PoE Lab 1: Bike Light

Braden Oh, Aiden Carly-Clopton

1 Introduction

For this lab we used an Arduino to make a blinking LED bike light with five modes and a push button to cycle between them. We debounced the push button and set the blink patterns with software (section 3). In addition, we added a potentiometer as an analogue input to control the blink rate of each of the modes.

2 Circuit Design

2.1 Current Limiting Resistors

Our first step in designing this circuit was to chose the value of the current limiting resistors for the LEDs. We found in the LED datasheet that the voltage drop across the LED is 1.9 V and the maximum allowable current is 30 mA. The voltage supplied by the arduino is 5 V, so the voltage drop across the current limiting resistor would be 3.1 V. We decided on a maximum current of 0.02 A, and using

$$\mathbf{V = I \times R}$$

we set up the following equation to calculate the resistor value:

$$\mathbf{3.1\ V = 0.02\ mA \times R}$$

$$\mathbf{R = 155\ \Omega}$$

The closest available resistor value was 160 Ω, and since 160 Ω is a greater resistance that 155 Ω it would result in a slightly lower current which is fine. For that reason, we chose to use a 160 Ω resistor.

2.2 Push Button

The next component in the basic bike light was the push button and pull down resistor, which is the 10k Ω resistor in figure 1. The purpose of the pull down resistor is to make it such that when the button is not pressed the value of the digital pin reads close to zero, since the resistor connects it to ground. When the push button is pressed, the pin is connected to the 5V supply and reads high. In reality, the spring in the button oscillates before coming to rest, connecting and disconnecting the two terminals several times. This can be handled with an RC low pass filter to smooth out the noise, however we chose to handle this issue (a process called *debouncing*) with software. This is explained in more detail in section 3.3, and section 3.4 shows the code for doing so.

2.3 Potentiometer

The final hardware component in this circuit is a potentiometer, which is used to continuously vary the rate at which the lights flash. While the push button is used as a digital input, with the pin either pulled low by the pull-down resistor or high by the closed switch, the potentiometer is wired as a voltage divider and connected to analogue pin 0. The voltage at the middle pin varies between 0V and 5V depending on the position of the potentiometer. Given the voltage resolution of the

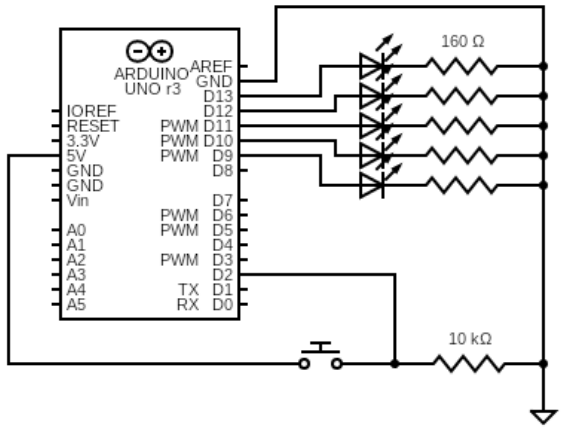

Figure 1: Bike light circuit with push button input

arduino analogue input, there are 1024 possible values this range could read as, meaning it can control the flicker speed in an effectively continuous manner.

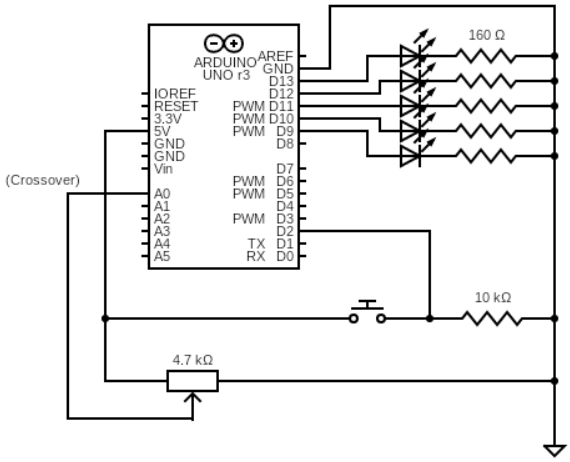

Figure 2: Bike light circuit with push button input and potentiometer

3 Software Design

Our software was divided into three functions: *setup()*, *loop()*, and *changeMode()*. The former two are special functions required by the Arduino: *setup()* handles code that needs to be run only once - upon initialization - while *loop()* contains the body of code that the Arduino runs on repeat during its duration of operation. The latter function, *changeMode()*, runs as a response to the button-pressed event and contains code to initiate a mode change.

3.1 *setup()*

Upon initialization, the *setup()* function is run once. This function defines five LED output pins (digital pins 9-13), and one interrupt pin (digital pin 2). It also defines the behavior of the interrupt: upon receiving a HIGH signal to pin 2 (the 'interrupt'), the software would break and execute function *changeMode()*. We chose to use Arduino's built-in interrupt capability because we wanted the ability to trigger *changeMode()* at any point in the loop, regardless of the position in the loop or whether it was during a time delay (more on delays below).

