Bike Light Lab

Jaime Grau and Nick Francisci

January 2015

1 Introduction

In this lab we work with an Arduino, 3 LED lights, some resistors, a button and a potentiometer. The goal was to make the LEDs turn on and off in several distinct patterns, as a bike-light would. The patterns were changed by pressing a button. The speed of the patterns is controlled by the potentiometer. We began the lab by setting up and testing the button, LEDs, and potentiometer first. We then connected the circuit elements and programmed the system.

2 Process

2.1 Circuit Design

For the circuit we began by wiring the LEDs, then the button, and finally the potentiometer. We use an Arduino Uno as the controller for the circuit. We use an analog port of the Arduino to connect to the potentiometer, digital ports 10-12 for outputting to the LEDs, and digital port 2 to connect to the button. We wired the LEDs in series with 1k ohm resistors in order to maintain a safe amperage level. We wired the button in series with a 10k ohm resistor to protect the Arduino digital input port. The circuit diagram is shown in figure 3 and the circuit itself is shown in figure 4.

2.2 Programming

For organization, we have split the code into two parts, registering the button push and running the LED patterns. To register the button press, we began by writing code that polled the button periodically to check it's state. Polling is a flawed method of detecting the button state because the button will not be polled if other code, such as a pattern, is being run. Therefore, we used an interrupt to register each button press. In the interrupt (shown in fig. 1), we advance a counter for the pattern by one for each press. Initially, we had issues with the button "bouncing": registering more than one edge detect per press. We resolved this in the interrupt routine by only registering a press if it occurred more than 200 ms from the last press.

```
void lightISR() {
  static volatile unsigned long lastInterruptTime = 0;
  unsigned long interruptTime = millis();

  // Wait for 200 ms before allowing another interrupt
  if (interruptTime - lastInterruptTime > 200) {
    patternNumber++;
  }

  lastInterruptTime = interruptTime;
}
```

Figure 1: The interrupt routine that is run when the button is pressed.

We placed each of the bike-light patterns in their own, numbered function. The central loop of the program checks a pattern variable which stores the number of the pattern to run. It then initiates that pattern. The pattern variable is incremented by the button interrupt function. The code for our primary loop is shown in figure 2. There is a significant drawback to this design: the pattern only changes after the previous pattern has completed. We had difficulty with the semantics of making the pattern change instantly on the button press. We considered having a new, infinite loop started on each interrupt, but were concerned that as the stack frames for each loop build up in memory, they may eventually cause a stack overflow.

The full source code for the lab is included in section 5.2.

3 Reflection

This lab was primarily useful in teaching us about Arduino programming. We had not used interrupts previously, nor debounced a button. The lab also required creativity to solve the issue of cycling through different LED patterns. Lastly, we found it useful to reference the LED spec-sheet and calculate a safe resistance value. Previous classes have emphasized circuit construction, but not circuit design.

4 Conclusion

The design functioned. Since the entire design existed on a breadboard and Arduino with jumpers for wiring, it was mechanically very fragile. However, the electrical wiring and programming were sound. Except for the aforementioned issue of patterns not changing instantly, the bike-light's button and patterns worked.

```
void loop(){
  switch (patternNumber%7) {
    case 1:
      pattern1();
      break;
    case 2:
      pattern2();
      break;

...

    default:
      clearAll();
      break;
  }
}
```

Figure 2: The primary loop of the code, which handles switching between patterns.

5 Appendix

5.1 Circuit Diagram

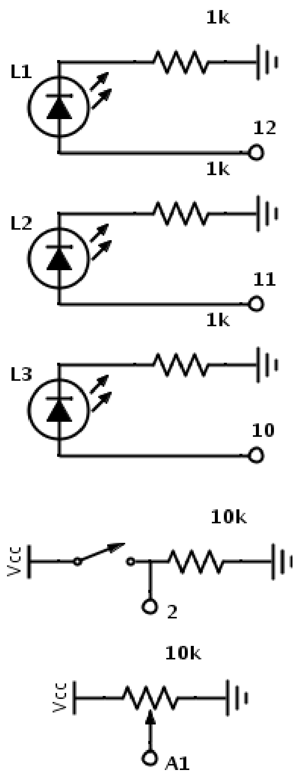

Figure 3: The full circuit diagram for the bike light. The numbers indicate the Arduino serial pin numbers or analog pin numbers.

