Cortland Bailey
05/01/2023
6193-3919

EE Design 1 Technical Report
Final Project
IoT Room Lighting

Introduction

My project is something that I designed with my house in mind. I wanted there to be lights underneath my cabinets in the kitchen that would activate as I walked towards the kitchen and have various outputs based on the ambient light conditions, as well as play music that corresponded to the light output.

All of this would require two independent circuits to include a motion sensor, ambient light sensor along with additional hardware for debugging and status displays. My goal was for this to be a real prototype for something I want to put in my house.

Methods

Motion Sensor

In order to capture motion I decided to use a PIR sensor. Our kit includes an SR501 sensor [1]. This is a sensor that outputs a digital signal based on motion sensor. This sensor is based around measuring differences in received infrared radiation between two separate windows in the chip [3]. Its tunable for sensitivity, duration of 'on' time, and whether it always turns off after this on time or if continuous motion keeps resetting this on time.

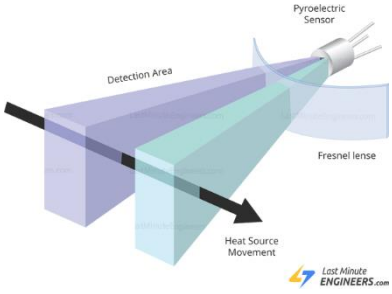


Figure 1 - PIR function visualization

Figure 2 - PIR

I elected to keep it on for roughly 30 seconds for this project because its just prototyping. It is tunable to around 3 minutes [3]. I also chose to have it remain on for constant motion to make testing easier as well. The output of this sensor is going to be used later in the methods.

Ambient Light Sensor

To measure the ambient light present I chose to use a simple photocell/photoresistor. This resistor varies its

internal resistance based on the light shining on the top of it, lower resistance means more light and vice versa.

The photocell I used is a 16-33kOhm unit, which measures closer to 36kOhms when in complete darkness and around 1k Ohm in the brightest light I could shine on it. This is plenty large enough for our ADC on the Raspberry Pi Pico to measure. In order to accomplish this I wired it in series with a traditional 10k Ohm resistor and if we input the voltage value between these two it allows a simple voltage divider to be calculated and then we have a useable range of values. These will later be used to determine the ultimate state that the output is in when motion is detected.

Microcontroller and IOT / WIFI

A core component of my project was to utilize wireless communications in order to have the motion sensor in a different location to the lights and speaker. I chose to find and use an ESP8266 based development board. These board are widely available, very inexpensive, and there is a lot of documentation and examples on the internet for them. I am using a NodeMCU ESP-12E board which has more capabilities than I needed but it was readily available and simple to use.

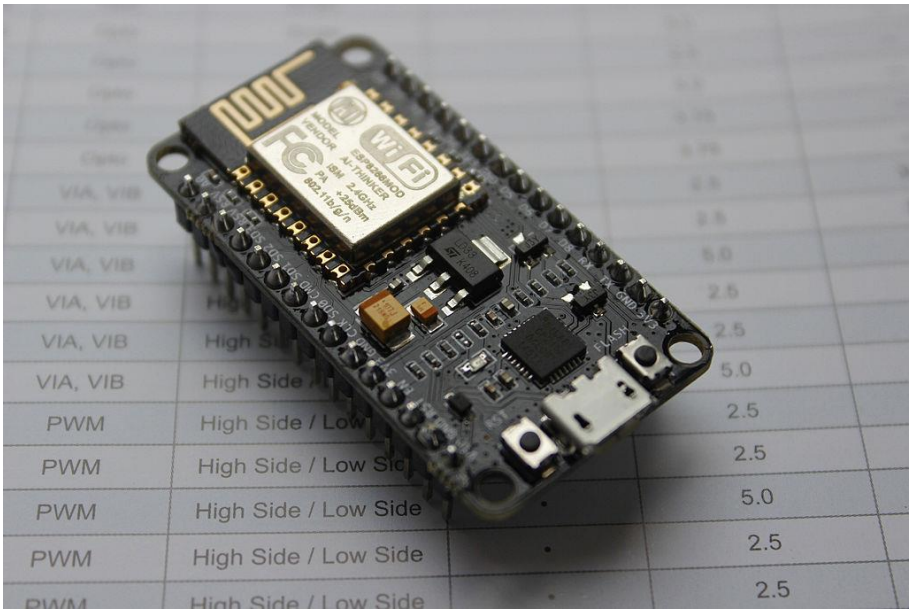


Figure 3 - NodeMCU ESP-12E built with the ESP8266

The central piece to the design is the Raspberry Pi Pico which runs the RP2040 microcontroller. The Pico has no wireless connection capabilities so my primary board utilizes both the Pico and the ESP in conjunction with each other. The ESP-12E could potentially run the bulk of this design on its own in future versions.

I used some of the Pico's many features, such as the build in SPI, I2C, and ADC capabilities. The ADC as mentioned earlier is used to digitize the photocell into a 10bit value that can be used to control overall output state. The I2C port I will use to control the LCD screen that will have displayed status messages. The SPI port I will use to communicate with an external digital to analog IC to generate an analog signal output. This would ideally be a musical chime or other sound.

The ESP8266 is a microcontroller that has a Wi-Fi antenna attached. Using examples provided in the Arduino IDE as well as Robotzeroone.com [7] I was able to create a file that uses one ESP to create a wireless access point that will receive data and another file that makes the other ESP a client on that wireless network to send data. The client will be the side with the motion sensor and the access point will be the ESP that shares a PCB with the Pico. Once the data is transmitted to the access point ESP, all it needs to send to the Pico is a simple High/Low for motion/no motion. I will create a connection for there to be I2C capability as well, just in case.

LED Strip

The LED strip I used is a WS2812B 3 meter strip. These are RGB LED's each with their own controller and all individually addressable. The current draw is quite high for this circuit, about 60mA per LED [14]. In order to keep the current draw to a minimum, this prototype will only utilize 5-6 LEDs at a time.

Since the LED strip contains a data input to control each of the LEDs, I utilized a video I found [12] that pointed me in the direction of a library [11] that gives background functions written in micropython that are for writing data out and controlling the strip. I evaluated these versus the functions and library that Micropython has 'built in' and found that these were more robust and user friendly.

