POE Lab 1: Bike Light

Raymundo Camacho and Benjamin Ziemann

September 11, 2017

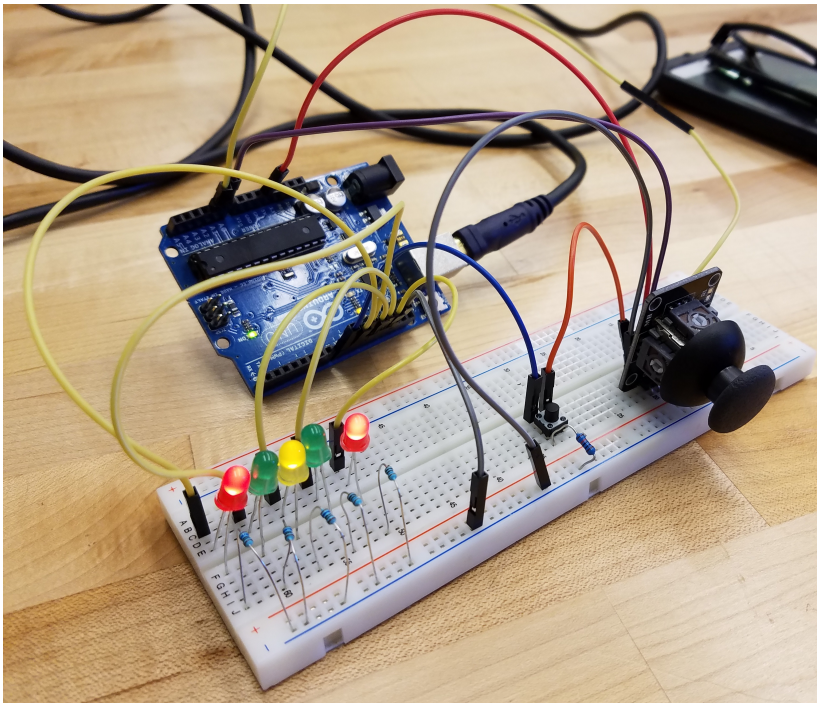

Figure 1: The final circuit

1 Part 1

1.1 Description

For this lab we created a multi-mode, "bike light" style line of LEDs, controlled by a single button. Using an Arduino Uno as both a microprocessor and a GPIO controller, we made it so the user could cycle the lights in 5 different patterns:

- All off
- All on
- All flashing
- Every other going on and off
- Lighting up one by one in a line

1.2 The Circuit

Using the 5V GPIO pins on the Arduino, we could send voltage to the LEDs. However the LEDs we were use are rated for between 1.9 and 2.3 volts and 30mA max so we need to drop the voltages using resistors. Where V is the voltage difference (5V-2V), I is the current (20 mA to avoid potentially providing over-current), and R is the resistance we are looking for, we used Ohm's Law

$$V = IR$$

$$3 = (2.0 \times 10^{-2})(R)$$

$$R = 150\Omega$$

to come up with the necessary resistor values of 150 ohms. Because of our resources though we ended up going with 160 Ohm resistors which, because of the tolerances we were playing with (1.9V to 2.3V and 10mA to 30mA) we recalculated and made sure they would still work.

For the button circuit, we attached the normally open terminal to pin 8 on the Arduino to be used as an input and a 10k ohm resistor to ground as seen in figure 2. When not pressed, this is the only path to provide power to the pin, ensuring the pin will be tied low. The other terminal is connected to our 5V supply from the Arduino. This means that when we press the button the voltage will take the path of least resistance, going to the pin instead of ground because of the 10K Ohm resistor impeding it. This will give the input pin a state of high when read by the Arduino, allowing us to detect button presses.

