PSU Capstone Project Final Report

Intel Mood Light

Adrian Steele steelead@pdx.edu

Bander Alenezi alenezi@pdx.edu

Dusan Micic dmicic@pdx.edu

Waleed Alhaddad alhad@pdx.edu

https://github.com/steelead/Capstone-Intel-Mood-Light
https://github.com/steelead/Capstone-Intel-Mood-Light/wiki

Table of Contents

I. Introduction:

Our task was to develop a system that can adjust properties on a room based on a person's preset preferences. The system must have the ability identify a person who enters the room by some method requiring as little user involvement as possible, and store lighting profiles that correspond to different users. By creating this device, the user will be able to automate their room preferences and will have to interact with the system as little as possible.

MUST DO:

- The system must be able to identify distinct users who enter a room.
- The Arduino must match sensor input to one of multiple preprogrammed profiles saved within its memory.
- The Arduino must control a circuit which will light an LED strip according to the specifications of the the particular profile.

SHOULD DO:

- The Arduino should adjust its parameters and learn over time to account for changes in individuals
    - ie: adjusting profile weight gradually as a child grows up
- Sensors and main Arduino board should eventually communicate wirelessly to avoid the need for wires and provide a greater freedom of movement
- Users should be able to add, remove or modify profiles saved in the system.

II. Specifications:

In order to identify a person entering the room, we are using their weight as the input into our system. The idea behind this is that the person will be able to simply walk through the entrance upon where the scale would be mounted into the floor and would be able to automatically get their weight, and set the profile with very little to no user interaction required. Since we do not have the money to mount a scale into the floor, we are just modifying an existing scale, by removing all the original circuitry and developing our own, so we can use it as a sensor input into the Arduino.

We are using an LED Strip as our lighting. Ideally, this would be installed all around the room so that the the room's lighting would change accordingly to what the user specifies in their profile.

One of our specifications was to use an Arduino Due as the microcontroller of the system which is ARM based.

III. Design Approach

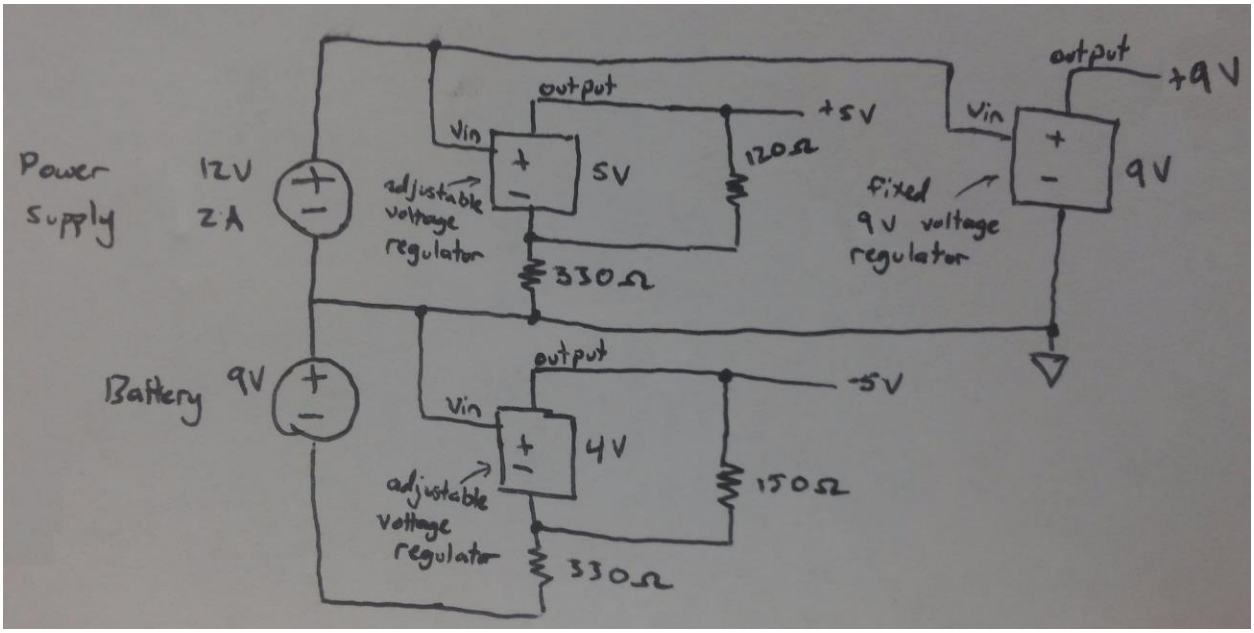
The system interfaces an Arduino Due with a load sensor to detect the weight of the person entering the room (in order to identify them) and configures an LED strip's output settings based on the person. The different profiles that the LED strip will be configured with will be input from the user interface (a program executed from the computer that will communicate between the user and the Arduino over bluetooth, allowing the user to add, remove, or modify profiles.

DESIGN STEPS:

- Interfacing the load sensor with the Arduino
- Interfacing the LED strip with Arduino
- Wireless connection from computer to the Arduino
- Connecting the system to a singular power source
- Method of getting user input profiles to Arduino (computer side user interface)
- Configuring user selectable LED color and brightness options

