Skyler MacDougall, Matthew Gerace, Kaitlin Berryman

Homework 5: Due 2/19/2020

1. Create a reaction timer.

- After a random amount of time, the yellow LED turns on, the timer starts.
- If the user presses the button before the timer times out, the green LED goes on, and the yellow turns off.
- If the timer times out, the red LED goes on, and the yellow goes off.
- Use `delay();` and `random();` for the delay before the yellow light.
- check for the button push in the main program loop and use the timer interrupt to know that time is up
- start with setting the timer to 1/2 second and shorten it from there.
- be sure to disable the interrupt as soon as the user presses the button.

```
// Homework 5, Problem 1
// digital pins D5, D6, D7, D8

void setup()
{
  DDRD |= 0x20; // make PD5 an output -- green
  DDRD |= 0x40; // make PD6 an output -- yellow
  DDRD |= 0x80; // make PD7 an output -- red
  DDRB |= 0x01; // make PB0 an input  -- switch

  PORTD &= 0xDF; // turn PD5 output off
  PORTD &= 0xBF; // turn PD6 output off
  PORTD &= 0x7F; // turn PD7 output off

  // CTC mode timer setups
  TCCR1A = 0;
  TCCR1B = 0x0C;
  TCNT1  = 0;
  OCR1B  = 31249;

  cli(); // turn off global interrupts
  // set the intertupt flag
  TIMSK1 = 0x04;
  sei(); // turn on global interrupts
}

void loop()
{
  // wait a random amount of time
  delay(random());

  // turn the yellow LED on
  PORTD |= 0x40;
  //reset the timer
```

```
  TCNT1  = 0;

  while(PIND & 0x40)
  {
    if (!(PINB & 0x01))
    {
      PORTD &= ~(0x40);
      PORTD |= 0x10;
    }
  }

  //spaced resets a set amount of time
  delay(100);
}

ISR(TIMER1_COMPB_vect)
{
  // code for servicing the interrupt
  PORTD &= ~(0x40); // turn yellow LED off
  PORTD |= 0x80;    // turn red LED on
}
```

2. Rewrite lab #3 section 6 using registers and timer interrupts. Note that when interrupts are used for timers, you don't have to implement a state timer. Remember that when a variable is updated outside the main loop you must use the volatile design so that the compile does not optimize the variable out.

```
volatile bool greenFirstRun=0, loopBack=0, redFirstRun=0, grFirstRun=0;
#define MSEC_SAMPLE 1
#define SW1_PIN 5
//#define LED1_PIN 8
//#define LED2_PIN 11
#define  LED_CLOCK_PIN 11
#define  LED_DATA_PIN 12
#define QTR_SIG_PIN A3
#define QTR_5V_PIN A4
#define QTR_GND_PIN A5
#define MSEC_SAMPLE 1


enum {LED_OFF, BLINK_G_OFF, BLINK_R, BLINK_G_ON, BLINK_GR, BLINK_RATE};

boolean isSwPressed, prevIsSwPressed, isSwJustReleased, isSwJustPressed,
isSwChange;
volatile int state = LED_OFF;
int prevState = !state;
int stateTimer, adcQTR;
boolean isNewState;
boolean greentimer = true;

void setup() {
  DDRD &= ~(1 << 5); PORTD |= (1 << 5); //pinMode(SW1_PIN, INPUT_PULLUP);
  DDRB |= (1 << 3);//pinMode(LED_CLOCK_PIN, OUTPUT);
  PORTB &= ~(1 << 3);//digitalWrite(LED_CLOCK_PIN, LOW);
  DDRB |= (1 << 4); //pinMode(LED_DATA_PIN, OUTPUT);
  PORTB &= ~(1 << 4);//digitalWrite(LED_DATA_PIN, LOW);
  pinMode(QTR_SIG_PIN, INPUT);
```

```
  DDRC |= (1 << 4);//pinMode(QTR_5V_PIN, OUTPUT);
  PORTC |= (1 << 4);//digitalWrite(QTR_5V_PIN, HIGH);
  DDRC |= (1 << 5);//pinMode(QTR_GND_PIN, OUTPUT);
  PORTC &= ~(1 << 5);//digitalWrite(QTR_GND_PIN, LOW);

  OCR1A = 15624;
  OCR1B = 31249;
  ICR1 = 0xFFFF;
  Serial.begin(9600);
  Serial.println(F("Lab 2 Complex State Machine"));
} // setup()
//*************************************************************************
****
void display_color_on_RGB_led(unsigned long color) {
  unsigned long bitmask=0UL; // UL unsigned long literal (forces compiler
to use long data type)
  unsigned long masked_color_result=0UL;
  // digitalWrite(LED_CLOCK_PIN,LOW); //start with clock low.
  PORTB&=0b11110111;
  for(int i=23; i>=0; i--) { // clock out one data bit at a time, starting
with the MSB first
    bitmask= (1UL<<i);    // build bit mask. Note must use "1UL" unsigned
long literal, not "1"
    masked_color_result = color & bitmask; // reveals just one bit of
color at time
    boolean data_bit=!(masked_color_result==0); // this is the bit of data
to be clocked out.
 //   digitalWrite(LED_DATA_PIN,data_bit);
 if( data_bit ==1)
 PORTB|=0b00010000;
 else
 PORTB&=0b11101111;


   // digitalWrite(LED_CLOCK_PIN,HIGH);
   PORTB|=0b00001000;
  //  digitalWrite(LED_CLOCK_PIN,LOW);
  PORTB&=0b11110111;
  }
 // digitalWrite(LED_CLOCK_PIN,HIGH);
 PORTB|=0b00001000;
  delay(1); // after writing data to LED driver, must hold clock line
            // high for 1 ms to latch color data in led shift register.
}

void redOn(void) {
  PORTB = PORTB |(1 << 0);  // sets Uno dig_8, PORTB.0, pin to 1 (HIGH)
   display_color_on_RGB_led(0xFF0000); // physical pin 14 (28 pin DIP)
}
//*************************************************************************
****
void greenOn(void) {
  display_color_on_RGB_led(0x0000FF);

  if (greentimer == true){
  unsigned long start_time_microseconds,end_time_microseconds;
  start_time_microseconds=micros();

