Interrupts

Performance Check

______ (30) Timer 0 Flash at 500ms

______ (30) ADC 8MSB on LED

______ (30) Volume knob on LED

______ (40) Mode operations with all functions operate properly (bonus, no late submission for bonus)

Objectives:

- The purpose of this laboratory is to become familiar with implementing interrupts.
- Develop a program in stages that utilizes interrupts as follows:
  - Replace the Timer0 1mS polling delay with interrupt delays using the interrupt
  - Replace the ADC polling function that returns a 16bit unsigned value with an interrupt based solution.
  - Develop a simple program using external interrupts by connecting encoder to INT0 and an input
  - Use an encoder to control LED bars, similar to a volume control knob.
  - Use external interrupt to control the information display by the LED: Timer flash, ADC value, or encoder information.

Introduction

Lab 6 Timer and PWM and Lab 7 ADC used polling method to accomplish the tasks assigned. When using polling method, the controller is not doing anything else except waiting for the corresponding interrupt flag to be set before the next command. This is usually not the best way to use the microcontroller's time. Imagine the controller is waiting in a delay function while a critical spike happened at one of the ADC channel. This ADC value may be missed by the time the delay is completed. To solve this, we can use interrupts. An interrupt service routine (ISR) is executed when the corresponding interrupt is triggered (interrupt flag is set). The tasks in this lab are to convert the codes using polling methods to interrupt.

At times, you may have an input connected to external interrupt pins. These interrupts are triggered externally. They can be configured to be triggered at falling edge or rising edge. External interrupt may be used for many applications. For examples, you may use it to wake up the controller for better power management. You may also use it to trigger an emergency routine for alarms. When an external interrupt is enabled and is triggered, the code inside the service routine that corresponds to the particular interrupt pin will execute.

Encoder

In this lab, you will use an encoder that connects to an external interrupt pin. There are two inputs from the encoder, one is connected to the external interrupt pin and another to a normal general purpose IO. When the interrupt is triggered by the falling edge at the external input pin, the status of the general purpose input determines the direction of the encoder. To avoid false trigger, consider using a capacitor to filter the noise at the input pins (connect a 0.1uF capacitor between the DT and ground and another 0.2uF capacitor between CLK and ground. Use the oscilloscope to monitor both pins while turning the knob for better understanding of the encoder's operation. Watch the video in BS, in the Lab 8 folder illustrating the waveforms of the CLK and DT waveforms in the oscilloscope.

External Input

Have you notice in previous labs, the pushbutton input sometimes are not responsive. Especially when you have delay functions. This is because the delay functions is taking up the resource and the code for the pushbutton can only execute when you press the pushbutton down for a long time. Using the external interrupt input can solve this problem. You will see the flashing LED is flashes in other modes and the pushbutton input is more responsive. The pushbutton input is used to switch the LED to display different information: flashing LED, ADC value, or encoder information. Watch the lab video to observe all the operations for this lab.

Before writing your code

1. Ensure that you have a working copy of the code developed in Lab 6 & 7 in which the 10bit result (0-1023) of an ADC conversion was used to control PWM duty cycle. We will use the ADC module and the millisecond timer.
2. Fill in the hardware connections **Table 1** below and in all your comment headers in your code:

Hardware connections

Table 1: Hardware Connections

Function	**ATmega2560 Outputs**
LED0	
LED1	
LED2	
LED3	
LED4	
LED5	
LED6	
LED7	
500ms LED	

Function	**ATmega2560 Inputs**
ADC0 (Wiper arm pot1)	
Encoder DT	
Encoder CLK	
Mode Pushbutton	