1.3 Figure 2: Schematic Part 1

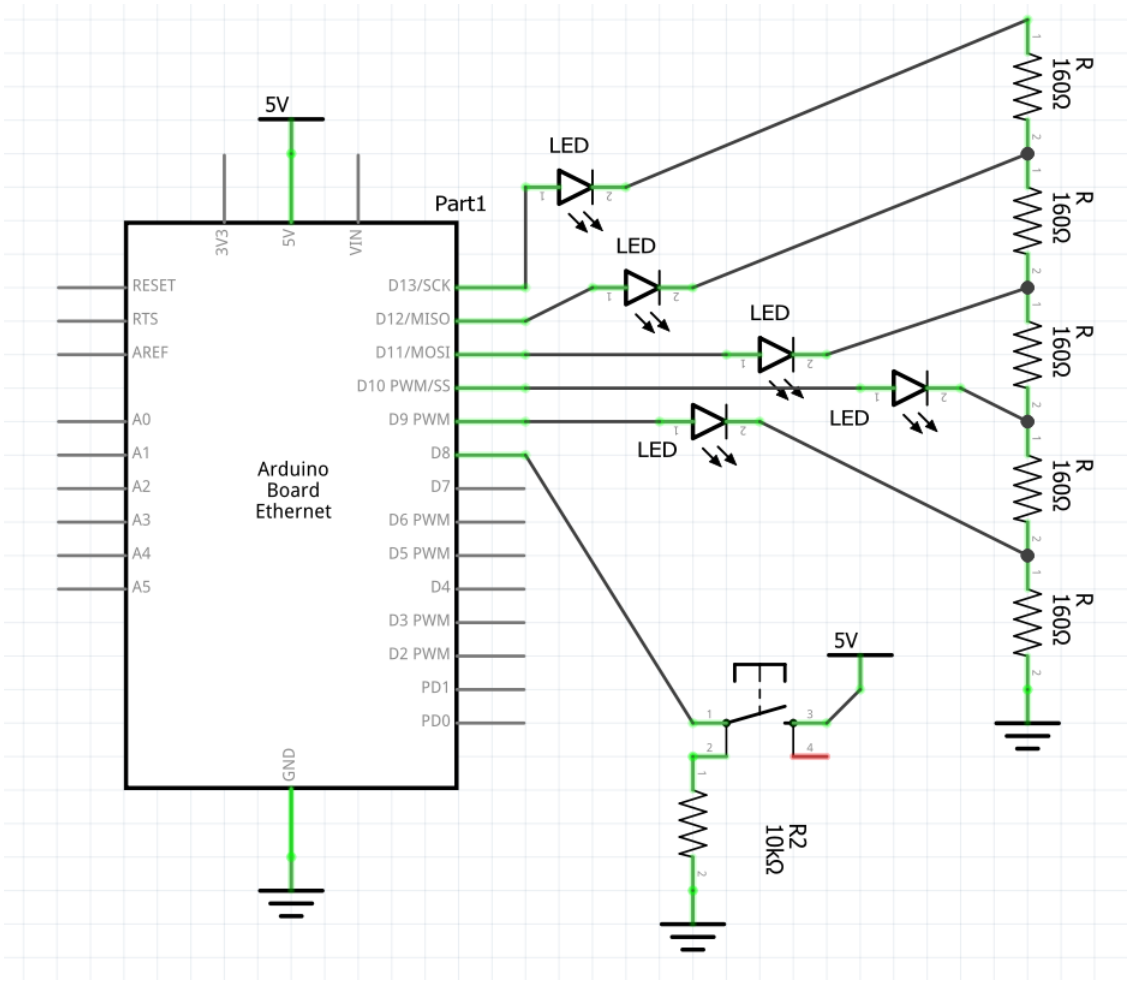

Figure 2: Part 1 schematic

1.4 The Code

Our full code can be seen in the appendix, figure 4. It is broken down into a few key components.

