Sam Wisnoski, Oscar Bao, Chris Nie (Group 18)
Principles of Integrated Engineering Section 2

Mini Project 1: Building a Bike Light

I. Introduction:

Our first mini- project focused around using an Arduino, basic breadboarding, and at least three LEDs to construct a multi-modal bike light. We chose to use a total of eight LEDs to construct a total of six modes:

- All OFF
- All ON
- Binary flash
- Bouncing
- Charging
- Lava lamp

Additionally, we used a button to be able to switch between each of the LED modes. We use a potentiometer as analog input to modify the speed of these modes. By rotating the potentiometer, we can switch between a “fast flash” and a “slow flash” without changing the mode via the push button. These modifications apply to the last four of the six modes listed above.

II. Calculations

In order to properly wire our LEDs, we need to calculate the necessary resistor needed to provide the LED with the correct amount of current. We used the provided LED product pages to obtain the voltage dropped across the LED and its maximum operating current. These values are summarized in figure 2.1.

Color	Forward Voltage V_f / V	Maximum current I_{max} / mA
red	2V	20 mA
green	1.9V	2 mA
yellow	2.1V	10 mA
blue	3.2V	20 mA

Figure 2.1 / Product Page Data

Each LED is on its own pin. They are connected to the 5V rail in parallel. Therefore, each LED is supplied with 5V. We use resistors to drop the current to the maximum operational current. We found the values of these resistors using Ohm's law. Refer to Figure 2.2 for our final theoretical values.

$$V = IR \quad \rightarrow \quad \frac{5 - V_f}{I_{max}} = R$$

Color	Calculated Resistance R / Ω
red	310 Ω
green	1550 Ω
yellow	145 Ω
blue	90 Ω

Figure 2.2 / Calculated Resistance

III. Implementation:

After calculating the proper resistances for each of the LEDs, we created a circuit diagram for our LEDs, button, and potentiometer systems. This diagram is shown in figure 3.1.