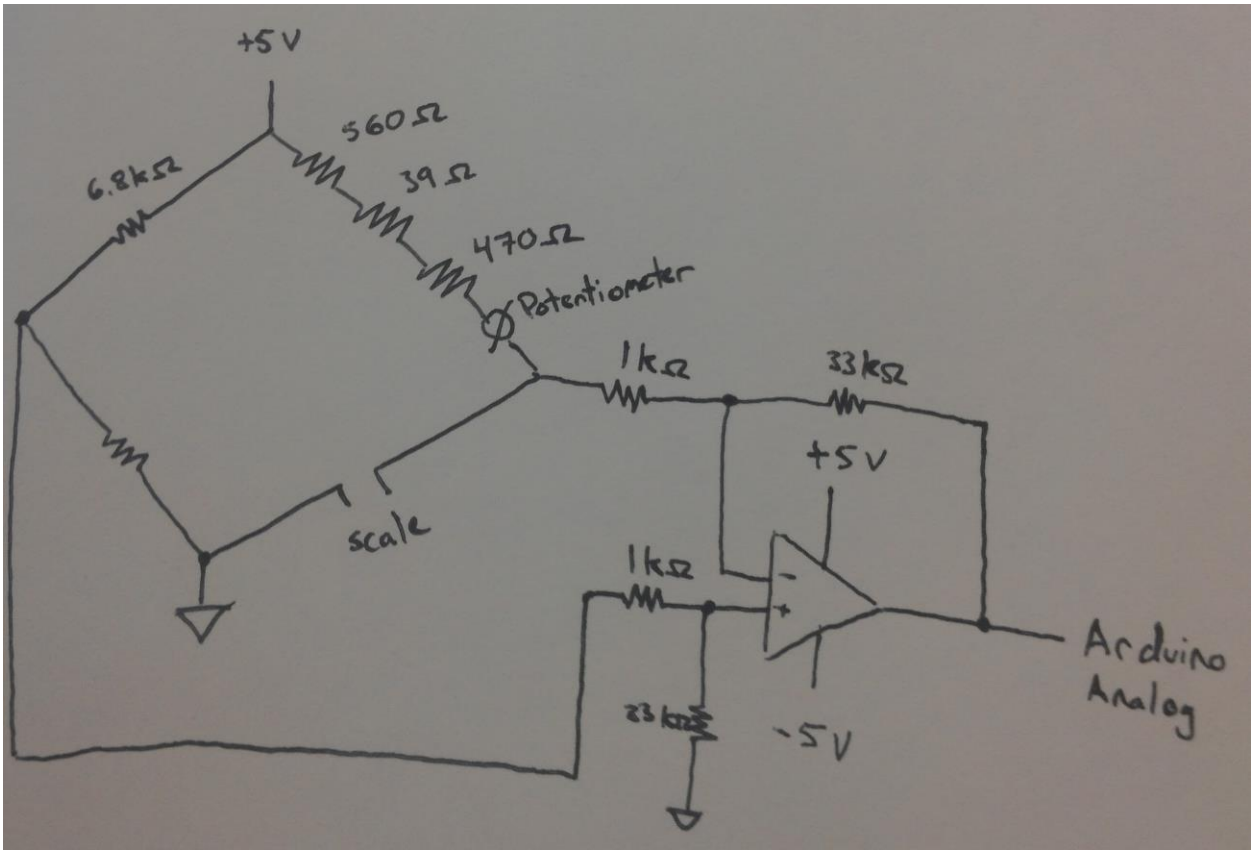
SECTIONS OF THE DESIGN:

Voltage Regulator Circuit



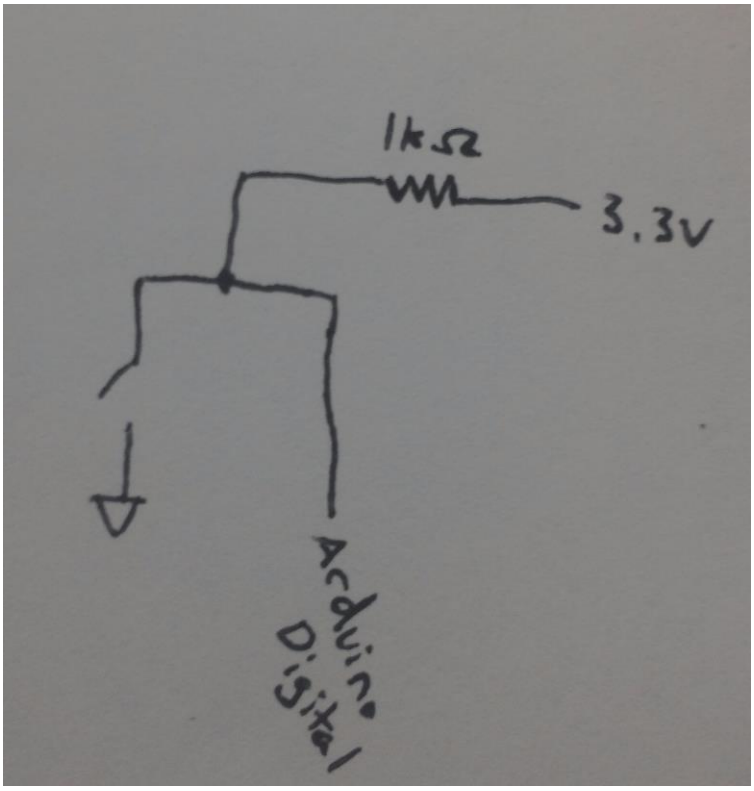
The purpose of this circuit is to supply the required power for all the components being used in the design. We are using several voltage regulators to achieve the necessary voltages for all the components in our design. The inputs to this circuit is a 12V 2A power supply (that plugs into the wall) and a 9V battery. The output voltages produced are -5V, 5V, 9V, which will be sent to the operational amplifier, logic level shifter, and Arduino Due. Additionally we are splitting off the 12V to the LED strip as well which requires 12V and at least 1A.

Wheatstone Bridge with Differential Operational Amplifier



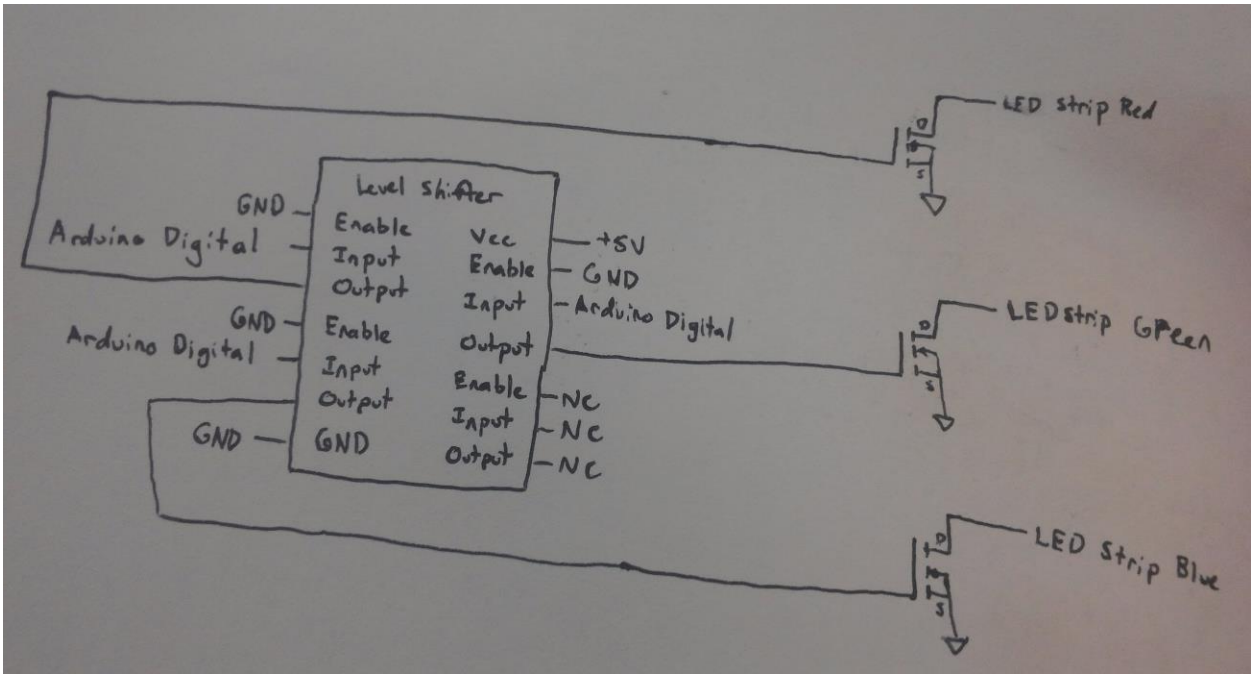
We are using a wheatstone bridge in order to measure the change in resistance of the strain gauge on the scale. When someone steps on the scale, the resistance will change on the scale, producing a change in voltage across the wheatstone bridge. The voltages at each side of the wheatstone bridge are sent to an operational amplifier set up to take the difference in of the inputs, and amplify it so the change in voltage is large enough for the Arduino to read the value.