First are initializing global variables. Although our looping function has the most lines of code, it is our three functions, setPins, checkButton and newDelay that run the most within the loop. Because of this we needed these functions to be able to access the same variables to change as our code is much like a state machine. These variables include pin numbers, the current mode of the lights, and flag variables.

```
void setPins(int p1, int p2, int p3, int p4, int p5) {
  digitalWrite(LED_RED_1, p1);
  digitalWrite(LED_GREEN_1, p2);
  digitalWrite(LED_YELLOW, p3);
  digitalWrite(LED_GREEN_2, p4);
  digitalWrite(LED_RED_2, p5);
}
```

Code Sample 1: setPins function for easy updating

Next are our three more customized functions. Code sample 1 shows our first, the setPins function. setPins acts as a simple and understandable way for us to individually set the states of all 5 LEDs at once. This is incredbily helpful because some of our patterns require a lot of switching which lights are on and off.

```
bool checkButton(){
    //Button press check
    if (digitalRead(BUTTON) == 1) {
      //Debounce time elapsed check
      if ((millis() - lastTime) > delayTime) {
        Serial.println("triggered");
        mode++;
        //Loop over modes
        if (mode > 4) {
          mode = 0;
        }
        lastTime = millis();
        return true;
    }
    else{
      Serial.println("refused");
    }
  }
  return false;
}
```

Code Sample 2: checkButton function for debounce and button recognition

Code sample 2, the checkButton function, handles the checking of the button and any additional checks we want to put around that. On a button press the mode increments, which in the main looping function will trigger a different if statement for different lights. There is also an if statement to check whether the mode number needs to reset to 0. This allows for easy implementation of additional modes. While it started off as just reading the pin state, our realization of the need for debounce (See below in "Obstacles" section) necessitated that we add in timer like system to prevent a single press from a human to be interpreted as multiple mode change.

```
bool newDelay(int wait){
  unsigned long startTime = millis();
  while((millis()-startTime) < wait){
    if(checkButton()){
      return true;
    }
    //Delay to stop Arduino from running full spee
    delay(5);
  }
  return false;
}
```

Code Sample 3: newDealy function to avoid delays

Finally, code sample 3, newDelay, was our solution to the problem of not being able to check the button's state when between switching lights (See below in the "Obstacles" section for more information). By comparing the time the function was called to the current time and a while loop, we created a timer that allows us to run any extraneous commands, in this case button checks, during that period while not progressing the main loop.

The final section is the primary Arduino loop function. In it we check the current mode of the system and combinations of setPins and newDelays to create the patterns we want.

1.5 Obstacles

```
int lastTime = 0;
int delayTime = 1000;

bool checkButton(){
    //Button press check
    if (digitalRead(BUTTON) == 1) {
      //Debounce time elapsed check
      if ((millis() - lastTime) > delayTime) {
```

Code Sample 4: Debounce portion of checkButton

One of the largest obstacles we ran into while writing the code for this project was the need for a debounce on the button. Without it, what a human perceives to be single press of the button is interpreted by the Arduino as multiple inputs, causing it to rapidly loop through the 5 modes and not cleanly move to the next as we had intended. Additionally, any noise in the system would could also increment the mode unintentionally. To solve this we implemented a simple timer-like if statement to compare the state and time of the button when it had last been pressed and its state then to the current system. If the state had changed and the time between the accepted change was long enough, the system changed the mode.

The other major obstacle we encountered was the need to interrupt any mode at any given time. Because of our use of the "delay" function, which stops the system for the specified time, in our first pass, we were unable to check the button state during the cycling to lights. This was most noticeable in the line lighting mode when the button press would only activate when the first light was about to be lit. However by creating our own delay function, seen in code sample 3, which used a similar timing system to the one described in the debounce timer above, we could implement additional checks during that "delayed" period. By returning booleans for our checkButton and newDelay function, we set up break statements, allowing for the switching of modes at any point.

2 Part 2

2.1 Description

To expand on our system, we used a two axis joystick controller to allow for the user to change the speed at which the LEDs flash based on the value of the Y-axis of the joystick.

2.2 The Circuit

Continuing with the circuit constructed in part one, we connected a joystick to the button circuit as shown in figure 3. The 5V pin on the Arduino was connected to the joystick. However, the button circuit was now not being powered, so the button circuit was connected to the joystick through the 5V pin on the joystick. The Arduino's analog A0 and A1 pins were connected to the joystick's VRX (potentiometer affected by the joystick's movement in the x direction) and VRY (potentiometer affected by the joystick's movement in the y direction) pins respectively. Finally, the joystick was grounded. When the joystick is not moved, the analog inputs (pins A0 and A1) receive normal reading of around 500. When the joystick is moved up and down, the analog readings are increased to 1023 and decreased to 0 respectively.