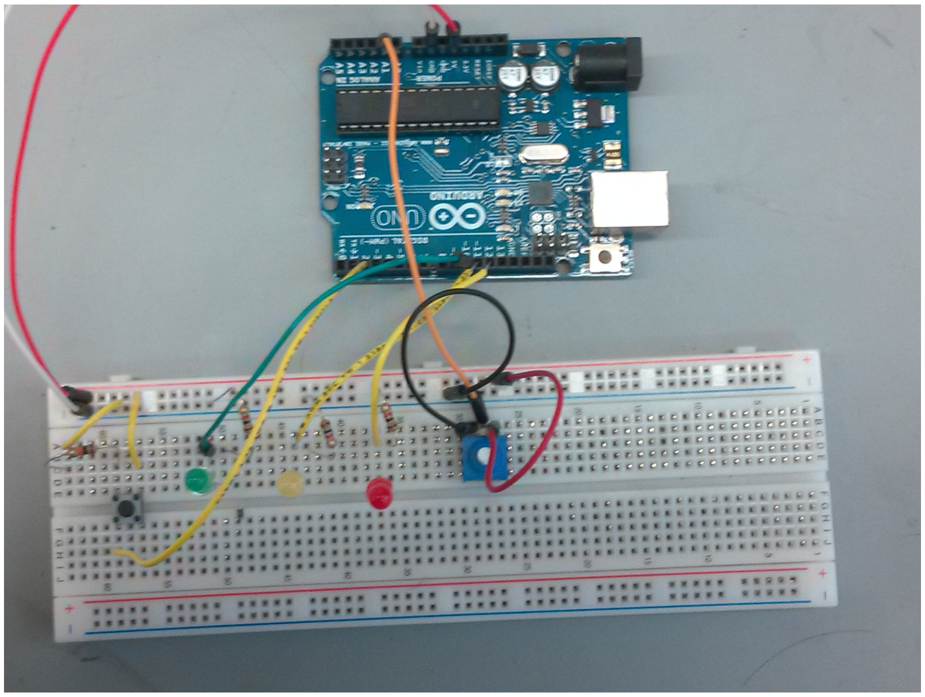

Figure 4: A picture of the completed circuit.

5.2 Full Source Code

```
const int analogPin = 1;
const int buttonPin = 2;
const int ledPin_1 =  12;
const int ledPin_2 =  11;
const int ledPin_3 =  10;
unsigned int patternNumber = 0;

void setup() {
  pinMode(ledPin_1, OUTPUT);
  pinMode(ledPin_2, OUTPUT);
  pinMode(ledPin_3, OUTPUT);
  pinMode(buttonPin, INPUT);
  digitalWrite(buttonPin, HIGH);

  attachInterrupt(0, lightISR, FALLING);
}

void loop(){
  switch (patternNumber%7) {
    case 1:
      pattern1();
      break;
    case 2:
      pattern2();
      break;
    case 3:
      pattern3();
      break;
    case 4:
      pattern4();
      break;
    case 5:
      pattern5();
      break;
    case 6:
      pattern6();
      break;
    default:
      clearAll();
      break;
  }
}


// ------ PATTERNS ------ //

// Pattern 0 is implicitly all off

// All flashing
void pattern1() {
  pattern2(); // All on
  delay(val());
  clearAll(); // All off
  delay(val());
}

// All on
```

```
void pattern2(){
  allOn();
}

// Bouncing
void pattern3() {
  digitalWrite(ledPin_1, HIGH);
  digitalWrite(ledPin_2, LOW);
  delay(val()/2);
  digitalWrite(ledPin_2, HIGH);
  digitalWrite(ledPin_1, LOW);
  delay(val()/2);
  digitalWrite(ledPin_3, HIGH);
  digitalWrite(ledPin_2, LOW);
  delay(val()/2);
  digitalWrite(ledPin_2, HIGH);
  digitalWrite(ledPin_3, LOW);
  delay(val()/2);
}

// Turn on in sequence
void pattern4() {
  digitalWrite(ledPin_1, HIGH);
  delay (val());
  digitalWrite(ledPin_2, HIGH);
  delay (val());
  digitalWrite(ledPin_3, HIGH);
  delay (val());
  clearAll();
  delay(val());
}

// Turn on and off in travelling pattern
void pattern5 () {
  digitalWrite(ledPin_3, HIGH);
  delay(val()/2);
  digitalWrite(ledPin_2, HIGH);
  delay(val()/2);
  digitalWrite(ledPin_3, LOW);
  digitalWrite(ledPin_1, HIGH);
  delay(val()/2);
  digitalWrite(ledPin_2, LOW);
  delay(val()/2);
  digitalWrite(ledPin_1, LOW);
}
```

```
// Alternate flash
void pattern6() {
  digitalWrite(ledPin_1, HIGH);
  digitalWrite(ledPin_3, HIGH);
  digitalWrite(ledPin_2, LOW);
  delay(val()/2);
  digitalWrite(ledPin_2, HIGH);
  digitalWrite(ledPin_1, LOW);
  digitalWrite(ledPin_3, LOW);
  delay(val()/2);
}

void clearAll() {
  digitalWrite(ledPin_1, LOW);
  digitalWrite(ledPin_2, LOW);
  digitalWrite(ledPin_3, LOW);
}

void allOn() {
  digitalWrite(ledPin_1, HIGH);
  digitalWrite(ledPin_2, HIGH);
  digitalWrite(ledPin_3, HIGH);
}


// ------ UTILITY FUNCTIONS -----//

int val(){
  return analogRead(analogPin);
}

/*
 * Interrupt for button push
 * Deals with debouncing switch. Advances
 * the pattern once for each button push
*/
void lightISR() {
  static volatile unsigned long lastInterruptTime = 0;
  unsigned long interruptTime = millis();

  // Wait for 200 ms before allowing another interrupt
  if (interruptTime - lastInterruptTime > 200) {
    patternNumber++;
  }

  lastInterruptTime = interruptTime;
}
```