Scale Button Circuit



The purpose of our button circuit is to tell if there is anyone standing on the scale. If the button is pressed (someone is standing on the scale) then the Arduino Due input will grounded. If the button is not pressed, then the Arduino Due input will read 3.3V.

Level Shifter with Mosfets Circuit



A logic level shifter is used to step up the output voltage of the Arduino from 3.3V to 5V. The Arduino Due runs at 3.3V, and consequently, the maximum voltage the input/output pins can tolerate is 3.3V. A level shifter is needed because the mosfets require 5V into the gate input in order to fully operate, so the outputs of the Arduino due must be shifted up to 5V. The mosfets are used as switches to send PWM signals to the LED strip in order to control the red, blue, and green brightness values in the LED strip.

IV. Testing:

The tested section of the design	**How it was tested**	**pass/no pass**
The button	Hook the button to a multimeter and when pushing the button it shows there is a connection.	pass

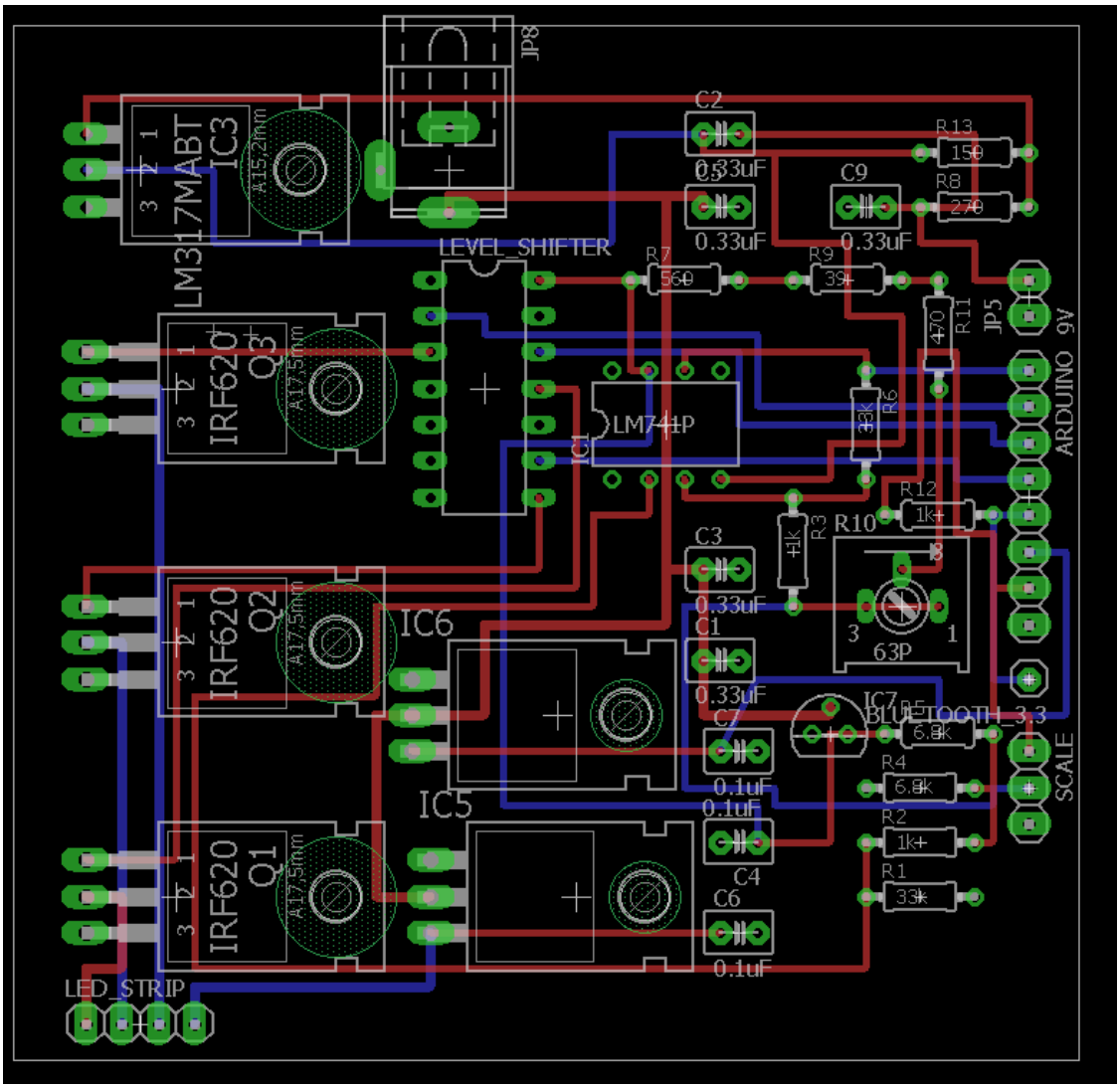
The button circuit	When the button is pressed, the Arduino digital input receives 0 V, when the button is not pressed, the Arduino digital sees 3.3 V	pass
Strain Gauge Circuit (wheatstone bridge and op amp)	Standing on the scale that is connected to Arduino program and check the output on the serial monitor. It should give the right weight of the person.	pass
The logic level shifter	Connect the level shifter to Arduino. It should bump up the 3.3V outputs of the Arduino Due to 5V. The output of the level shifter can be checked using a multimeter.	pass
The amplifier circuit	Check the output voltage of the op amp compared to the difference in the input voltages going into the op amp.	pass
The Leds strip to the due	Connecting the led strip to the Arduino due using Arduino program to adjust leds colors.	pass
Strain gauge scale weights to set LED strip colors through Arduino Due connection	Set a couple profiles and have the corresponding people step onto the scale and see if the strip gives the right colors for a specific weight profile.	pass
Bluetooth Connection	Send data between the computer and the Arduino over a bluetooth connection	pass
Persistent profiles upon reboot of the Arduino	Store a profile on the Arduino, and then reboot it and see if the profile is still saved on it.	pass