2.3 Figure 3: Schematic Part 2

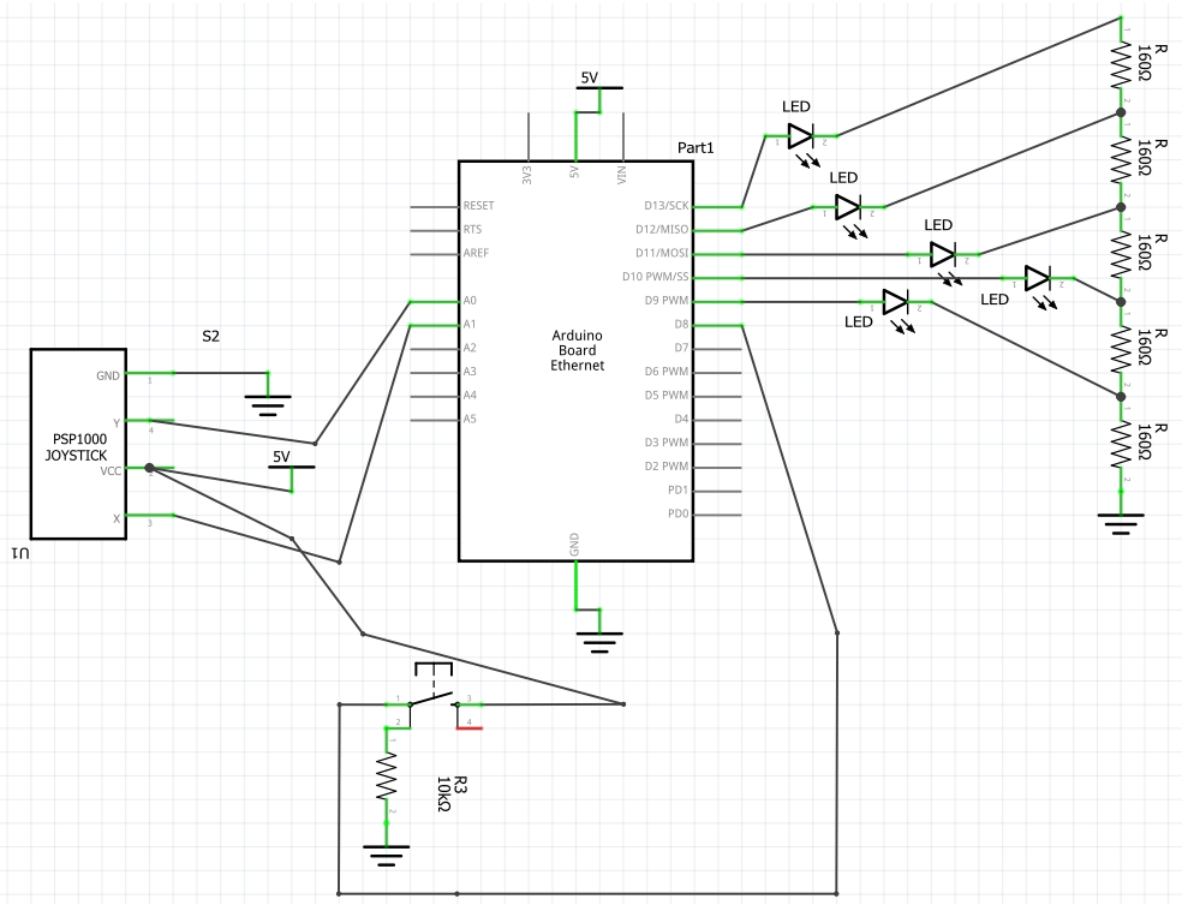

Figure 3: Part 2 schematic

2.4 The Code

```
int makeDelay(){
  long axis = analogRead(YAXIS);
  //Original map function with a function we generated
```

```
  //Later realized there was a function to do this already in Arduino
  //joyDelayTime = (1/1000.0)*(axis*axis*1.0)+(-2.5*axis)+2000;
  joyDelayTime = map(axis, 0, 1023, 1000,250);
  return joyDelayTime;
}
```