Once this library is saved to the Pico is allows any GPIO pin to control the data line to the LED strip. The functions in the library handle all of the communication protocol with the strip and don't need a special pin or port. These allowed me to set multiple combinations of colors and light sequences as different outputs for different states.

LCD

The LCD screen I used is a simple 16x2 screen, but with an I2C "piggyback" attached. This add-on device allows us to control the screen with only 2 I2C-capable GPIO pins. The specific addon I used was a 'no-name' brand device and found a great write up on Tomshardware.com [15] that gives a detailed explanation of how to use the I2C protocol as well as a full MicoPython library to give simple functions for me to write out data to the screen as well as control its functions, such as the backlight and the cursor blinker.

Using these functions I wrote code to write the output state to the LCD, only refreshing it when the state changes so that the screen does not appear to be 'flashing' from re-writing constantly and to free up processor time. I also have an external button connected to a GPIO pin to control the backlight of the LCD so it can be turned on or off depending on the users preference.

Power Supply

To power this project, both boards will need a voltage source that is 5V. The Pico, both ESP's, and the SR501 PIR sensor all have built in voltage regulators that can accept an input voltage of around 4.5-10V, but all of the devices on both PCBs are designed to function around 5V. Using the Pico datasheet [1] and a LM2940 LDO voltage regulator I implemented the regulator circuit found the in the LM2940 datasheet [16].

Both the Pico and the ESP have several power pins on their boards. VSYS on the Pico and Vin on the ESP function as Input/Output for power according to the datasheets. The Pico has a very detailed explanation on how to power it externally, including a recommendation for a Zener diode on this pin to help prevent any potential issues if it is plugged in via micro-USB and externally powered. The Zener as well as the on-board power circuit will act as an "OR" for input voltage to the Pico. The ESP functions in a similar way and this gives a lot of flexibility for powering the circuit as either micro-USB or the external voltage source will power the entire PCB.

Sound

A requirement of the project is to create an analog output with at least 3 states. This is accomplished using the DAC IC mentioned earlier, a LT1661. My Analog signal is fed into an LM386 Audio amplifier IC and then into a simple 8 ohm speaker. This output is sufficient to play simple tones and other short sounds. Using the datasheets for both IC's [17][18], I used the design from an earlier module [19] to create the circuit to output the amplified analog signal.

The code to create this is going to use the SPI protocol. One problem with MicroPython is that its slow. The Pico has a fast clock speed, but it can't do enough in order to run our main function accomplishing all the other tasks AND outputting a decently high sampled signal to the DAC. This I learned in the previous DAC module. There are two potential ways to work around this problem as I found.

The first method is to run the Pico's microcontroller's two cores independently. Use one core to run our primary program and the other to handle the SPI output to the DAC. This is probably the best solution however I was not sold on this at the start.

The other method, which I am going to use, is to use the Timer() function that is built into the MicroPython. The RP2040 does not have multiple threads on the processor, but my research found that the Timer() function actually runs on the 2nd core automatically. This allows it to have a much higher sample rate and at decent output frequencies. The code can be found in the appendix, but it boils down to creating a sine wave generating function with a chosen frequency. This is then called by our timer function which has its own period. This period is just how often the callback function is called, in this case our sine wave generator code. This bypasses our problem of needing to call the generator within our main function explicitly, and the Pico runs it automatically, until its deactivated.