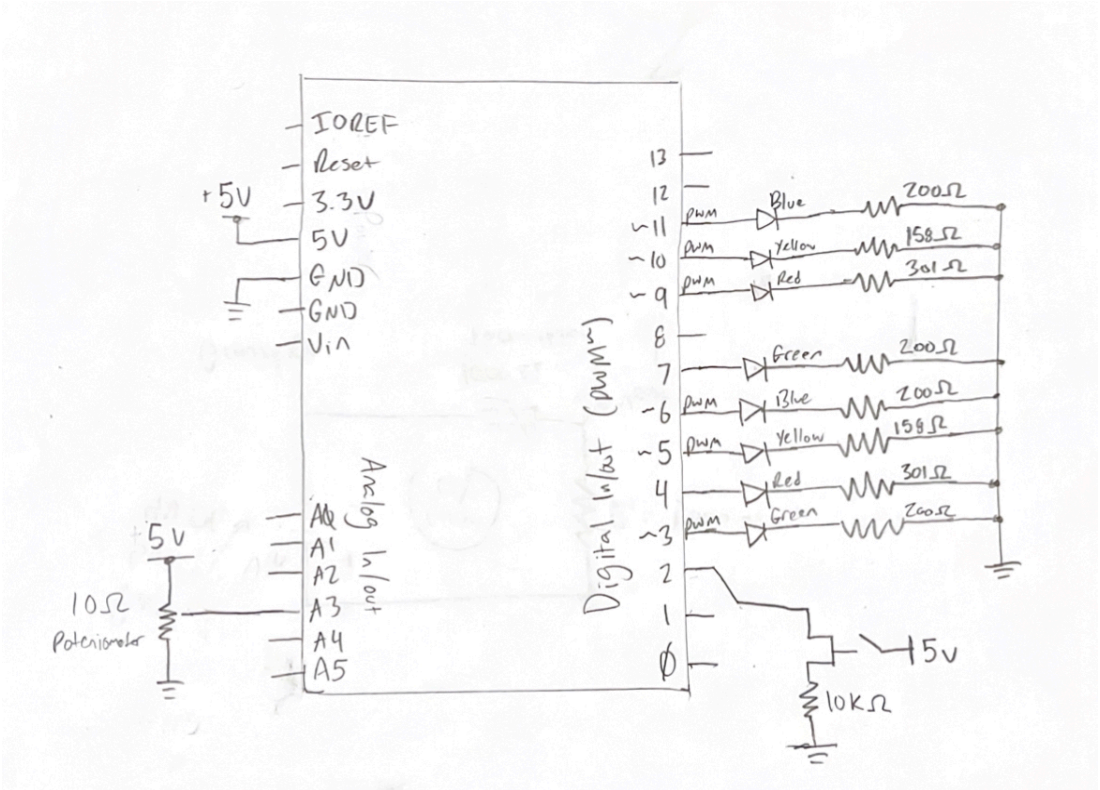

Figure 3.1 / Circuit Diagram

We tested our LEDs with the closest resistors we could find for the calculated resistor values (figure 2.2). For red, calculated at 310Ω, we tested a 301Ω resistor. For blue, calculated at 90Ω, we tested a 100Ω resistor. For yellow, calculated at 145Ω, we tested a 158Ω resistor, and for green, calculated at 1550Ω, we tested a 2000Ω resistor. While our yellow and red LEDs performed as expected, the blue LED was much brighter than the others, and the green LED was significantly dimmer. We experimented using different resistors until we were able to have all LEDs at relatively the same brightness. This led us to the final resistor values featured in figure 3.2

Color	**Used Resistances R / Ω**
red	301 **Ω**
green	200 **Ω**
yellow	158 **Ω**
blue	200 **Ω**

Figure 3.2 / Used Resistance

After finalizing our resistor values, we hooked up all eight LEDs and resistors to our Arduino board and to ground as shown in figure 3.1. We then began to write and test each of our functions.

We designed our code structure to be function based, with each function corresponding to one mode, and we call the function we need in the main loop when needed. Specifically, the on and off functions are achieved by writing a for loop to turn each pin to LOW output. The Binary Flash mode is achieved using the millis() time counter and turning the LED on and off repeatedly. For the Bounce mode, we used the same counter and another variable to track which LED is currently on, and one more to track whether the LED should go forward or backward. For the Charge mode, we continue to build off the Bounce mode and introduce a new tracker to track how many LEDs have been permanently lit up, and use simple algebra to do the charging effect. Lastly, for the Lava mode we used analogWrite function, utilizing the PWM feature that some pins on the Arduino board has, and dynamically change the brightness of the LEDs. The entirety of this code can be found in the Appendix.

With all the modes coded, we then added a button to our circuit connected to 5v, ground, and a digital input as shown in figure 3.1. Pressing the button connected the digital input to 5v, and as such we were able to read whenever the button was pressed. We then modified our code to switch between each of the modes whenever the button was pressed using the finite state machine method. Specifically, we used a switch-case method and have one case for one mode, and a button detection code to decide when to switch modes. Between each mode, we switch off all of the LEDs to “reset” our display before each mode starts. Additionally, we added a delay to the button so that the button would not trigger more than once for a single press.

Lastly, we used an analog input to modify the behavior of our bike light in an interesting way. We chose to use a potentiometer to vary the amount of current that was received by analog port 3. Next, for all of our modes that used a time constant to regulate them (flashLED, bounceLED, chargeLED, and lavaLED), we made their time constants directly proportional to the current received by analog port 3. When we increase the potentiometer's resistance (decreasing the current), the modes slow down, and when we decrease the resistance (increasing the current), the modes speed up. A diagram of our potentiometer attached to our Arduino is in figure 3.1. The code to read the current to analog port 3 and regulate the speed of our modes is found in the Appendix.

IV. Demonstration

Here is a link to our video demonstrating the six different modes, the button, and the potentiometer. It was uploaded via Onedrive and is publicly accessible.

https://olincollege-my.sharepoint.com/:v:/g/personal/xbao_olin_edu/EccgVpj7wm1IiNPRt9QzhawBAI2B9077XSp9cZbBXc-XgQ?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGluaONvcHkifX0&e=ldpC0A

V. Reflection

We are satisfied with our bike light and have gained valuable insights that will benefit us throughout this course. This project enabled us to gain familiarity with Arduinos and class resources. We assessed the efficacy of various approaches to an Arduino program, identifying both successful strategies and those with limited efficacy. The practice we gained using documentation and reading datasheets will be advantageous in future projects.

We encountered several challenges throughout this project. The mismatch between our final and theoretical resistor values highlighted the complexities of applying theory to practice. For example, we had to use the "closest appropriate" resistors, such as substituting a 301 Ω resistor for a 310 Ω one. We also considered combining resistors in series to this end. Additionally, we observed discrepancies in data sheet values, such as the unusually high calculated 1550 Ω resistor for the green LED, an order of magnitude greater than the other resistors. Finally, we had to increase the resistance of the blue LED, to bring its resistance in line with the other LEDs.

We also identified areas for improvement in our project planning and execution. We could have avoided refactoring our code in Part 2 with better pre-planning of variables. Furthermore, our initial reliance on delay() despite our suspicions of its interaction with the push button led to malfunctioning code. More thorough testing would have provided concrete insights into the limitations of delay(), allowing us to avoid rewriting functions.