3.2 *loop()*

The main body of program is looped indefinitely in the *loop()* function. This function contains an if/else-if/else block that controls the system "mode." A global variable (***int*** *mode*) stores an integer value from 0 to 5, with each integer value corresponding to a case in the if-else block (for a total of six cases). Each case contains a set of commands to the output pins which power the LEDs on or off, generating a given pattern. These cases are, in order: all on; all flashing; rolling down; alternate flashing; bouncing; and all off. For dynamic patterns (e.g. flashing) the software sets a HIGH voltage state for LEDs that need to be on (e.g. all pins set HIGH) and then executes a *delay()* command. *delay(n)* is a built-in Arduino function that pauses the system for a delay period of n ms. During this delay the system stops at the delay's position in the code and waits for the delay to run out. At the conclusion of the delay, the next pattern is set and another delay is executed. We acknowledge that *delay()* prevents "multitasking" in Arduino, but it is a simple solution that works well with *interrupt*: even if the system is in the middle of executing a delay, the Arduino system interrupt will override the delay, break free of the main loop, and execute a mode change.

3.3 *changeMode()*

The *changeMode()* function, triggered by a system interrupt, performs two actions: debouncing the interrupt signal and advancing the *mode* variable. The button turns out to be "bouncy" in that when the button is pressed (to close 5V to the interrupt pin) the signal received is not a clean railing of 0V to 5V, but oscillates rapidly ("bounces") between 0V and 5V a number of times before a clean 5V offset is received at the input pin. Since we configured the interrupt to trigger upon reading a HIGH voltage, every time the signal bounces up to 5V, the interrupt is triggered. As a result, a single button press causes a variable number of mode changes to occur, equal to the number of times the signal bounces before being read as a clean offset. The *changeMode()* function "debounces" the button-press signal by logging each time an interrupt is triggered. Upon being triggered, the function checks the log to see the last time an interrupt occurred and performs no action if the last interrupt occurred a very short time ago. We experimentally determined that a threshold of 200 ms yields a clean button press, and so, in our implementation, any interrupt that occurs within 200 ms of the previous interrupt is ignored. We did not use a delay for this implementation, as delaying the system for 200 ms results in a noticeable lag between pressing the button and observing a mode change. Instead, we log the system time to a variable, *last_interrupt_time*, using *millis()*, a built-in function that logs the system time in milliseconds. Each time a HIGH signal is received (including each bounce), the current clock time is subtracted from the previous logged time and the difference is checked for a delay of >200 ms. If the condition is true, it executes a mode change, otherwise it does nothing. The variable *last_interrupt_time* is then updated with the present clock time and the main *loop()* is re-entered once again.

3.4 Debounce Code Segment

```
// Debouncer
static unsigned long last_interrupt_time = 0;
unsigned long interrupt_time = millis();
// If interrupts come faster than 200ms, assume it's a bounce and ignore
if (interrupt_time - last_interrupt_time > 200)
{
  Serial.println("Triggered changeMode");
  if (mode == 5) {mode = 0;}
  else {mode += 1;}
}
last_interrupt_time = interrupt_time;
```

3.5 Adding the Potentiometer

The potentiometer was integrated as a piece of hardware that could adjust the speed of the bike light patterns (e.g. faster frequency of blinking). Our blinks were created with use of small time delays, and so speeding up or slowing down the rate of blinking is as simple as multiplying a given time delay by a (linear) factor.

Included at the start of the *loop()* function is a line to read an integer input from analog pin zero. This input is defined by the Arduino API as an integer value from 0 to 1023 that corresponds proportionally to voltages in the range 0V to 5V. Being that we want the potentiometer to be able to both speed up and slow down the rate of blinking, we defined the middle of the analog input range to correspond to a multiplicative factor of 1. This was achieved by simply dividing the input integer by 512. Thus,

$$dt = 100 \times \frac{input}{512}$$

where dt is the new time delay calculated by multiplying the factor $\frac{input}{512}$ with the nominal blink delay of 100 ms. When the potentiometer is positioned at the middle of the range, the analog pin returns a value of 512, which corresponds to a multiplicative factor of 1, and thus a delay of 100 ms. If the potentiometer is positioned to result in a higher or lower voltage on the analog pin, the multiplicative factor varies within the range [0:2], ultimately resulting in a time delay between zero ms (very high frequency) and 200 ms (very low frequency), respectively.

4 Conclusion

In this lab we found that even objectives which seem relatively simple can have unforseen difficulties arise along the way. The first of these we encountered was debouncing, which meant that we had to account for a very specific feature of the hardware in our code. Additionally, the problem of event handling also came into play. Our system was built around the use of delays but we wanted to be able to take in input as close to real time as possible. It turned out that this is a common enough issue that the arduino had a built in way of handling cases like this (interrupt), and we were able to get the results we wanted.