Hardware Layout

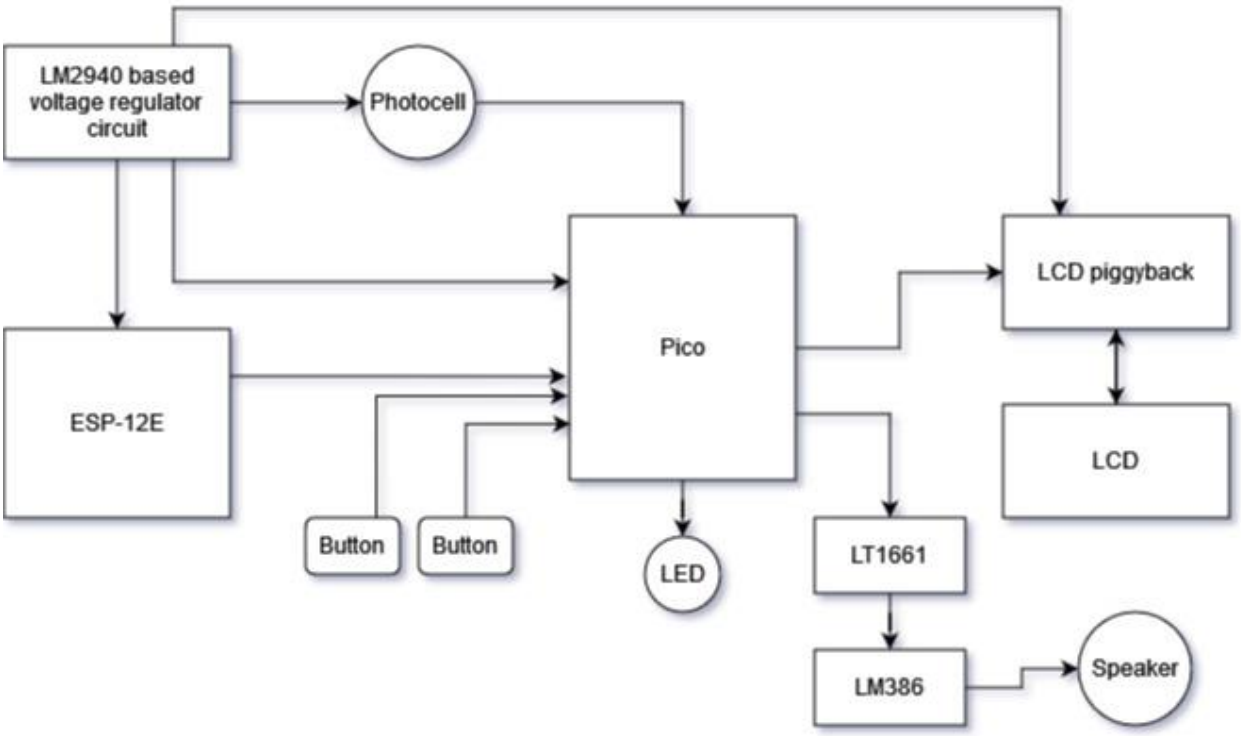


Figure 4 – Primary board hardware layout

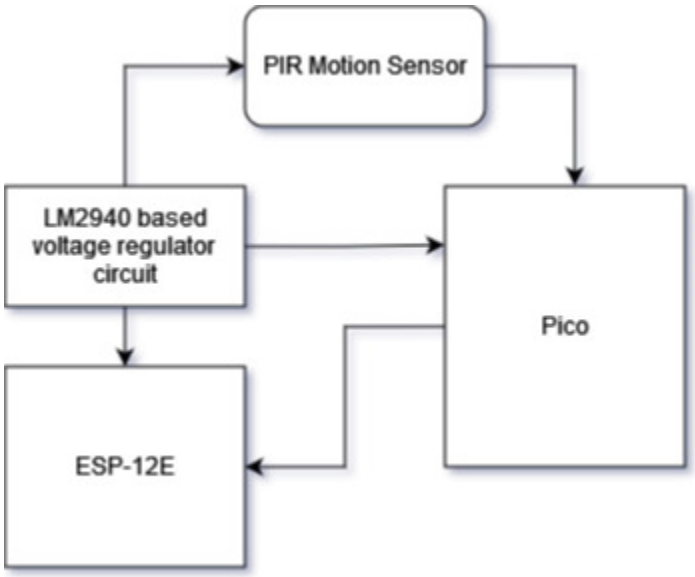


Figure 5 - Secondary board hardware layout

Software flowchart

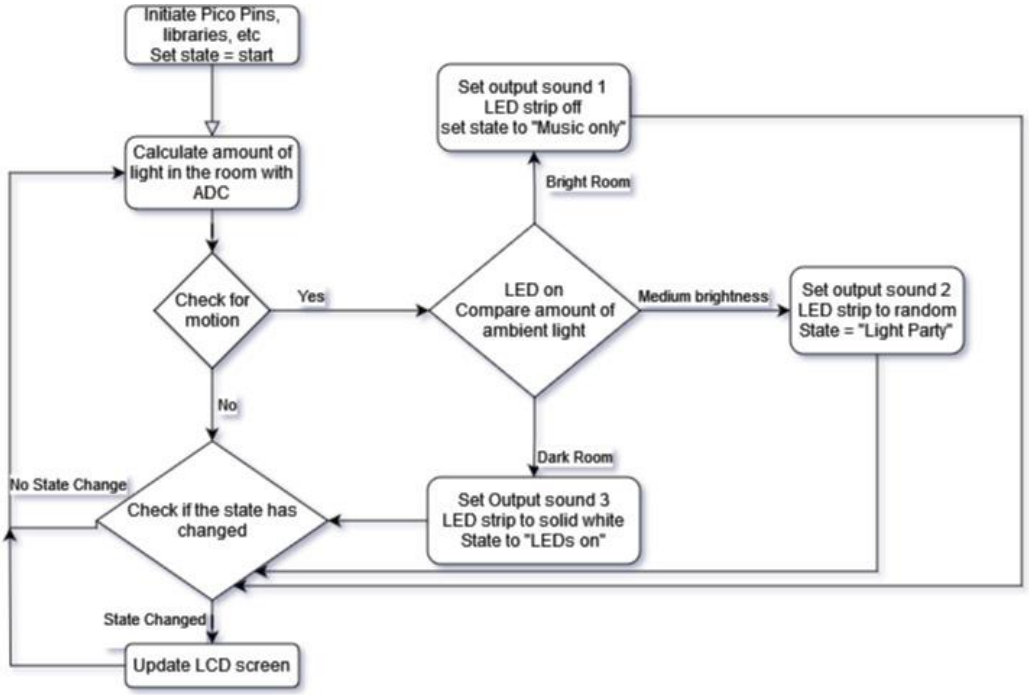


Figure 8 - Pico program flowchart