V. Future Work

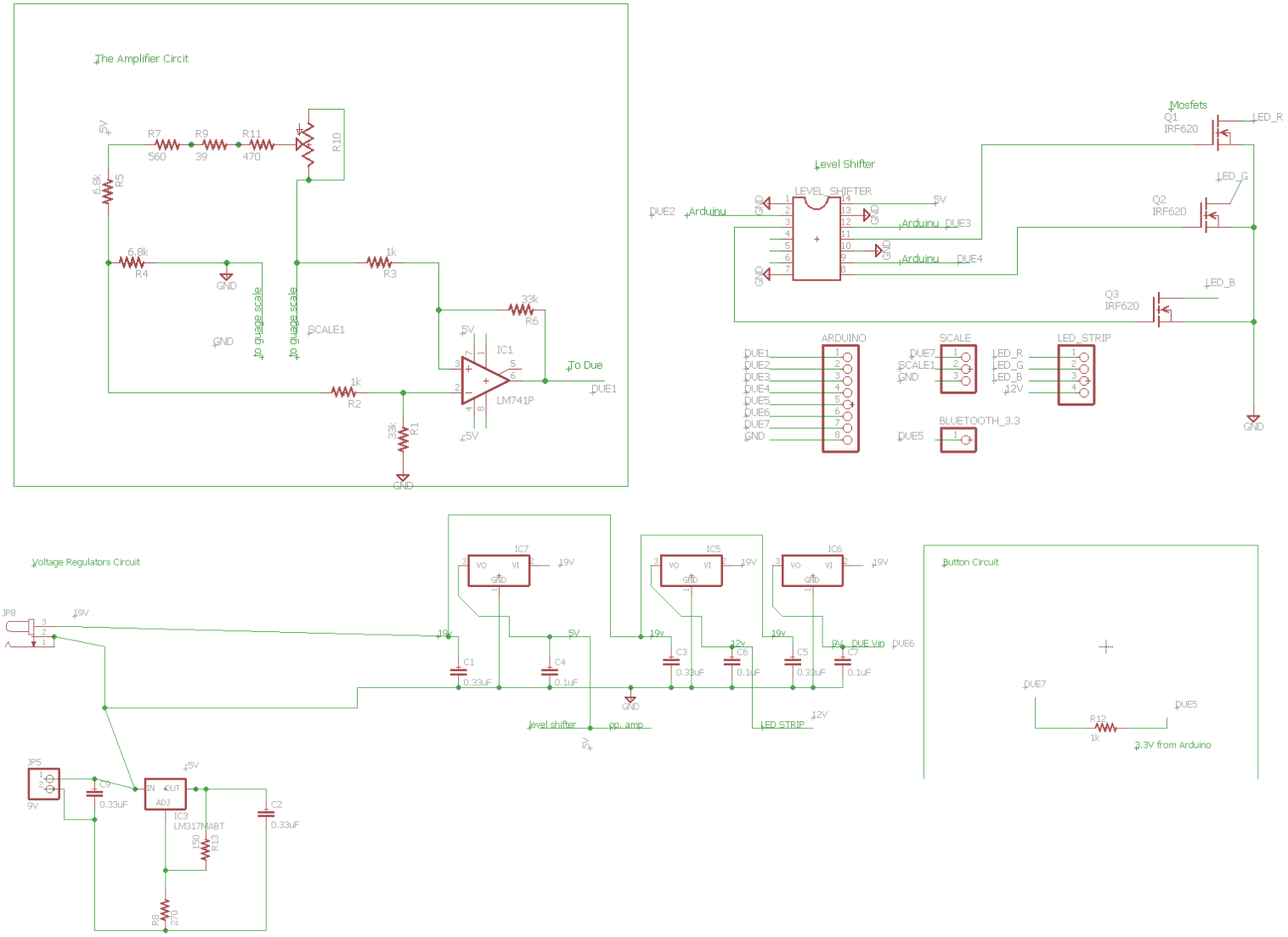
1. Adding another sensor such as “height sensor” to support the identification of the personal profile.
2. Using a more advanced interface device such as a touchscreen.
3. Add supporting sensors such as a temperature and humidity sensor to be set the user and controlled by the system.
4. Add new features to the device such as setting up music profile to be played alongside the light.
5. Add an option of voice command to interface with the device.

VI. Appendices

PCB BOARD AND LAYOUT:



SCHEMATIC:



CODE:

Arduino code:

```
/*
 * PSU Capstone Intel Mood Light 2015-2016
 * This program runs on the Arduino to control the LED strip and get input from a scale to identify users,
 * and input over bluetooth to add, modify and remove user profiles.
 *
 * Adrian Steele (steelead@pdx.edu)
 * Bander Alenzi (alenezi@pdx.edu)
 * Dusan Micic (dmicic@pdx.edu)
 * Waleed Alhaddad (alhad@pdx.edu)
```

```
 *
 * Dependencies:
 * arduino LinkedList library
 * arduino DueFlashStorage library
 */

#include <LinkedList.h>
#include <DueFlashStorage.h>
DueFlashStorage dueFlashStorage;


//1 to print debug statements to Serial (serial monitor), 0 to turn off
#define DEBUG 1
#define DEBUG_WEIGHT 0

#ifdef DEBUG
  #define DEBUG_PRINT(x)  Serial.print(x)
#else
  #define DEBUG_PRINT(x)
#endif

#ifdef DEBUG_WEIGHT
  #define DEBUG_DELAY(x) delay(x)
#else
  #define DEBUG_DELAY(x)
#endif

#define SCALE_ANALOG_PIN 0              //analog pin on the arduino to get weight readings from
#define REDPIN 9                        //LED strip digital output pin for red color settings
#define GREENPIN 10                     //LED strip digital output pin for green color settings
#define BLUEPIN 11                      //LED strip digital output pin for blue color settings
#define PERSON 8                        //digital input pin for button to tell if there is a person on the
scale
#define MULTIPLIER 2.43                 //scale analog input multiplier to convert to lbs (based on the
resolution of the scale)
#define MAXUSERLEN 50                   //maximum characters allowed for user profile names
#define NUMSAMPLES 500                  //number of weight samples to take for checking the weight of the
person on the scale
#define CALIBRATE_WEIGHT_LED_PIN 13     //LED pin output of the LED attached to the scale to tell when the
weight has been accuratly calculated
#define RANGE 20                        //acceptable range between readings of the weight (because scale is not
perfectly accurate each reading)

float weightSample[NUMSAMPLES];     //array of samples to average when calculating the weight of the person
stepping on the scale
int i;
int lastVal = 0;                    //weight of the last person who stepped on the scale
int baseVal;                        //base weight scale anlaog reading when there is nobody on the scale
String input = "";                  //holds input from messages sent to us over bluetooth
int currentMenu = 1;                //holds what menu the user interface program is in (to determine what
actions to perform)
int storeWeight = 0;                //whether to store the weight as a new profile or set the room lighting
int editProfileIndex = 0;           //index of the profile to edit



//structure for containing user profile info
struct Database
{
  char user[MAXUSERLEN];
  int weight;
  int color;
};
typedef struct Database database;

LinkedList<database*> profiles = LinkedList<database*>();     //linked list holding all the user profiles
```

```
database *newProfile;                                          //pointer to a new profile
int numProfiles;                                               //number of profiles we have stored


//runs once then transitions to loop()
void setup()
{
  //setup listening on Bluetooth connection (to modify profile info from computer script)
  Serial3.begin(9600);

  //setup debug printing to serial monitor
  #ifdef DEBUG
    Serial.begin(9600);
  #endif
  DEBUG_PRINT("Connected to bluetooth\n");

  //set up pin modes on arduino
  pinMode(PERSON, INPUT);
  pinMode(REDPIN, OUTPUT);
  pinMode(GREENPIN, OUTPUT);
  pinMode(BLUEPIN, OUTPUT);
  pinMode(CALIBRATE_WEIGHT_LED_PIN, OUTPUT);

  //test LED strip and weight calibration LED
  analogWrite(REDPIN, 255);
  analogWrite(GREENPIN, 255);
  analogWrite(BLUEPIN, 255);
  digitalWrite(CALIBRATE_WEIGHT_LED_PIN, HIGH);
  delay(500);
  analogWrite(REDPIN, 0);
  analogWrite(GREENPIN, 0);
  analogWrite(BLUEPIN, 0);
  digitalWrite(CALIBRATE_WEIGHT_LED_PIN, LOW);

  //get user profiles from flash memory
  //check if we have written to the flash before
  uint8_t firstRun = dueFlashStorage.read(0);
  //if we have not written to the memory location before, the byte value will be 255
  if(firstRun)
  {
    DEBUG_PRINT("Have not written to flash before, no profiles in memory yet\n");
    dueFlashStorage.write(0,0);    //write 0 to byte at address 0 to signal that we have written to the flash
before
    byte b[sizeof(int)];
    unsigned int currentAddress = 4;
    numProfiles = 0;
    memcpy(b, &numProfiles, sizeof(int));
    dueFlashStorage.write(currentAddress, b, sizeof(int));     //write that the number of profiles we have
stored is 0
  }

  else
  {
    DEBUG_PRINT("Flash has been written to before, retrieving profiles.\n");
    //get number of profiles (stored at address 4)
    unsigned int currentAddress = 4;
    byte *b = dueFlashStorage.readAddress(currentAddress);
    memcpy(&numProfiles, b, sizeof(int));

    //get profiles (stored in following addresses)
    currentAddress += (sizeof(int) * 4);
    database *tempProfile;
    for(i = 0; i < numProfiles; ++i)
    {
      tempProfile = new database;
      byte *b2 = dueFlashStorage.readAddress(currentAddress);
```

```
      memcpy(tempProfile, b2, sizeof(database));
      profiles.add(tempProfile);
      currentAddress += (sizeof(database) * 4);
    }

    DEBUG_PRINT("Number of profiles retrieved from memory: ");
    DEBUG_PRINT(numProfiles);
    DEBUG_PRINT("\n");
  }
}


//this code will loop infinitely
void loop()
{
  //Get input from computer (over bluetooth)
  if (Serial3.available() > 0)  // do nothing if no serial connection
  {
    //serial connection sends one byte (character) at a time, grab the byte and append to string buffer
    char state = Serial3.read();
    input += state;

    //comma delimiter to signal end of message
    if (state == ',')
    {
      DEBUG_PRINT("Message over bluetooth recieved: ");
      DEBUG_PRINT(input);
      DEBUG_PRINT("\n");

      //messages sent from computer are in key value pairs separated by an '=' without spaces
      String inputKey = input.substring(0, input.indexOf('='));
      String inputVal = input.substring(input.indexOf('=')+1, input.indexOf(','));

      if(inputKey == "menu")
      {
        currentMenu = inputVal.toInt();

        //send over bluetooth all the profiles we have in the system
        if (currentMenu == 2)
        {
          DEBUG_PRINT("currentMenu = 2, returning profiles to user interface program.\n");
          Serial3.print("numProfiles=");
          Serial3.println(numProfiles);

          for(i = 0; i < numProfiles; ++i)
          {
            Serial3.print("user=");
            Serial3.print(profiles.get(i)->user);
            Serial3.print(", weight=");
            Serial3.print(profiles.get(i)->weight);
            Serial3.print(", color=");
            Serial3.println(profiles.get(i)->color);
          }
        }
      }

      else if(inputKey == "name" && currentMenu == 1)
      {
        DEBUG_PRINT("currentMenu = 1, got new profile name.\n");
        newProfile = new database;
        inputVal.toCharArray(newProfile->user, MAXUSERLEN);
      }

      else if(inputKey == "color" && currentMenu == 1)
      {
        DEBUG_PRINT("currentMenu = 1, got new profile color.\n");
```

```
        newProfile->color = inputVal.toInt();
      }

      else if(inputKey == "weight" && currentMenu == 1)
      {
        DEBUG_PRINT("currentMenu = 1, got indication to store the next weight calibrated as a new profile\n");
        storeWeight = inputVal.toInt();
      }

      else if(inputKey == "profile" && currentMenu == 3)
      {
        DEBUG_PRINT("currentMenu = 3, got profile index to modify.\n");
        editProfileIndex = inputVal.toInt();
      }

      else if (inputKey == "name" && currentMenu == 3)
      {
        DEBUG_PRINT("currentMenu = 3, got name to modify.\n");
        database *aProfile;
        aProfile = profiles.get(editProfileIndex);