3. Read through this lab ALL procedures before attempting to write any code.

Lab Activities

4. End Goal (using interrupts), you may check off all procedures at the end or one at a time:
    a. Mode 1: the LED bar flashes every 500ms, can flash one bit or all of the LEDs. There is no specifications on the flashing values.
    b. Mode 2: the LED bar shows the ADC values 10 bit resolutions, using only the 8 MSB (need to use bit shifting for this).
    c. Mode 3: the LED bar shows encoder information. Turning the encoder counter clockwise, the LED bars lights up from right to left. Turning the knob clockwise, the LED bars lights up from left to right.
    d. Pressing the Mode pushbutton is continue to sequence the modes from 1 to 3.
5. Create a new project for this lab. Its name should be meaningful to illustrate the main objective of this lab. Work on one functionality at a time, then test and verify its performance before next functionality. Ensure proper wiring before moving on to next step.
6. Add all the modules from previous lab to this project. This way, you will not corrupt the code from last week while working on this lab.
7. Connect the IO and verify the connections using the debugger.
8. Modify the Timer0 delay function that used polling with Interrupt Service Routines (ISRs). Blink an LED at 500ms. Modify the program to blink the LED at rate of 500ms. Below some reminders to accomplish this:
    a. Add #include interrupt.h file to your main module and the module where you have the timer0 delay. #include statements should be located in the .h file

   b. Add sei(); in the main function, above the while(1) loop. This enable the global interrupt
   c. In the Timer0's initialization code, enable the timer interrupt. Check out Timer Interrupt Mask register. You should also start the timer at the initialization code.
   d. Add Interrupt Service Routine.
      i. Add a global variable called tick in your timer.h file. Make this volatile and with proper data type.
      ii. In the ISR, tick increases by 1
      iii. If you use Compare Output Flag, reset the timer counter. If you use the overflow flag, reset the timer counter to the predetermined value.
   e. Function of the interrupt service routine replaces the 1 msec delay function you built from previous lab. Delete the 1msec delay function.
   f. In your main function, check the status of tick. Toggle the LEDs whenever tick is bigger or equal to 500. Reset Tick to zero after the LED output has been changed. Test for proper operation before the next step.

Instructor Checkoff

a. Show completed IO table
b. Demonstrate and explain the code of Timer 0 interrupt to establish 500ms flash

9. Replace the ADC **10bit** polling function with an ISR and demonstrate its functionality. Display the **8 MSB** ADC value from to the LEDs. Hint:
   a. Start conversion at initialization
   b. Set the Mux channel to zero. We will not change channel.
   c. #include the interrupt.h file in adc
   d. ISR for ADC should locate in the adc module
   e. Don't forget the global variable for the adc_value.
   f. Once the adc_value is used to display in the LED, right shift by two to show the 8 MSB of the ADC values. Test for proper operation before the next step.

Instructor Checkoff

a. Demonstrate changing potentiometer, LED shows its corresponding ADC values (8MSB)
b. Explain the code on how the ADC interrupt works

10. Add another module for external interrupts. You may call this the Encoder module. External interrupts are to increment and decrement a counter or a flag. There are many ways to complete

this task, think about how you will reuse the code in the future when writing this module. You may use a counter that increment or decrement by one dependent on the direction of the turn. You may also use a flag that signal the main code to do something. If you use a flag, this flag should be reset in the main code. There should be two flags, one for clockwise turn, and another for counter-clockwise turn. The interrupts should be configured so that falling edge on the interrupt pin triggers the interrupts. See the external interrupt tables in the ATmega2560 data sheet.

11. In the main code, the encode is to use as a volume knob. When it turns CCW, one LED bar is turned on at a time, when it turns CW, LED bar is turned off one at a time. Review the video on Bb for detail operations.

Instructor Checkoff

a. Demonstrate the volume knob with the LEDs
b. Explain the code on how the external interrupt works

12. Combine all the functionalities described above. A push buttons, connected to an external interrupt input pin to enable which function the LED bar is to represent. Mode one, the LED bar shows a LED turns on and off every 500ms. Mode two, the LED bar shows the ADC value input at channel 0. Mode three, the LED bar shows the encoder volume information. While switching between modes, another LED should flashes every 500ms uninterrupted.

Instructor Checkoff

a. Demonstrate the mode operation with all the previous functions operational
b. Explain the code