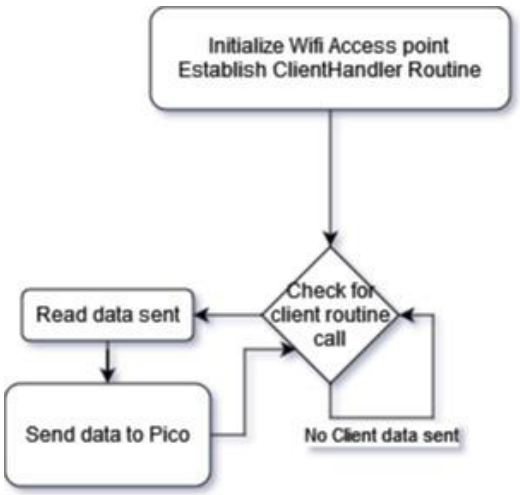


Figure 6 - Wifi Access Point Software

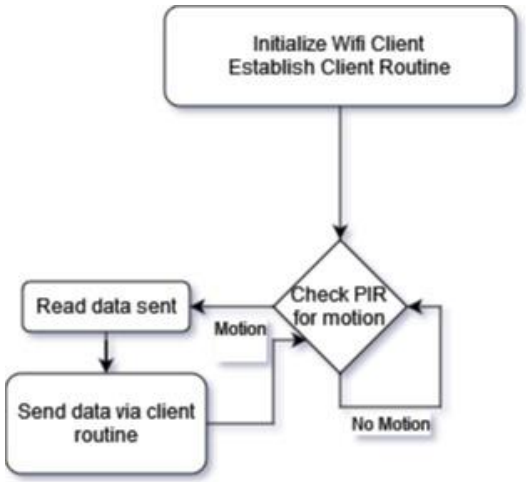


Figure 7 - Wifi Client Software

Altium Schematics

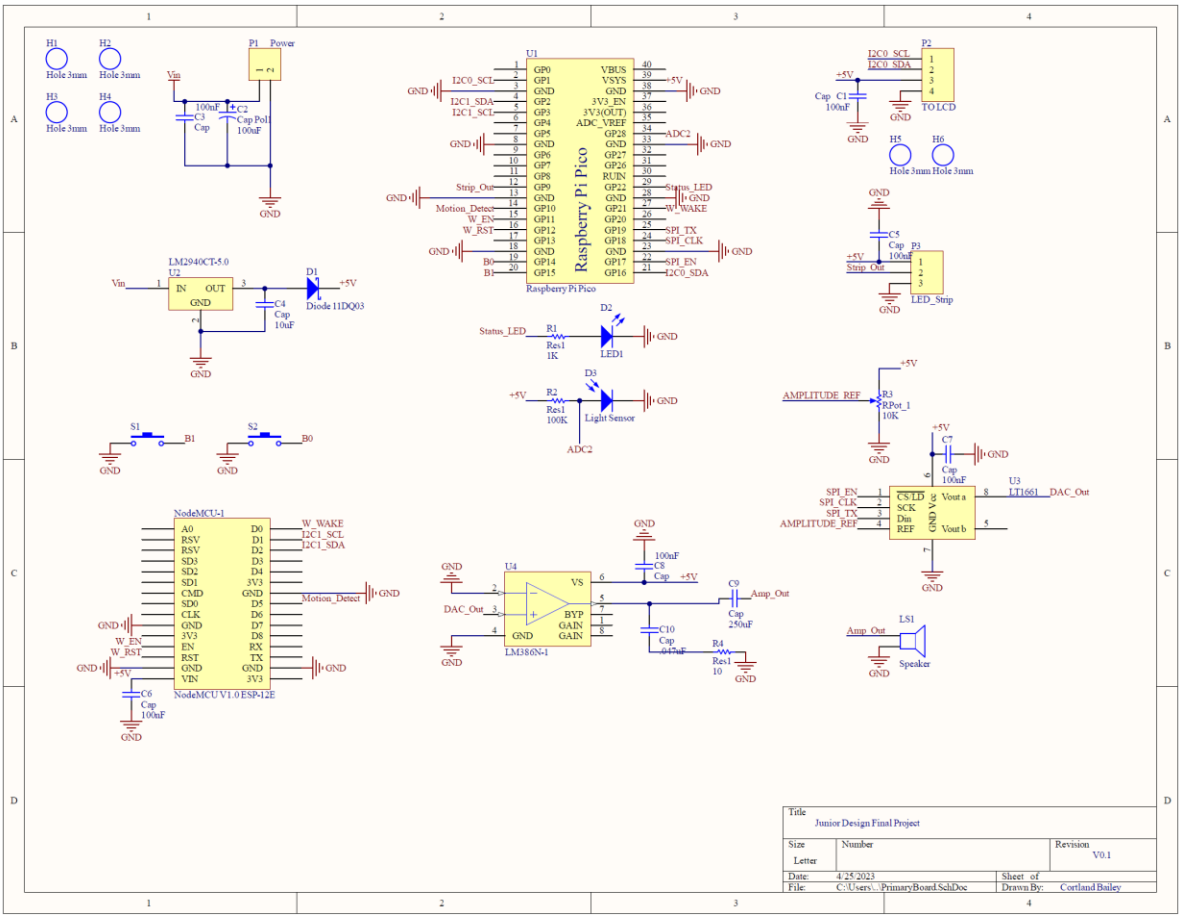


Figure 9 - Primary Circuit Board Schematic

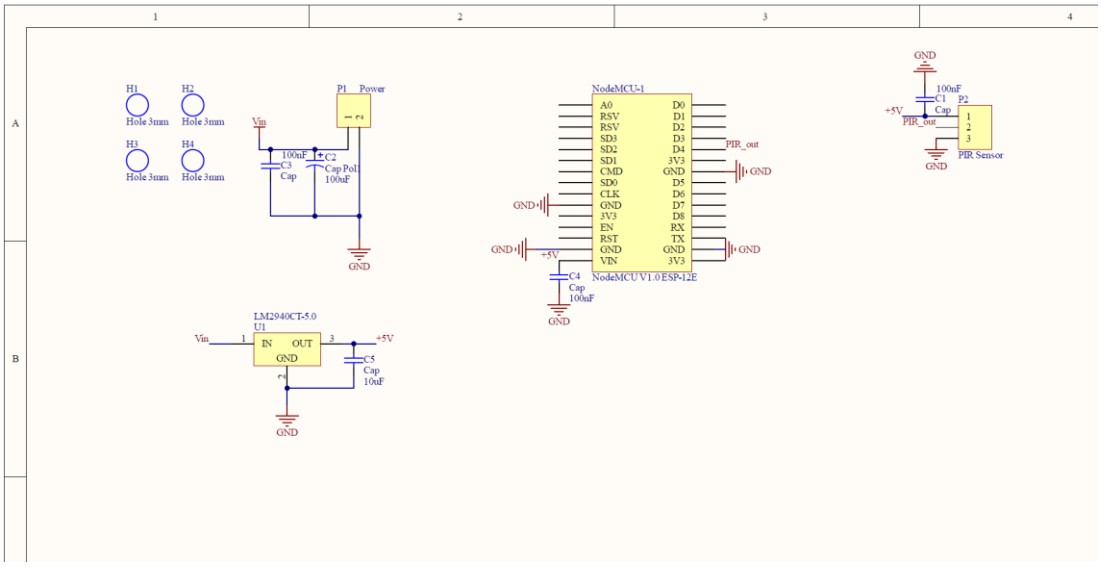