        inputVal.toCharArray(aProfile->user, MAXUSERLEN);
        profiles.set(editProfileIndex, aProfile);
        write2flash();
      }

      else if (inputKey == "color" && currentMenu == 3)
      {
        DEBUG_PRINT("currentMenu = 3, got color to modify.\n");
        database *aProfile;
        aProfile = profiles.get(editProfileIndex);
        aProfile->color = inputVal.toInt();
        profiles.set(editProfileIndex, aProfile);
        write2flash();
      }

      else if(inputKey == "weight" && currentMenu == 3)
      {
        DEBUG_PRINT("currentMenu = 3, got indication to modify an existing profile for next weight on
scale.\n");
        storeWeight = inputVal.toInt();
      }

      else if(inputKey == "profile" && currentMenu == 4)
      {
        DEBUG_PRINT("currentMenu = 4, removing a profile.\n");
        int removeIndex = inputVal.toInt();
        database * aProfile;
        aProfile = profiles.remove(removeIndex);
        DEBUG_PRINT("profile # - ");
        DEBUG_PRINT(removeIndex);
        DEBUG_PRINT("name - ");
        DEBUG_PRINT(aProfile->user);
        DEBUG_PRINT(" color - ");
        DEBUG_PRINT(aProfile->color);
        DEBUG_PRINT(" weight - ");
        DEBUG_PRINT(aProfile->weight);
        DEBUG_PRINT("\n");
        delete aProfile;
        --numProfiles;
        write2flash();
      }

      else
      {
        DEBUG_PRINT("Unrecognized input received from user program.");
        Serial3.println("Error: unrecognized input");
```

```
    }

    //Clear the serial3 (bluetooth) input buffer
    input = "";
  }
}



// Button on scale is pressed, someone on the scale
if(digitalRead(PERSON) == LOW)
{
  int redo = 1;
  int error = 0;
  float weightSum;

  DEBUG_PRINT("Someone is on the scale.\nSetting led pin on DUE high.\nCalibrating weight.\n");

  digitalWrite(CALIBRATE_WEIGHT_LED_PIN, HIGH);

  while(redo == 1)
  {
    weightSum = 0;
    redo = 0;
    for(i = 0; i < NUMSAMPLES; ++i)
    {
      delay(2);
      weightSample[i] = analogRead(SCALE_ANALOG_PIN);     //get sample
      weightSample[i] = (weightSample[i] - baseVal) * 2.43;     //convert weight to pounds

      weightSum += weightSample[i];

      //check if sample is outside of acceptable range
      if ((((weightSum / (i+1)) + RANGE) < weightSample[i]) ||
          (((weightSum / (i+1)) - RANGE) > weightSample[i]))
      {
        redo = 1;
        DEBUG_PRINT("first check\n");
        DEBUG_PRINT("average weight = ");
        DEBUG_PRINT(weightSum/(i+1));
        DEBUG_PRINT("\nOut of range sample = ");
        DEBUG_PRINT(weightSample[i]);
        DEBUG_PRINT("\n");
        DEBUG_DELAY(1000);
        break;
      }

      //if the person stepped off the scale before we were able to finish calibrating the weight
      if(digitalRead(PERSON) == HIGH)
      {
        DEBUG_PRINT("Person stepped off scale before able to fully calibrate weight.\n");
        error = 1;
        break;
      }
    }

    //if person stepped off the scale before we were able to finish calibrating the weight
    if (error == 1)
    {
      break;
    }

    if (redo == 0)
    {
      for(i = 0; i < NUMSAMPLES; ++i)
      {
```

```
        if ((((weightSum / NUMSAMPLES) + RANGE) < weightSample[i]) ||
            (((weightSum / NUMSAMPLES) - RANGE) > weightSample[i]))
            redo = 1;
            DEBUG_PRINT("second check\n");
            DEBUG_PRINT("average weight = ");
            DEBUG_PRINT(weightSum/(NUMSAMPLES));
            DEBUG_PRINT("\nOut of range sample = ");
            DEBUG_PRINT(weightSample[i]);
            DEBUG_PRINT("\n");
            DEBUG_DELAY(1000);
            break;
      }
    }

    if(redo == 0)
    {
      lastVal = int(weightSum / NUMSAMPLES);
      DEBUG_PRINT("Got weight on scale as :");
      DEBUG_PRINT(lastVal);
      DEBUG_PRINT("\n");
    }
  }

  //if person stepped off scale before we were able to finish calibrating the weight
  if(error == 1)
  {
    digitalWrite(CALIBRATE_WEIGHT_LED_PIN, LOW);
  }

  //compare lastVal to profiles
  if(storeWeight == 0 && error != 1)
  {
    DEBUG_PRINT("Comparing weight to profiles.\n");
    int userWeight = 0;
    int profileMatch = 0;

    //go through the list of profiles and find the closest match
    for(i = 0; i < numProfiles; ++i)
    {
      userWeight = profiles.get(i)->weight;
      if(abs(profiles.get(i)->weight - lastVal) < abs(profiles.get(profileMatch)->weight - lastVal))
      {
        profileMatch = i;
      }
    }
    DEBUG_PRINT("Closest profile weight is: user - ");
    DEBUG_PRINT(profiles.get(profileMatch)->user);
    DEBUG_PRINT(" weight - ");
    DEBUG_PRINT(profiles.get(profileMatch)->weight);
    DEBUG_PRINT("\n");

    //set led strip to profile color
    setLEDStrip(profiles.get(profileMatch)->color);
  }

  //store weight into new profile
  else if(storeWeight == 1 && currentMenu == 1 && error != 1)
  {
    DEBUG_PRINT("Storing weight as a new profile.\n");
    newProfile->weight = int(lastVal);
    profiles.add(newProfile);
    ++numProfiles;
    write2flash();
    Serial3.print("weight=");
    Serial3.println(newProfile->weight);
    storeWeight = 0;
```

```
    }

    else if(storeWeight == 1 && currentMenu == 3)
    {
      database *aProfile;
      aProfile = profiles.get(editProfileIndex);
      DEBUG_PRINT("Modifying existing profile weight: profile # - ");
      DEBUG_PRINT(editProfileIndex);
      DEBUG_PRINT(" user - ");
      DEBUG_PRINT(aProfile->user);
      DEBUG_PRINT(" color - ");
      DEBUG_PRINT(aProfile->color);
      DEBUG_PRINT(" oldweight - ");
      DEBUG_PRINT(aProfile->weight);
      DEBUG_PRINT("\n");
      aProfile->weight = int(lastVal);
      profiles.set(editProfileIndex, aProfile);
      write2flash();
      Serial3.print("weight=");
      Serial3.println(newProfile->weight);
      storeWeight = 0;
    }

    //delay loop wait for user to step off scale
    while(digitalRead(PERSON) == LOW)
    {
      DEBUG_PRINT("Done, blinking DUE LED, waiting for person to step off scale.\n");
      digitalWrite(CALIBRATE_WEIGHT_LED_PIN, HIGH);
      delay(200);
      digitalWrite(CALIBRATE_WEIGHT_LED_PIN, LOW);
      delay(200);
    }

  }

  // Button on scale not pressed, nobody on the scale
  else
  {
    baseVal = analogRead(SCALE_ANALOG_PIN);
  }


}


void write2flash()
{
  DEBUG_PRINT("Writing to flash.\n");
  byte b[sizeof(int)];
  unsigned int currentAddress = 4;
  int i;
  memcpy(b, &numProfiles, sizeof(int));
  dueFlashStorage.write(currentAddress, b, sizeof(int));

  currentAddress += (sizeof(int) * 4);
  for(i = 0; i < numProfiles; ++i)
  {
    byte b2[sizeof(database)];
    memcpy(b2, profiles.get(i), sizeof(database));
    dueFlashStorage.write(currentAddress, b2, sizeof(database));
    currentAddress += (sizeof(database) * 4);
  }

  return;
}
```

```
void setLEDStrip(int color)
{
  int r, g, b;

  //off
  if(color == 0)
  {
    DEBUG_PRINT("Setting LED strip off\n");
    r = 0;
    g = 0;
    b = 0;
  }

  //green
  else if(color == 1)
  {
    DEBUG_PRINT("Setting LED strip to green\n");
    r = 0;
    g = 255;
    b = 0;
  }

  //red
  else if(color == 2)
  {
    DEBUG_PRINT("Setting LED strip to red\n");
    r = 255;
    g = 0;
    b = 0;
  }

  //blue
  else if(color == 3)
  {
    DEBUG_PRINT("Setting LED strip to blue\n");
    r = 0;
    g = 0;
    b = 255;
  }

  //yellow
  else if(color == 4)
  {
    DEBUG_PRINT("Setting LED strip to yellow\n");
    r = 255;
    g = 255;
    b = 0;
  }

  //white
  else if(color == 5)
  {
    DEBUG_PRINT("Setting LED strip to white\n");
    r = 255;
    g = 255;
    b = 255;
  }

  //purple
  else if(color == 6)
  {
    DEBUG_PRINT("Setting LED strip to purple\n");
    r = 238;
    g = 130;
    b = 238;
```

```
  }

  //orange
  else if(color == 7)
  {
    DEBUG_PRINT("Setting LED strip to orange\n");
    r = 255;
    g = 165;
    b = 0;
  }

  analogWrite(REDPIN, r);
  analogWrite(GREENPIN, g);
  analogWrite(BLUEPIN, b);

  return;
}
```

