Lab 06
Jonathan Gusler
3/14/2019

Overview
This lab's primary purpose was to introduce the use of timer interrupts and timer scaling. The process of using and setting up timer interrupts is similar to using hardware interrupts, as used in Lab 05 [1], except for using different registers.

Background
This lab requires knowledge of Atmel assembly language, a working knowledge of what happens in a program when an interrupt occurs, binary and hexadecimal numbers, how to use PORTs, and hardware interrupts.

Discussion
Functionally, the program implemented turns on and off LEDs in defined sequences. There are five sequences which are cycled through, each change being triggered by pressing the joystick inward. The LEDs are only on for 50ms at a time, this is controlled by a timer interrupt. The timer interrupt is setup such that the interrupt is generated every millisecond. A variable "Tick" is incremented every 1ms by the interrupt. A conditional checks for Tick to be equal to 50, when it is, the LED's in the light sequence are updated to their next state. After the LED's are updated, Tick is reset to 0 to begin counting up to 50 again. The light sequence execution is held in a while loop that runs indefinitely.

To set up the timer, three registers had to be set specifically for the needs of the program. TCNT0 was initialized to zero. TCCR0 was set to 0x06 to say the timer is operating in normal mode, the source clock is the system clock, and that the clock was being prescaled by a factor of 256. TIMSK was set to 0x01 to allow interrupts from timer 0. Prescaling is used to make the clock seem slower than it actually is. With a prescale of 256, the clock appears to the timer counter to be 256 times slower. Along with this, the timer was set up to only count to 29. The combination of the count limit and the prescale factor of 256 resulted in Tick being incremented every 1.002 ms. The math for this is as follows:

1) Divide the original clock frequency (7.3728 MHz) by the prescale factor 256
2) Divide that result by the count limit 29.
3) Inverse that result to get 0.001007 seconds.

The setup of the hardware interrupt was similar to what was done in Lab 05, the primary difference being which joystick action caused an interrupt. The code for the program is shown in Appendix A.

Analysis and Results
Tested on the microcontroller, the code ran as intended. The light sequences ran correctly, the sequences changed on the release of the joystick push in button, and the LEDs updated every time Tick was equal to 50.

Conclusion
This lab was successful in introducing the concept of and implementing the use of timer interrupts. The code involving the timer interrupt ran completely as intended.

References

[1] J. Gusler, "Lab 05," 2019.

Appendix A

```
/*
 * Lab06.c
 *
 * Created: 3/1/2019 2:09:44 PM
 * Author : Jonathan Gusler
 *
 * This program executes different LED turn on and turn off sequences.
 * The pattern changes when the joystick is released after being
pressed in.
 * An LED is only on for 50 Ticks, with one Tick being 1.007 ms, at
which point
 * it will turn off and the next LED in the sequence. There are 5
unique sequences.
 *
 *
 */

#define DELAY 50
#define F_CPU 7372800UL
#define TIMERCOUNT  255-29  // TCNT0 value to cause 1.007mS interrupts
#include <avr/io.h>         // Standard AVR header
#include <avr/interrupt.h> // Needed for interrupts
// For generating random numbers with rand()
#include <stdlib.h>



// Enumerate the possible states
enum states

{
  RightToLeft,          // Right to left, D0 to D7
  LeftToRight,          // Left to right, D7 to D0
  BackAndForth,         // Back and forth, D0 to D7 to D0
  Random,               // Random LED
  PairedInward          // Paired inward, D0 and D7 then D1 and D6...
};

// Create a variable to keep track of the program's states
enum states Mode;

// Create Interrupt to keep track of and change states
ISR(INT0_vect){
  switch(Mode)
  {
    case(RightToLeft):
      Mode = LeftToRight;
      break;
    case(LeftToRight):
```

```
        Mode = BackAndForth;
        break;
      case(BackAndForth):
        Mode = PairedInward;
        break;
      case(PairedInward):
        Mode = Random;
        break;
      case(Random):
        Mode = RightToLeft;
        break;
    }
}

// Initialize Tick counter
volatile unsigned char Tick = 0x00;

ISR(TIMER0_OVF_vect){
  TCNT0 = TIMERCOUNT; // Reset counter
// 1 overflow per 1ms with a 256 prescale and timer count of 255-29
  Tick++;

}

int main(void)
{
  // Set Joystick push in (PORTB4) as input
  DDRB  = 0b11101111;

  // Enable the pull-up resistors to prevent floating
  PORTB = 0b00010000;

  // Set PORTD0 as input to trigger ISR from joystick
  DDRD  = 0b11111110;

  // Set all of PORTE as output
  DDRE  = 0xFF;

  // Init timer count value
  TCNT0 = 0x00;

  // Set timer/counter 0 control register
  TCCR0 = 0b00000110; // Normal mode, internal clock, prescale by 256

  // Enable interrupt from timer 0
  TIMSK = 0x01;

  // Set the doors for the external interrupts
  EICRA = 0b00000011; // Triggered on release
  EIMSK = 0b00000001; // Only can come from PD0
```

```
//Allow Interrupts
sei();

// Set starting Mode
Mode = RightToLeft;

// Create variables for the different modes
unsigned char ledNum_0 = 0x01;
unsigned char ledNum_1 = 0x80;
// Used for BackAndForth
unsigned char cyclePhase_0 = 0x00;
unsigned char ledNum_2 = 0x01;
// Used for PairedInward

unsigned char ledLowerHalf = 0x01;
unsigned char ledUpperHalf = 0x80;

// Infinite loop that executes the different LED sequences
  while (1)
  {
   if(Tick == 0x32){ // Is Tick 50?
     switch(Mode)
     {
       case(RightToLeft):
         PORTE = ~(ledNum_0);
         if(ledNum_0 == 0x80){
           ledNum_0 = 0x01;
         }
         else{
           ledNum_0 = ledNum_0 << 1;
         }
         break;

       case(LeftToRight):
         PORTE = ~(ledNum_1);
         if(ledNum_1 == 0x01){
           ledNum_1 = 0x80;
         }
         else {
           ledNum_1 = ledNum_1 >> 1;
         }
         break;
       case(BackAndForth):
         PORTE = ~(ledNum_2);
         if(cyclePhase_0 == 0x00) // Going right to left
         {
           if(ledNum_2 == 0x40){
             cyclePhase_0 = 0x01; // Change phase to left to right
           }
           ledNum_2 = ledNum_2 << 1;
         }else{                   // Going left to right
           if(ledNum_2 == 0x02){
```

```
                cyclePhase_0 = 0x00; // Change phase to right to left
              }
              ledNum_2 = ledNum_2 >> 1;
            }
            break;
          case(PairedInward):
            PORTE = ~(ledUpperHalf | ledLowerHalf);
            if((ledUpperHalf | ledLowerHalf) == 0x18){
              ledUpperHalf = 0x80;
              ledLowerHalf = 0x01;
            } else {
            ledUpperHalf = ledUpperHalf >> 1;
            ledLowerHalf = ledLowerHalf << 1;
            }
            break;
          case(Random):
// Displays random 8-bit binary number in decimal range [1,255]
            PORTE = ~(rand() % 255 + 1);
            break;
       }
     Tick = 0x00; // Reset to wait another 50 Ticks
     }

     }
  return 0;
}
```