's blink rate. Also, the ISR will cause four other on-board LEDs to show the current blink rate (~1 to 15 Hz) in 4-bit binary. See files **EECE3624_Lab05_Demo.mp4** and **EECE3624_Lab05_Notes.pdf** on Canvas for more info.

Lab Procedure:

Begin by accepting the Lab05 assignment. After the remote assignment repo has been created for you, open **Git Shell** and navigate to your `D:\EECE3624Labs` solution directory. Enter the following command to clone the remote `Lab05` assignment repo to your local storage device.

```
D:\EECE3624Labs> git clone <URL-to-remote-assignment-repo> Lab05
```

Add the new `Lab05` project to your `EECE3624Labs` solution using **Atmel Studio**, and remember to set `Lab05` as the new startup project and `Lab05.asm` as the entry file.

Open the `Lab05.asm` file and notice that it's the same code you used to begin your `Lab02` assignment a few weeks ago. Make the following changes to the file:

1. **Update the program header** to reflect the new program's operation. Mention that the program blinks the "BOOT" LED (connected to **PORTA.7**) at a rate of ~1 to 15 Hz, adjustable in 15 steps, as the joystick is toggled up (faster) or down (slower), and that the blink rate is displayed in 4-bit binary on LEDs 0-3, which are connected to **PORTC.0 (LSB) through PORTC.3 (MSB)**. *NB:* The blink rate and the display must only update after the joystick is ***released***. Pressing the joystick up and *holding it* should have no effect until released. Only then should the rate increase by one (assuming, of course, that it wasn't already at 15, in which case nothing should change).
2. Create and use these define and equate statements to make your code more readable:

```
.def BlinkFreq        = R20 ; holds current blink rate (1-15 Hz)
.equ BlinkFreqMin     = 1
.equ BlinkFreqMax     = 15
.equ InitialBlinkFreq = BlinkFreqMin
```

3. Since you'll be using PORTA.7 for the blinking LED and PORTC.3:0 for displaying the 4-bit blink rate, configure those five pins as **output pins** via the DDRA and DDRC registers. The remaining pins should be set as inputs. You'll need to update the `sbi` and `cbi` instructions as well. And be sure to **update any and all related code comments**.
4. In both outer delay loops, don't load R16 with 40, but rather with 16_{10} − `BlinkFreq`. (Remember, `BlinkFreq` is updated by the ISRs and holds the most current blink rate.) Loading R24 and R25 with $FFFF instead of $4000 should give an ~1Hz blink rate when `BlinkFreq` has a value of 1. Your program must default to a ~1Hz blink rate at startup.
5. **Test that the "BOOT" LED actually blinks at about 1 Hz** with a 50% duty cycle. **DO NOT GO ON UNTIL THE LED BLINKS AT ABOUT 1 HZ.**

Next, design and implement the **Interrupt Service Routines** (ISRs) that will be executed whenever the joystick is toggled up or down. The following **pseudocode** gives a general idea of how the ISRs should behave.

```
Initialize the interrupt system (explained in class and in your book)

mainLoop
   Blink PORTA.7 LED on and off once according to the value in BlinkFreq
   goto mainLoop

ISRJoystickUp (invoked when joystick is released after being pressed up)
   if BlinkFreq < BlinkFreqMax then
      increment BlinkFreq
      display new blink rate on the PORTC.3:0 LEDs
   return from interrupt

ISRJoystickDown (invoked when joystick is released after pressed down)
   if BlinkFreq > BlinkFreqMin then
      decrement BlinkFreq
      display new blink rate on the PORTC.3:0 LEDs
   return from interrupt
```

As shown above, the main loop continuously blinks the PORTA.7 LED at a rate determined by the current value of `BlinkFreq`. The key is that **`BlinkFreq` is changed by the ISRs—the main program never changes `BlinkFreq`**. And note that **all both ISRs do is change the value of BlinkFreq and update the four LEDs!**

The joystick is actually a set of five buttons. The two that represent "up" and "down" are connected to ATmega128A pins PORTB.3 and PORTB.1, respectively. However, **PORTB pins cannot generate interrupts, but PORTD pins can**, so use PORTD.3 (for "up") and PORTD.1 (for "down") to actually generate the necessary interrupts when their inputs go **from low to high**. Of course, PORTD.1 and PORTD.3 are not internally connected to the joystick switches, so **you'll have to connect PORTB.1 to PORTD.1 and PORTB.3 to PORTD.3 using two external jumper wires**. This way, the joystick signals will actually cause the "up" and "down" ISRs to be invoked.

Assuming everything has been properly initialized, and your main program is happily blinking the "BOOT" LED, here's how the ISRs are used to change the blink rate:

1. The user presses and releases the up or down joystick button, causing a low-to-high transition on either pin PORTB.1 or PORTB.3 (which are connected externally to pins PORTD.1 or PORTD.3). **Don't forget to enable pull-ups on the two PORTB pins!**
2. The low-to-high transition on PORTD.1 or PORTD.3 generates an interrupt which in turn causes the ATmega128A to invoke the corresponding ISR which increments or decrements the `BlinkFreq` value and updates the 4-bit blink rate on the four LEDs.
3. The ISR ends and control returns to the main program, where the PORTA.7 LED begins blinking at the new rate. Simple, right?

The key to success in this lab is **understanding how to configure ports and interrupts** on the ATmega128A. Your instructor should have already presented some material to get you started, but your "go to" text for such information is the **AVR ATmega128A data sheet**. Specifically, you should refer to the chapters on "Interrupts," "External Interrupts," and "I/O Ports."

Report:

This is a **one week lab**, and everything must be completed **by lab on Friday, Nov 7th**. In this case, "everything" includes:

1. A live demonstration of your working hardware and code (in class on the 7th),
2. Your PDF lab report (which must include a printout in of your code in an appendix) submitted via Canvas, and
3. Your code pushed to your GitHub repository.

As always, you'll be graded on the correctness, completeness, readability and organization of your report and your code, so make sure everything looks professional! And, of course, have fun! (And if you're lucky, you'll be able to complete this lab without too many interruptions…)