```

```
  end_time_microseconds=micros();
  Serial.print("Displaying the green color took ");
  Serial.print(end_time_microseconds-start_time_microseconds);
  Serial.println(" microseconds ");
  }
}
//************************************************************************
****
void allOff(void) {
   display_color_on_RGB_led(0x000000);
}
ISR(TIMER1_COMPA_vect){
  if(greenFirstRun){
    allOff();
    greenFirstRun = !greenFirstRun;
  }
  if(redFirstRun){
    allOff();
    redFirstRun = !redFirstRun;
  }
  if(grFirstRun){
    redOn();
    grFirstRun = !grFirstRun;
  }
}

ISR(TIMER1_COMPB_vect){
  loopBack=true;
}

void loop() {
  prevIsSwPressed = isSwPressed;
  isSwPressed = !(PINC & 0x20);
  isSwJustPressed = (isSwPressed && !prevIsSwPressed);
  isSwJustReleased = (!isSwPressed && prevIsSwPressed);
  isSwChange = (isSwJustReleased || isSwJustPressed);

  isNewState = (state != prevState);
  prevState = state;

  switch (state) {

    case LED_OFF:
      if (isNewState) Serial.println("LED_OFF");
      allOff();
      if (isSwJustReleased)
      state = BLINK_G_ON;
    break;

        case BLINK_G_ON:
      if (isNewState) {
      Serial.println("BLINK_G_ON");
      greenFirstRun = true;
      grFirstRun = false;
      OCR1A = 15624;
      OCR1B = 62499;
      TIMSK1 = 0x06;//Use OCR1A and OCR1B
      sei();
```

```
      greenOn();
      }
     if (loopBack){
       greenOn();
       TCNT1 = 0;
       loopBack = 0;
       greenFirstRun = true;
     }
      greentimer = false;
      if (isSwJustReleased) {
       cli();
       allOff();
       state = BLINK_R;
      }
    break;

case BLINK_R:
      if (isNewState) {
      TCNT1 = 0;
      greenFirstRun = false;
      redFirstRun = true;
      sei();
      redOn();
      Serial.println("BLINK_R");
      }
      if (loopBack){
       redOn();
       TCNT1 = 0;
       loopBack = false;
       redFirstRun = true;
     }
      if (isSwJustReleased) {
        allOff();
        cli();
        state = BLINK_GR;
      }
    break;


    case BLINK_GR:
      if (isNewState) {
        stateTimer = 0;
        TCNT1 = 0;
        OCR1A = 15624*2;
        grFirstRun = true;
        redFirstRun = false;
        sei();
        Serial.println("BLINK_GR");
      }
      if (loopBack){
        greenOn();
        TCNT1 = 0;
        loopBack = false;
        grFirstRun = true;
      }

      if (isSwJustReleased) {
        allOff();
```

```
196           state = LED_OFF;
197         }
198       break;
199
200
201       default: state = LED_OFF;
202     } // switch (state)
203
204 } // loop()
```

3. Servos should react to a PWM signal as shown in the homework diagram. however, a pulse width of 1ms does not always put the servo at 0° and a pulse width of 2ms does not always put the servo at 180°. Using timer1 and a FAST PWM mode create a PWM signal to drive a servo motor. You may need to adjust the registers for pulse width to get a full 180° of rotation. Note, you cannot just copy the code from lab 4 as that was not fast PWM mode.

```
1  unsigned long servoValue=0;
2 //*********************************************************************
  *********
3 void setup()
4 {
5  pinMode(9,OUTPUT);
6  Serial.begin(9600);
7  TCCR1A=0xB2;
8  TCCR1B=0x1B;
9  ICR1=0x1387;
10
11 }
12 //*********************************************************************
  *********
13 void loop()
14 { delay(100);
15  servoValue++;
16  OCR1A=144+servoValue;
17  if (servoValue>=470) servoValue=0;
18 }
```

4. Read sections “Reset and Interrupt Handling” in the datasheet and answer the following questions:

    1. How does microcontroller handle multiple interrupts arriving from different peripherals while an ISR is being serviced?

       All interrupts are disabled while an ISR is being serviced.

    2. What does microcontroller do in the 4 cycle response time to service an ISR?

To enter the ISR, the program counter is pushed onto the Stack during the first cycle. The next three cycles are to jump to the correct spot in memory where the interrupt routine is stored. When exiting the ISR, the program counter is popped back from the stack (two cycles), incremented twice (one cycle), and interrupts are re-enabled.