Code Sample 5: makeDelay function to set a delay based on joystick value

The first step in writing the code was creating a function to map our range of analog readings from the joystick to the delay we wanted. We generated a simple quadratic function with the analog value as the variable to do this. Within the code we made this its own function, makeDelay as seen in code sample 5, since it's fairly tedious to type out and we knew we would need it a lot. However we realized that it would be impractical to regenerate a new function whenever we wanted to change these time delays so we switched to using the built in Arduino map function, which allows us to quickly change time delays.

```
bool newDelay(){
  int wait = makeDelay();
  unsigned long startTime = millis();
  while((millis()-startTime) < wait){
    if(checkButton()){
      return true;
    }
    //Delay to stop Arduino from running full spee
    delay(5);
  }
  return false;
}
```

Code Sample 6: Overlaoded newDelay function to allow for changing speeds any time

Next we overloaded the newDelay function we wrote in part 1. Since we want the delay to be based on the joystick value and not a predefined time, we took out the parameter and instead compute the delay time directly in the overloaded newDelay function. This can be seen by comparing code samples 3 and 6. By putting it here, we make the check for a change in speed occur far more often, allowing for a more responsive system.

2.5 Obstacles

Like the interrupt in part one, we had a problem of updating the delay time at any time during the system. Initially we had it so the speed change would only take effect at the beginning of a loop through one of the modes. While fine for something like the flashing mode, which had a loop cycle that consisted of two LED changes, it was very noticeable in the longer ones, most notably the line flash. To solve this, we implemented an overloaded newDelay function with no parameters. Instead, it gets its wait time from the joystick Y-value plugged into the function we generated. Now everywhere there is a delay between the LEDs changing, this version of the function may be implemented, allowing the time delay between the switch to be updated between each light switch.

3 Reflection

We thought the lab turned out well, with the circuit being complex enough but doable and learning to debug the code.

Ray: My experience in ISIM allowed me to get the necessary values for the resistors and having Ben get the same values cemented my knowledge. Having Ben capable in coding allowed me to understand the language of Arduino C as he went over the key features of. He was a great help in helping with the circuit and learning how to read Arduino C.

Ben: Having had experience with Arduino projects before, I thought it was a great refresher on many of the basics. I also found out I had forgotten a lot of ISIM, so having Ray as a partner helped me a lot to understand the math for the electrical components and the mapping function.