Figure 10 - Secondary Circuit Schematic

Altium PCB Design

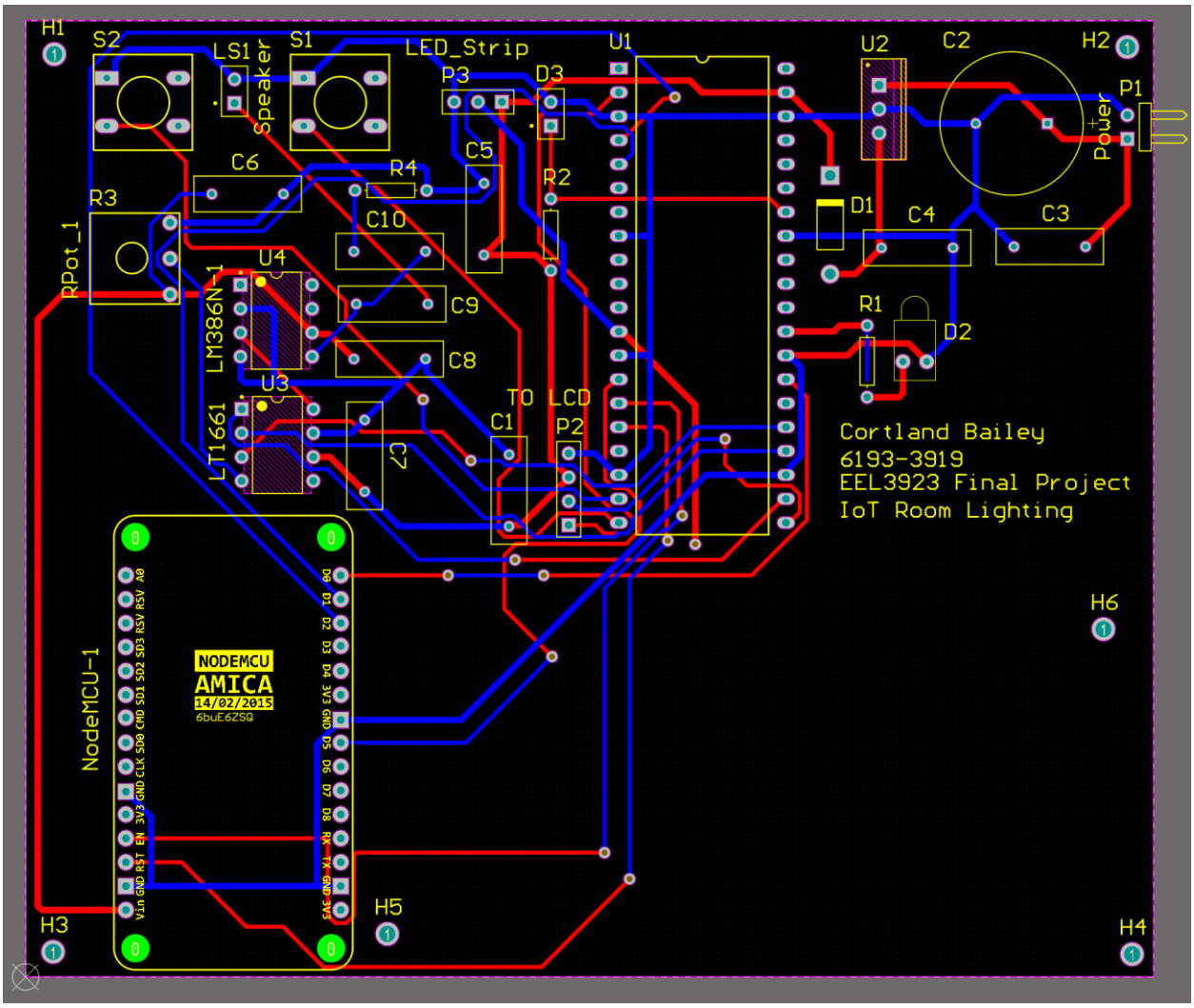
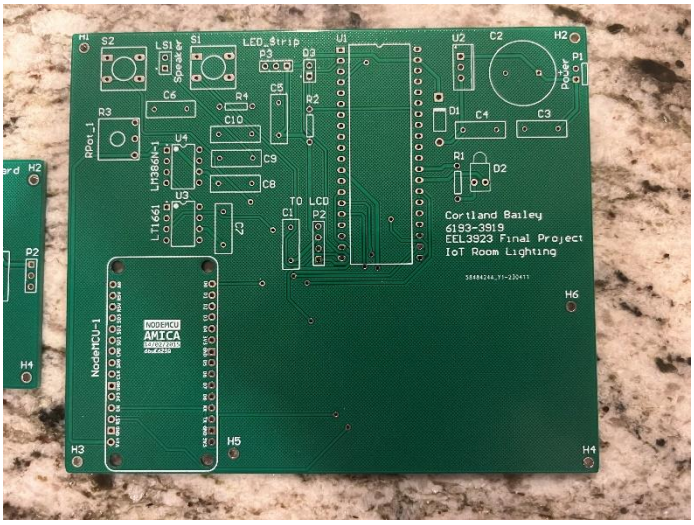


Figure 11 - Primary board PCB layout (With polygon pours shelved)



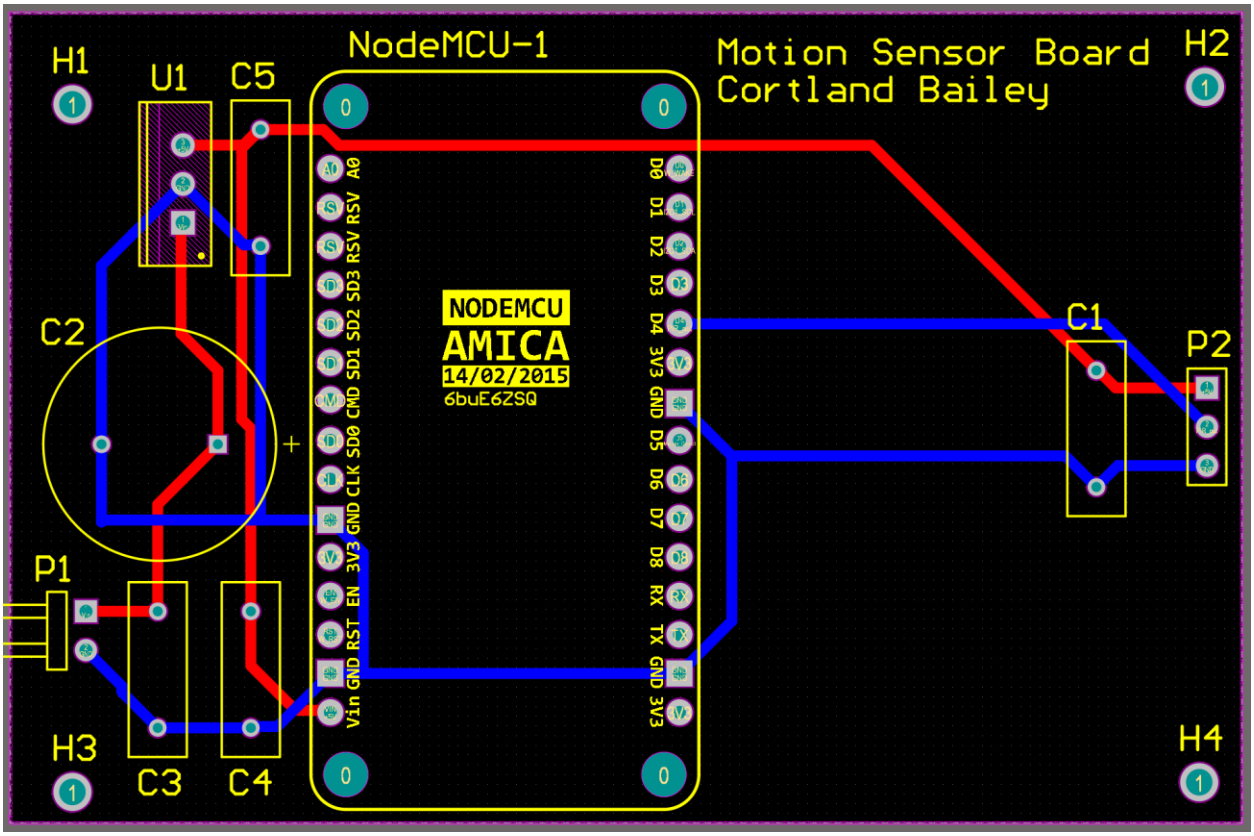
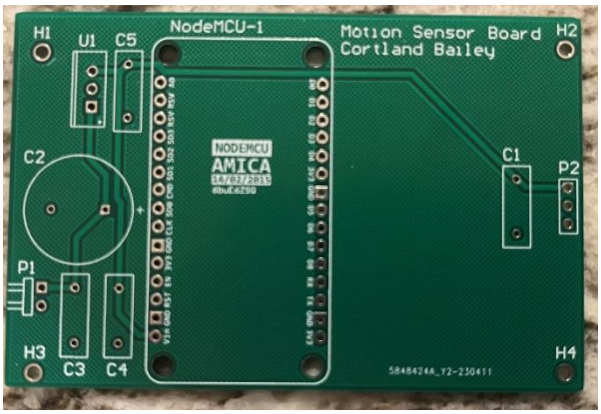