Python User Interface Code:

```
# PSU Capstone Intel Mood Light 2015-2016
# This program functions as the user interface to allow the user to set profiles in the the arduino
# Connect to the arduino over a serial com port connection to send and receive information from the arduino
#
# Adrian Steele (steelead@pdx.edu)
# Bander Alenezi (alenezi@pdx.edu)
# Dusan Micic (dmicic@pdx.edu)
# Waleed Alhaddad (alhad@pdx.edu)

# Dependencies:
# windows pySerial library

# Tested on Windows 7, python 2.7.6, pyserial 2.7

import argparse
import errno
import os
import sys
import signal
import serial
import socket
import subprocess
import threading
import time


def main():
    """
        Main function, user should specify the COM port that the arduino is connected on when they begin the
program.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument('comPort', action='store', help="The serial com port that the arduino is connected to")
    args = parser.parse_args()
```

```
    #bind a serial connection to a computer communication port
    ser = serial.Serial(args.comPort, 9600, timeout=1)

    #open the bluetooth connection
    if ser.isOpen() == True:
        print "Connected to bluetooth module"
    else:
        print "Could not connect to bluetooth module"
        sys.exit()

    #get command from user and send to arduino over serial connection
    while True:
        print "Main Menu"
        print "1) Add profile"
        print "2) Edit existing profile"
        print "3) Remove profile"
        print "4) Quit"
        cmd = int(raw_input("Enter the number of the command to execute: "))

        if cmd == 1:
            addProfile(ser)

        elif cmd == 2:
            editProfile(ser)

        elif cmd == 3:
            removeProfile(ser)

        elif cmd == 4:
            break



def addProfile(ser):
    """
        Add a new profile to the arduino
        Args:
          - ser: serial connection object to the arduino
        Returns:
            None
    """
    username = raw_input("Enter the profile name: ")
    print "Color options:"
    print "1) Green"
    print "2) Red"
    print "3) Blue"
    print "4) Yellow"
    print "5) White"
    print "6) Purple"
    print "7) Orange"
    color = int(raw_input("Enter the number of the color to set for the profile: "))
    print "Step on the scale to calibrate the user's weight."

    ser.write("menu=1,")
    ser.write("name=" + username + ",")
    ser.write("color=" + str(color) + ",")
    ser.write("weight=1,")

    while True:
        resp = ser.readline()
        if resp != '':
            break

    try:
        print "Weight calibrated to: %d" % int(resp.split('=')[1].strip())
    except IndexError:
```

```
        print "Bluetooth communication error with Arduino, please try again.\n"


def getProfiles(ser):
    """
        Gets a list of profiles currently stored in the Arduino
        Args:
          - ser: serial connection object to the arduino
        Returns:
            A tuple containing:
            - Integer representing number of profiles
            - List object containing dictionaries of the profiles
    """
    ser.write("menu=2,")
    while True:
        line = ser.readline()
        if line != '':
            break

    try:
        # first thing returned from arduino should be the number of profiles it has
        numProfiles = int(line.split('=')[1].strip())
    except IndexError:
        print "Bluetooth communication error with Arduino, please try again.\n"
        return (0, [])

    profiles = []

    for i in range(0, numProfiles):
        # each profile will be read in in one line (ending with a newline)
        line = ser.readline()

        try:
            # each item in the profile will be separated by a comma
            for item in line.split(','):
                #key value pair will be separated by an '='
                if "user=" in item:
                    user = item.split('=')[1].strip()
                elif "weight=" in item:
                    weight = int(item.split('=')[1].strip())
                elif "color=" in item:
                    color = int(item.split('=')[1].strip())

            newProfile = {"user": user, "weight": weight, "color": mapVal2Color(color)}
            profiles.append(newProfile)

        except IndexError:
            print "Bluetooth communication error with Arduino, please try again.\n"
            return (0, [])

    return (numProfiles, profiles)


def mapVal2Color(colorInt):
    """
        Maps an integer to a color
        Args:
          - colorInt: the integer value of the color to map
        Returns:
            returns a string of the mapped color value
    """
    colorDict = {1: "Green",
                 2: "Red",
                 3: "Blue",
```

```
                4: "Yellow",
                5: "White",
                6: "Purple",
                7: "Orange",
               }
    return colorDict[colorInt]


def editProfile(ser):
    """
        Edit an existing profile on the arduino
        Args:
          - ser: serial connection object to the arduino
        Returns:
            None
    """
    (numProfiles, profiles) = getProfiles(ser)

    if numProfiles == 0:
        print "No existing profiles found on the Arduino.\n"
        return

    print "There are currently %d profiles stored on the Arduino." % numProfiles

    for i in range(0, len(profiles)):
        print "Profile %d" % (i+1)
        print "    User: %s" % profiles[i]["user"]
        print "    Weight: %d" % profiles[i]["weight"]
        print "    Color: %s\n" % profiles[i]["color"]

    editIndex = int(raw_input("Enter the number of the profile you want to make edits to: ")) - 1

    if editIndex < 0 or editIndex >= len(profiles):
        print "Error: Specified profile number out of range."
        return

    print "Profile %d" % (editIndex+1)
    print "    1) User: %s" % profiles[editIndex]["user"]
    print "    2) Color: %s" % profiles[editIndex]["color"]
    print "    3) Weight: %d\n" % profiles[editIndex]["weight"]
    item = int(raw_input("Enter the number of the item you want to modify: "))

    if item < 1 or item > 3:
        print "Error: specified item to edit out of range (1-3)"
        return

    if item == 1:
        username = raw_input("Enter the profile name: ")
        ser.write("menu=3,")
        ser.write("profile=" + str(editIndex) + ",");
        ser.write("name=" + username + ",")

    elif item == 2:
        print "Color options:"
        print "1) Green"
        print "2) Red"
        print "3) Blue"
        print "4) Yellow"
        print "5) White"
        print "6) Purple"
        print "7) Orange"
        color = int(raw_input("Enter the number of the color to set for the profile: "))

        if color < 1 or color > 7:
            print "Error: specified color out of range (1-7)."
```

```
            return

        ser.write("menu=3,")
        ser.write("profile=" + str(editIndex) + ",")
        ser.write("color=" + str(color) + ",")

    elif item == 3:
        ser.write("menu=3,")
        ser.write("profile=" + str(editIndex) + ",")
        ser.write("weight=1,");
        print "Step on the scale to calibrate the user's weight."
        resp = ser.readline()
        while True:
            resp = ser.readline()
            if resp != '':
                break
        print [resp]
        print "Weight calibrated to: %d" % int(resp.split('=')[1].strip())


def removeProfile(ser):
    """
        Remove a profile from the arduino
        Args:
          - ser: serial connection object to the arduino
        Return:
            None
    """
    (numProfiles, profiles) = getProfiles(ser)

    if numProfiles == 0:
        print "No existing profiles found on the Arduino.\n"
        return

    for i in range(0, len(profiles)):
        print "Profile %d" % (i+1)
        print "    User: %s" % profiles[i]["user"]
        print "    Weight: %d" % profiles[i]["weight"]
        print "    Color: %s\n" % profiles[i]["color"]

    removeIndex = int(raw_input("Enter the number of the profile you want to remove: ")) - 1

    if removeIndex < 0 or removeIndex >= len(profiles):
        print "Error: Specified profile number out of range."
        return

    ser.write("menu=4,")
    ser.write("profile=" + str(removeIndex) + ",")


if __name__ == '__main__':
    main()
```