4 Appendix

4.1 Figure 4: Code

```
1
2  //LED pin numbers
3  int LED_RED_1 = 13;
4  int LED_GREEN_1 = 12;
5  int LED_YELLOW = 11;
6  int LED_GREEN_2 = 10;
7  int LED_RED_2 = 9;
8
9  //Button pin numbers
10 int BUTTON = 8;
11
12 //Axis analog pin numbers
13 int YAXIS = 0;
14
15 //Bike lamp modes
16 // 0 = all off
17 // 1 = all on
18 // 2 = all flashing
19 // 3 = two on - three on
20 // 4 = line flash
21 int mode = 0;
22
23 //Debounce
24 unsigned long lastTime = 0;
25 unsigned long delayTime = 500;
26
27 //newDelayTime
28 int joyDelayTime = 500;
29
30 //Flag variable to restart loops
31 int flag = 0;
32
33 void setup() {
34   Serial.begin(9600);
35
36   //LED pin modes
37   pinMode(LED_RED_1, OUTPUT);
```

```
38    pinMode(LED_GREEN_1, OUTPUT);
39    pinMode(LED_YELLOW, OUTPUT);
40    pinMode(LED_GREEN_2, OUTPUT);
41    pinMode(LED_RED_2, OUTPUT);
42
43    //Button pin mode
44    pinMode(BUTTON, INPUT);
45
46  }
47
48  //Set pins to corresponding output
49  void setPins(int p1, int p2, int p3, int p4, int p5) {
50    digitalWrite(LED_RED_1, p1);
51    digitalWrite(LED_GREEN_1, p2);
52    digitalWrite(LED_YELLOW, p3);
53    digitalWrite(LED_GREEN_2, p4);
54    digitalWrite(LED_RED_2, p5);
55  }
56
57  //Custom delay to allow for interrupt with joy input
58  bool newDelay(){
59    int wait = makeDelay();
60    unsigned long startTime = millis();
61    while((millis()-startTime) < wait){
62      if(checkButton()){
63        return true;
64      }
65      //Delay to stop Arduino from running full spee
66      delay(5);
67    }
68    return false;
69  }
70
71  //Overloaded function for user defined wait
72  bool newDelay(int wait){
73    unsigned long startTime = millis();
74    while((millis()-startTime) < wait){
75      if(checkButton()){
76        return true;
77      }
78      //Delay to stop Arduino from running full spee
79      delay(5);
80    }
81    return false;
82  }
83
84  bool checkButton(){
85      //Button press check
86      if (digitalRead(BUTTON) == 1) {
87        //Debounce time elapsed check
88        if ((millis() - lastTime) > delayTime) {
89          Serial.println("triggered");
90          mode++;
91          //Loop over modes
92          if (mode > 4) {
93            mode = 0;
94          }
95          lastTime = millis();
96          return true;
```

```
97      }
98      else{
99        Serial.println("refused");
100     }
101   }
102   return false;
103 }
104
105 //Set up delay based on joystick
106 int makeDelay(){
107   long axis = analogRead(YAXIS);
108   //Original map function with a function we generated
109   //Later realized there was a function to do this already in Arduino
110   //joyDelayTime = (1/1000.0)*(axis*axis*1.0)+(-2.5*axis)+2000;
111   joyDelayTime = map(axis, 0, 1023, 1000,250);
112   return joyDelayTime;
113 }
114
115 void loop() {
116   checkButton();
117
118   //All off
119   if (mode == 0) {
120     setPins(0, 0, 0, 0, 0);
121   }
122
123   //All on
124   else if (mode == 1) {
125     setPins(1, 1, 1, 1, 1);
126   }
127
128   //Flashing
129   else if (mode == 2) {
130     flag = 0;
131     while(flag == 0){
132       setPins(1, 1, 1, 1, 1);
133       if(newDelay()){
134         break;
135       }
136       setPins(0, 0, 0, 0, 0);
137       if(newDelay()){
138         break;
139       }
140       flag == 1;
141     }
142   }
143
144   //Two on, three on
145   else if (mode == 3) {
146     flag = 0;
147     while(flag == 0){
148       setPins(0, 1, 0, 1, 0);
149       if(newDelay()){
150         break;
151       }
152       setPins(1, 0, 1, 0, 1);
153       if(newDelay()){
154         break;
155       }
```

```
156         flag == 1;
157       }
158     }
159
160     //Line flash
161     else if (mode == 4) {
162       flag = 0;
163       while(flag == 0){
164         setPins(1, 0, 0, 0, 0);
165         if(newDelay()){
166           break;
167         }
168         setPins(0, 1, 0, 0, 0);
169         if(newDelay()){
170           break;
171         }
172         setPins(0, 0, 1, 0, 0);
173         if(newDelay()){
174           break;
175         }
176         setPins(0, 0, 0, 1, 0);
177         if(newDelay()){
178           break;
179         }
180         setPins(0, 0, 0, 0, 1);
181         if(newDelay()){
182           break;
183         }
184         setPins(0, 0, 0, 1, 0);
185         if(newDelay()){
186           break;
187         }
188         setPins(0, 0, 1, 0, 0);
189         if(newDelay()){
190           break;
191         }
192         setPins(0, 1, 0, 0, 0);
193         if(newDelay()){
194           break;
195         }
196         setPins(1, 0, 0, 0, 0);
197         if(newDelay()){
198           break;
199         }
200         flag == 1;
201       }
202     }
203
204     //Delay to stop Arduino from running full speed
205     delay(5);
206
207 }
```