Figure 12 - Secondary board PCB design (with polygon pours shelved)



Bill of Materials

Item	Qty	Unit Cost	Ext. Cost	Source
Photocell	1	$1.15	$1.15	[6]
Raspberry Pi Pico	1	$4.00	$4.00	adafruit.com
NodeMCU ESP-12E	2	$5.49	$10.98	Amazon
SR501 PIR Sensor	1	$1.98	$1.98	temu.com
10k Resistor	1	$0.06	$0.06	[6]
1k Resistor	1	$0.06	$0.06	[6]
LCD	1	$4.95	$4.95	[6]
SPI Backpack for LCD	1	$2.95	$2.95	[6]
9V Leads	2	$0.45	$0.90	[6]
9V battery	2	$1.95	$3.90	[6]
LT1661	1	$6.21	$6.21	DigiKey
LM386N-1	1	$1.27	$1.27	DigiKey
LM2940CT-5.0	2	$1.85	$3.70	DigiKey
8Ohm speaker	1	$2.35	$2.35	[6]
100nF Ceramic Capacitor (pack of 10)	1	$1.35	$1.35	[6]
10uF Electrolytic capacitor	2	$0.17	$0.34	[6]
100uF Electrolytic Capacitor	2	$0.19	$0.38	[6]
Push Buttons	2	$0.20	$0.40	[6]
0.47uF Electrolytic Capacitor	2	$0.19	$0.38	[6]
10ohm Resistor	1	$0.06	$0.06	[6]
10k Potentiometer	1	$1.39	$1.39	[6]
Green LED (small)	1	$0.01	$0.01	[6]
Headers	1	$5.00	$5.00	Amazon
WS2812B Led Pixel Strip	1	$6.32	$6.32	BTF Lighting
Schottky diode - SD103B	1	$0.29	$0.29	[6]
PCB Cost	1	$41.59	$41.59	JLCPCB

Total Cost = $101.96

Results

Figure 13 - Final System

The outcome of my design functioned very well overall. The PCBs came with no flaws present. I encountered a few flaws while putting the parts together and testing the code. One of the 3 ESPs I ordered had a voltage regulation problem. It would not power up and the output power was only giving ~1V. After swapping out for my backup board the current one functioned and accepted the program correctly.

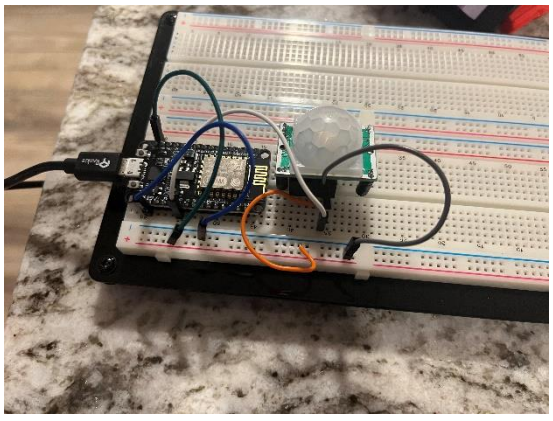


Figure 14 - Temporary motion sensor board setup

At this point I discovered another problem, the GPIO pin on the ESP board I selected was bouncing between 3.3V and 0V. I confirmed this wasn't a problem with the PCB by reconstructing the circuit on a breadboard and the same flaw existed. This told me that the problem was on the ESP itself so I changed which GPIO I was using in the program and left it on

the breadboard. Once this problem was solved the remainder was functioning correctly except the analog output.

My analog input had a DC offset and was a very messy signal. Upon further inspection of the datasheets for the LT1661 and LM386, I found that I had forgotten to include a coupling capacitor between these chips. This allowed a heavy DC component to go to the amplifier and it clips quite harshly due to not being able to operate at full-rail capacity. This would be rectified in a future version of the prototype.