CODE DEPENDENCIES:

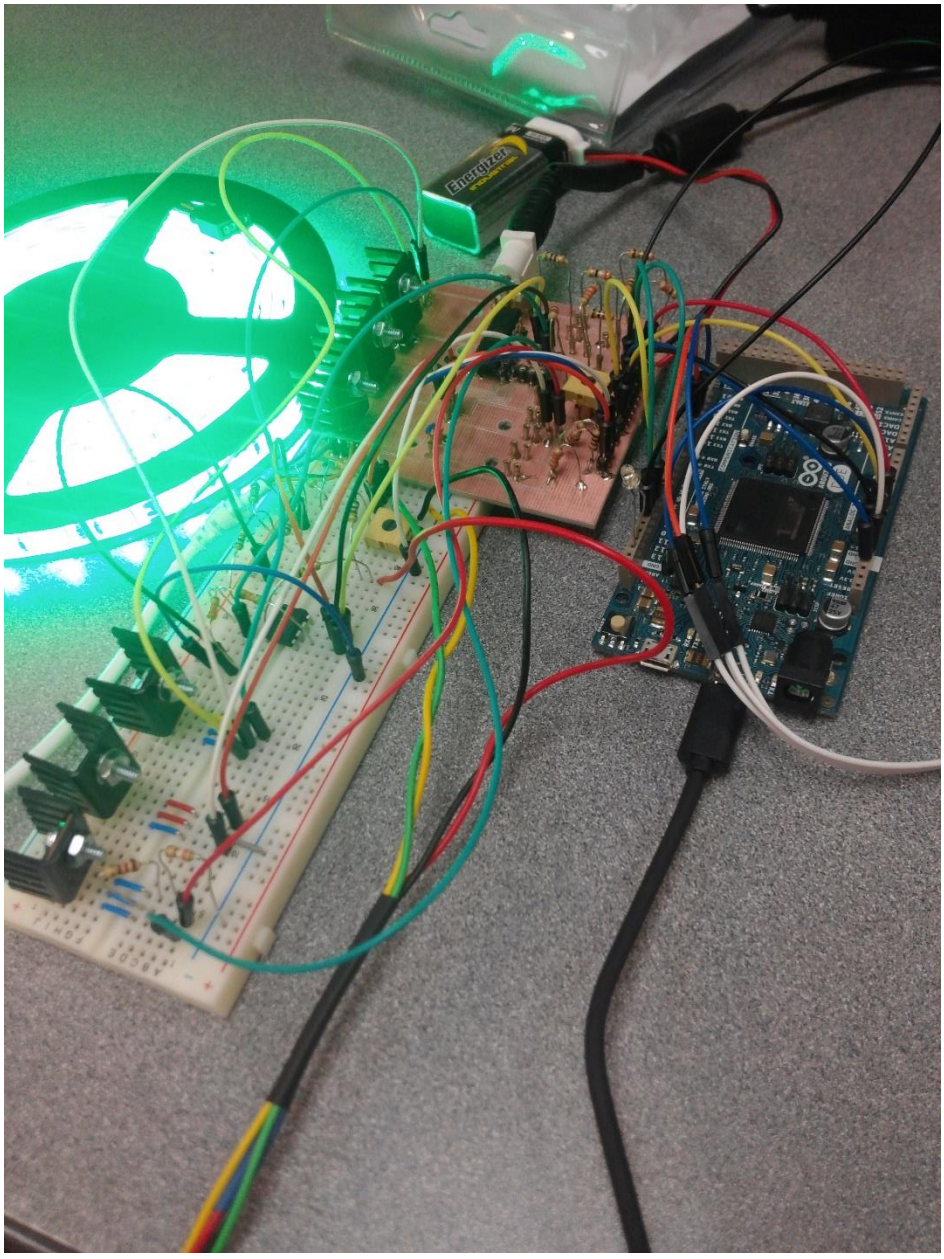
In order to run the Arduino code, you will need the Arduino IDE (version 1.6.6 or later), the DueFlashStorage Library (open source downloaded from github), and the LinkedList Library (open source downloaded from github).
In order to run the Python code for the user interface, you will need a Windows system (tested on windows 7), python for windows (tested on version 2.7.6) and the windows pySerial Library (open source, downloaded from python package index PyPi)

Arduino IDE:
https://www.arduino.cc/en/Main/Software
DueFlashStorage Library for Arduino:
https://github.com/sebnil/DueFlashStorage
LinkedList Library for Arduino:
https://github.com/ivanseidel/LinkedList

Python 2.7.6:
https://www.python.org/ftp/python/2.7.6/python-2.7.6.msi

pySerial Library for Python:
https://pypi.python.org/packages/any/p/pyserial/pyserial-2.7.win32.exe#md5=21555387937eeb79126cde25abee4b35

FINAL PRODUCT



Note: we had some problems with the PCB and some slight alterations to the design after printing the board so we had make some modifications on a breadboard to make the edits. The schematics in the "design approach" section are accurate, the schematics on for the PCB in the "appendices" section are slightly out of date.

RESOURCES AND REFERENCES:

Strain Gauges:
http://www.allaboutcircuits.com/textbook/direct-current/chpt-9/strain-gauges/

Differential Op Amp Circuit:
http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/ietron/diffa2.gif

LED Strip Wiring:
http://www.jerome-bernard.com/blog/2013/01/12/rgb-led-strip-controlled-by-an-arduino/