VI. Appendix

Arduino Code

```
// Setting up pins
const uint8_t SW1 = 2;
const uint8_t GRE1 = 3;
const uint8_t RED1 = 4;
const uint8_t YLW1 = 5;
const uint8_t BLU1 = 6;
const uint8_t GRE2 = 7;
const uint8_t RED2 = 9;
const uint8_t YLW2 = 10;
const uint8_t BLU2 = 11;

// Button status
bool SW1_went_low = true;
bool SW1_HIGH;

static const uint16_t pin_list[] = {GRE1, RED1, YLW1, BLU1, GRE2, RED2, YLW2,
BLU2};
static const uint8_t num_pins = 8;
const uint16_t BLINK_INTERVAL = 500; // Interval time for blink mode in ms
const uint16_t CHARGE_INTERVAL = 200; // Interval time for charge mode in ms

// Time stamps for different modes
uint32_t blink_time;
uint32_t charge_time;
uint32_t bounce_time;

bool charge_on = true; // Charge mode turn on status

// Lava mode variables
int BRIGHTNESS = 0;
int LAVA_INTERVAL = 10;
uint32_t lava_time;
bool INCREASING = true;

// Charge mode variables
int lit_counter = 0; // Which LED is on
int perm_lit_counter = 0;

double scale = 1; // Potentiometer controlled time interval scale (0-2)

bool forward_status = true; // Bounce mode forward status
```

```
enum states { // For switch case
  OFF, ON, FLASH, BOUNCE, CHARGE, LAVA
};
states state;


void setup() {
  // put your setup code here, to run once:
  pinMode(SW1, INPUT);
  pinMode(RED1, OUTPUT);
  pinMode(GRE1, OUTPUT);
  pinMode(YLW1, OUTPUT);
  pinMode(BLU1, OUTPUT);
  pinMode(RED2, OUTPUT);
  pinMode(GRE2, OUTPUT);
  pinMode(YLW2, OUTPUT);
  pinMode(BLU2, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);

  digitalWrite(LED_BUILTIN, LOW);
  digitalWrite(RED1, LOW);
  digitalWrite(GRE1, LOW);
  digitalWrite(YLW1, LOW);
  digitalWrite(BLU1, LOW);
  digitalWrite(RED2, LOW);
  digitalWrite(GRE2, LOW);
  digitalWrite(YLW2, LOW);
  digitalWrite(BLU2, LOW);

  blink_time = millis();
  charge_time = millis();
  lava_time = millis();
  bounce_time = millis();

  state = OFF;

}

void loop() {
  // put your main code here, to run repeatedly:
  scale = checkPotentiometer(); // Check the resistance of potentiometer every
loop

  switch (state) { // Different modes
```

```
    case OFF:
      offLED();

      SW1_HIGH = digitalRead(SW1) == HIGH; // Code to detect button pressing
      if (SW1_HIGH && SW1_went_low){ // If the switch is pressed and it was
released already
        state = ON;
        delay(200); // To prevent repeated triggering
        SW1_went_low = false;
      } else if (!SW1_HIGH && !SW1_went_low){
        SW1_went_low = true;
      }
      break;

    case ON:
      onLED();

      SW1_HIGH = digitalRead(SW1) == HIGH;
      if (SW1_HIGH && SW1_went_low){
        state = FLASH;
        delay(200);
        SW1_went_low = false;
      } else if (!SW1_HIGH && !SW1_went_low){
        SW1_went_low = true;
      }
      break;

    case FLASH:
      flashLED();

      SW1_HIGH = digitalRead(SW1) == HIGH;
      if (SW1_HIGH && SW1_went_low){
        state = BOUNCE;
        delay(200);
        SW1_went_low = false;
        offLED();
      } else if (!SW1_HIGH && !SW1_went_low){
        SW1_went_low = true;
      }
      break;

    case BOUNCE:
      bounceLED();

      SW1_HIGH = digitalRead(SW1) == HIGH;
```

```
      if (SW1_HIGH && SW1_went_low){
        state = CHARGE;
        charge_on = true; // Telling charge function this is its first run
        delay(200);
        SW1_went_low = false;
        offLED();
      } else if (!SW1_HIGH && !SW1_went_low){
        SW1_went_low = true;
      }
      break;

    case CHARGE:
      chargeLED();

      SW1_HIGH = digitalRead(SW1) == HIGH;
      if (SW1_HIGH && SW1_went_low){
        state = LAVA;
        delay(200);
        SW1_went_low = false;
        offLED();
      } else if (!SW1_HIGH && !SW1_went_low){
        SW1_went_low = true;
      }
      break;