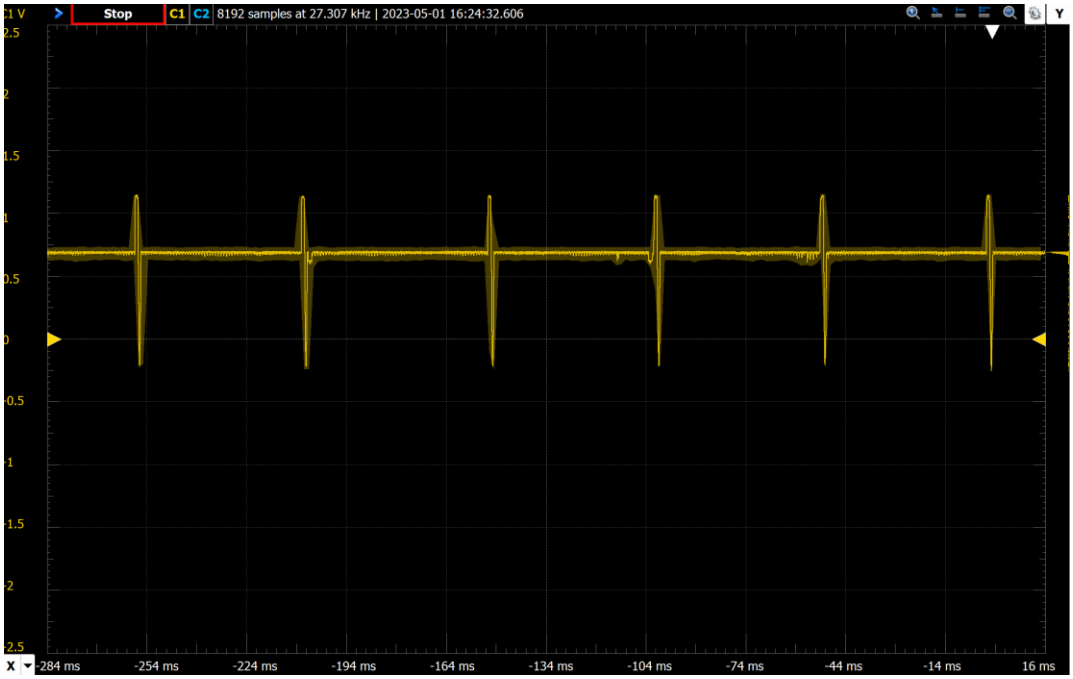


Figure 15 - Amplified analog output

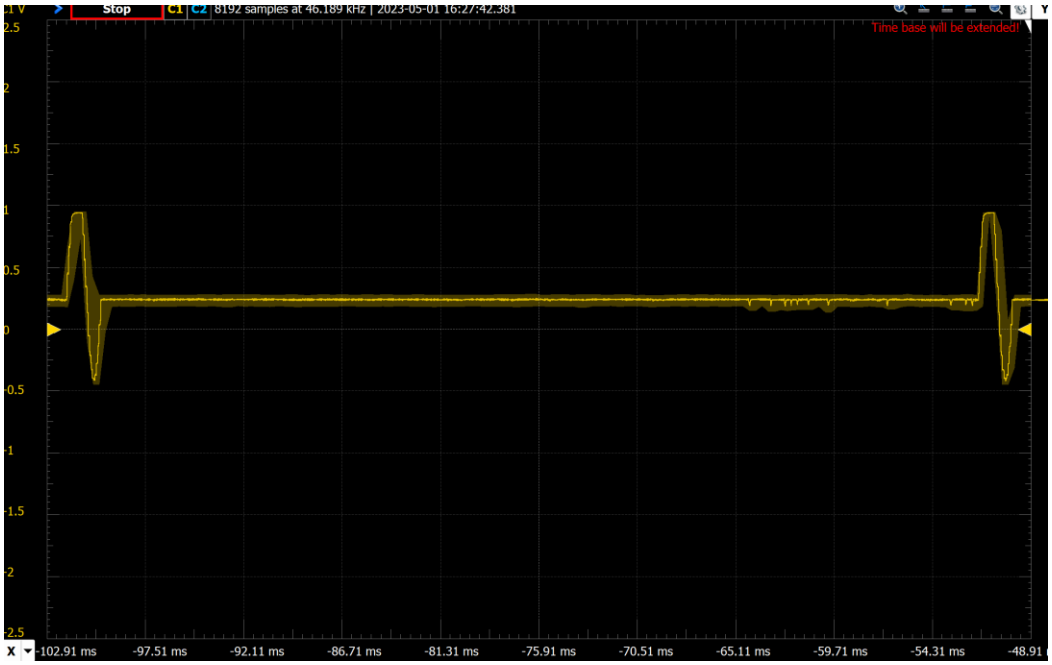


Figure 16 - Zoomed image of analog output

System Function

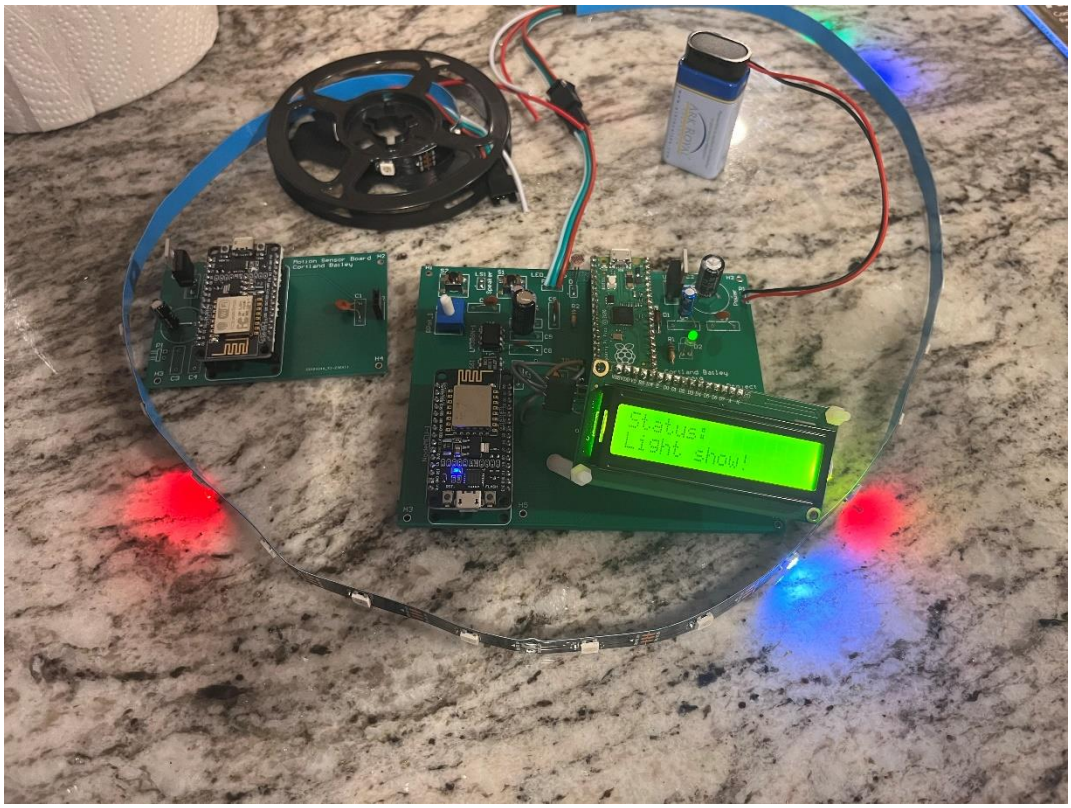


Figure 17 - Normal room light condition

The above picture shows the circuit in its “motion detected – medium light” condition.

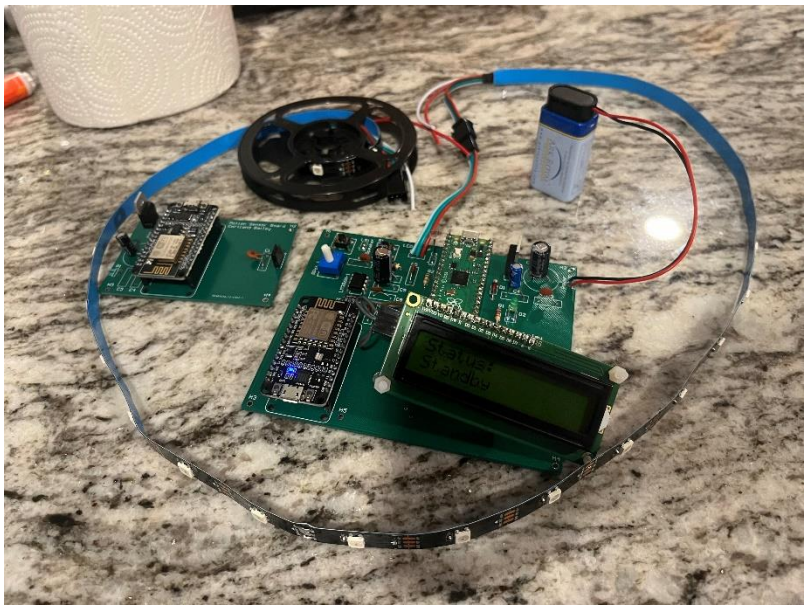


Figure 18 - System in standby

This picture shows both the LCD backlight off which is a function supported by a pushbutton on the circuit, while the system is in “No motion detected” mode, or Standby.

Figure 19 - Dark Room Function

Discussion/Conclusion

This was a challenging project. Given that this was a product I'd like to implement in my own house I was motivated to make it as close to fully functional and production ready quickly. One of the most challenging aspects was integrating 3 microcontrollers and the Analog signal.

Using multiple microcontrollers and getting them to function correctly is a big win for me. Having the issues with the ESPs I purchased was quite a problem to debug. This is a testament to build quality being a factor when purchasing development boards. I had 2 of 3 of my boards with a flaw. One of those two would be an easy workaround easily, but the other may be useless.

The analog output has been my nemesis since the first DAC module we did and this was no different. I completely neglected to include the coupling capacitor, despite learning about it earlier and it being the in datasheet. If version 2 is going to retain the speaker, it will likely get a higher sophistication speaker with a built in amplifier and the necessary capacitors to make the audio as clear as possible.

References

[1] "Raspberry Pi Documentation - Raspberry Pi Pico," *www.raspberrypi.com*. https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