5 Appendices

5.1 Appendix 1: Source Code (Arduino C)

```
int ledPin = 13;
int interruptPin = 2;
int analogPin = 0;
int mode = 0;
int val = 0;
float dt = 100;

void setup() {
  // put your setup code here, to run once:
  pinMode(ledPin, OUTPUT);
  pinMode(ledPin-1, OUTPUT);
  pinMode(ledPin-2, OUTPUT);
  pinMode(ledPin-3, OUTPUT);
  pinMode(ledPin-4, OUTPUT);
  //pinMode(interruptPin, INPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), changeMode, HIGH);
  Serial.begin(9600);
  Serial.println("Completed setup");
}

void loop() {
  // Check analog input
  val = analogRead(analogPin);      // read the input pin
  dt = (int) 100*val/512;
  dt = abs(dt);
  Serial.println(dt);
  Serial.println(val);
  // All on
  if (mode == 0) {
    Serial.println("Entered mode 0");
    digitalWrite(ledPin, HIGH);
    digitalWrite(ledPin-1, HIGH);
    digitalWrite(ledPin-2, HIGH);
    digitalWrite(ledPin-3, HIGH);
    digitalWrite(ledPin-4, HIGH);
  }
  // All flashing
  else if (mode == 1) {
    Serial.println("Entered mode 1");
    digitalWrite(ledPin, LOW);
    digitalWrite(ledPin-1, LOW);
    digitalWrite(ledPin-2, LOW);
    digitalWrite(ledPin-3, LOW);
```

```
    digitalWrite(ledPin-4, LOW);
    delay(5*dt);
    digitalWrite(ledPin, HIGH);
    digitalWrite(ledPin-1, HIGH);
    digitalWrite(ledPin-2, HIGH);
    digitalWrite(ledPin-3, HIGH);
    digitalWrite(ledPin-4, HIGH);
    delay(5*dt);
  }
  // Rolling Down
  else if (mode == 2) {
    Serial.println("Entered mode 2");
    digitalWrite(ledPin, HIGH);
    delay(dt);
    digitalWrite(ledPin, LOW);
    digitalWrite(ledPin-1, HIGH);
    delay(dt);
    digitalWrite(ledPin-1, LOW);
    digitalWrite(ledPin-2, HIGH);
    delay(dt);
    digitalWrite(ledPin-2, LOW);
    digitalWrite(ledPin-3, HIGH);
    delay(dt);
    digitalWrite(ledPin-3, LOW);
    digitalWrite(ledPin-4, HIGH);
    delay(dt);
    digitalWrite(ledPin-4, LOW);
  }
  // Alternate LEDs Flashing
  else if (mode == 3) {
    Serial.println("Entered mode 3");
    digitalWrite(ledPin, LOW);
    digitalWrite(ledPin-1, HIGH);
    digitalWrite(ledPin-2, LOW);
    digitalWrite(ledPin-3, HIGH);
    digitalWrite(ledPin-4, LOW);
    delay(5*dt);
    digitalWrite(ledPin, HIGH);
    digitalWrite(ledPin-1, LOW);
    digitalWrite(ledPin-2, HIGH);
    digitalWrite(ledPin-3, LOW);
    digitalWrite(ledPin-4, HIGH);
    delay(5*dt);
  }
  // Bouncing
  else if (mode == 4) {
    Serial.println("Entered mode 4");
    // Roll down
```

```
    digitalWrite(ledPin, HIGH);
    delay(dt);
    digitalWrite(ledPin, LOW);
    digitalWrite(ledPin-1, HIGH);
    delay(dt);
    digitalWrite(ledPin-1, LOW);
    digitalWrite(ledPin-2, HIGH);
    delay(dt);
    digitalWrite(ledPin-2, LOW);
    digitalWrite(ledPin-3, HIGH);
    delay(dt);
    digitalWrite(ledPin-3, LOW);
    digitalWrite(ledPin-4, HIGH);
    delay(dt);
    digitalWrite(ledPin-4, LOW);
    // Roll up
    digitalWrite(ledPin-3, HIGH);
    delay(dt);
    digitalWrite(ledPin-3, LOW);
    digitalWrite(ledPin-2, HIGH);
    delay(dt);
    digitalWrite(ledPin-2, LOW);
    digitalWrite(ledPin-1, HIGH);
    delay(dt);
    digitalWrite(ledPin-1, LOW);
  }
  // All off
  else if (mode == 5) {
    Serial.println("Entered mode 5");
    digitalWrite(ledPin, LOW);
    digitalWrite(ledPin-1, LOW);
    digitalWrite(ledPin-2, LOW);
    digitalWrite(ledPin-3, LOW);
    digitalWrite(ledPin-4, LOW);
  }
}

void changeMode() {

  // Debouncer
  static unsigned long last_interrupt_time = 0;
  unsigned long interrupt_time = millis();
  // If interrupts come faster than 200ms, assume it's a bounce and ignore
  if (interrupt_time - last_interrupt_time > 200)
  {
    Serial.println("Triggered changeMode");
    if (mode == 5) {mode = 0;}
    else {mode += 1;}
```

```
    }
    last_interrupt_time = interrupt_time;

}
```