    case LAVA:
      lavaLED();

      SW1_HIGH = digitalRead(SW1) == HIGH;
      if (SW1_HIGH && SW1_went_low){
        state = OFF;
        delay(200);
        SW1_went_low = false;
      } else if (!SW1_HIGH && !SW1_went_low){
        SW1_went_low = true;
      }
      break;
  }

}



void offLED() { // Turn off all LEDs
  for (int i = 0; i <= num_pins; i++) {
```

```
    digitalWrite(pin_list[i], LOW); // Loop through all pins to turn it off
  }
}

void onLED() { // Turn on all LEDs
  for (int i = 0; i <= num_pins; i++) {
    digitalWrite(pin_list[i], HIGH); // Loop through all pins to turn it on
  }
}

void flashLED() { // Flash the LEDs at rate determined by potentiometer input
  uint32_t t;

  t = millis();
  if (t >= blink_time + scale*BLINK_INTERVAL) { // If enough time has passed
since last flash...
    digitalWrite(RED1, !digitalRead(RED1));
    digitalWrite(BLU1, !digitalRead(BLU1));
    digitalWrite(GRE1, !digitalRead(GRE1));
    digitalWrite(YLW1, !digitalRead(YLW1));
    digitalWrite(RED2, !digitalRead(RED2));
    digitalWrite(BLU2, !digitalRead(BLU2));
    digitalWrite(GRE2, !digitalRead(GRE2));
    digitalWrite(YLW2, !digitalRead(YLW2));
    blink_time = t; // Document the new timestamp of flashing
  }
}

void chargeLED() { // LEDs "charge up" one by one and loop once all "charged"
  uint32_t t;
  t = millis();

  if (charge_on == true) { // If coming from another mode
    offLED();
    lit_counter = 0;
    perm_lit_counter = 0;
    charge_on = false;
  }

  if (perm_lit_counter == num_pins - 1) { // If all lit up
    offLED();
    lit_counter = 0;
    perm_lit_counter = 0;
  }
```

```
  if (t > charge_time+scale*CHARGE_INTERVAL && lit_counter == num_pins -
perm_lit_counter) {
    digitalWrite(pin_list[num_pins-perm_lit_counter - 1], HIGH);
    digitalWrite(pin_list[num_pins-perm_lit_counter - 2], LOW);
    lit_counter = 0;
    perm_lit_counter += 1;
  } else if (t > charge_time+scale*CHARGE_INTERVAL) {
    digitalWrite(pin_list[lit_counter-1], LOW);
    digitalWrite(pin_list[lit_counter], HIGH);
    lit_counter += 1;
    charge_time = millis();
  }
}

void bounceLED() { // LED "bounce" between the two ends
  uint32_t t;
  t = millis();

  if (charge_on == true) {
    offLED();
    lit_counter = 0;
    charge_on = false;
  }

  if (t > charge_time+scale*CHARGE_INTERVAL && lit_counter == num_pins) {
    lit_counter = 0;
    forward_status = !forward_status;

  } else if (t > charge_time+scale*CHARGE_INTERVAL) {
    if (forward_status == true) { // Governs if the LED travels forward or
backwards
      digitalWrite(pin_list[lit_counter-1], LOW);
      digitalWrite(pin_list[lit_counter], HIGH);
    } else if (forward_status == false) {
      digitalWrite(pin_list[num_pins - lit_counter], LOW);
      digitalWrite(pin_list[num_pins-lit_counter-1], HIGH);
    }
    lit_counter += 1;
    charge_time = millis();
  }
}

void lavaLED() { // LED change brightness like a lava lamp
  uint32_t t;
  t = millis();
```

```
  if (t > lava_time + scale*LAVA_INTERVAL) {
    analogWrite(BLU1, BRIGHTNESS);
    analogWrite(BLU2, BRIGHTNESS);
    analogWrite(GRE1, BRIGHTNESS);
    analogWrite(YLW1, BRIGHTNESS);
    analogWrite(YLW2, BRIGHTNESS);
    analogWrite(RED2, BRIGHTNESS);

    if (BRIGHTNESS>240) {
      INCREASING = false;
    }
    else if (BRIGHTNESS < 10) {
      INCREASING = true;
    }
    if (INCREASING == true) {
      BRIGHTNESS = BRIGHTNESS + 2;
    }
    else {
      BRIGHTNESS = BRIGHTNESS - 2;
    }

    lava_time = millis();
  }
}

double checkPotentiometer() { // Check potentiometer input and determine the
scale
  double potmeter_scale = 500;
  potmeter_scale = analogRead(A3); // analogRead range from 0-1000
  potmeter_scale = potmeter_scale/500;
  return potmeter_scale;
}
```