[2] "HC-SR501 PIR Motion Sensor: Datasheet, Pinout and Specifications," *www.utmel.com*. https://www.utmel.com/components/hc-sr501-pir-motion-sensor-datasheet-pinout-and-specifications?id=696 (accessed May 02, 2023).

[3] "How HC-SR501 PIR Sensor Works & How To Interface It With Arduino," *Last Minute Engineers*, Jul. 03, 2018. https://lastminuteengineers.com/pir-sensor-arduino-tutorial/

[4] "Overview — MicroPython 1.19.1 documentation," *docs.micropython.org*. hfttps://docs.micropython.org/en/latest/index.html

[5] "Photo resistor | Resistor types | Resistor Guide," *eepower.com*. https://eepower.com/resistor-guide/resistor-types/photo-resistor/

[6] "Jameco Electronics - Electronic Components Distributor," *www.jameco.com*. https://www.jameco.com/

[7] "ESP8266 Sending Data Over Wi-Fi to another ESP8266 – Robot Zero One," *robotzero.one*, Apr. 26, 2018. https://robotzero.one/sending-data-esp8266-to-esp8266/ (accessed May 02, 2023).

[8] "NodeMCU DEVKIT V1.0," *GitHub*, Dec. 11, 2021. https://github.com/nodemcu/nodemcu-devkit-v1.0

[9] "Overview - NodeMCU Documentation," *nodemcu.readthedocs.io*. https://nodemcu.readthedocs.io/en/release/

[10] J. B. at C.-P. in C. S. / S. T. E. Researcher, "Basic ESP8266 NodeMCU tutorial: Breadboard, Pinout and Dimmable LED with Pulse-Width Modulation (PWM)," *Mechatronics Blog*, Feb. 13, 2019.

https://mechatronicsblog.com/basic-esp8266-nodemcu-tutorial-breadboard-pinout-and-dimmable-led-with-pulse-width-modulation-pwm/ (accessed May 02, 2023).
[11] B. Rolih, “pi_pico_neopixel,” *GitHub*, Apr. 18, 2023. https://github.com/blaz-r/pi_pico_neopixel (accessed May 02, 2023).
[12] “Raspberry Pi Pico Tutorial : NeoPixels / WS2812B LED’s,” *www.youtube.com*. https://www.youtube.com/watch?v=WpaXMcmwyeU (accessed May 02, 2023).
[13] Arduino, “Arduino Reference,” *Arduino.cc*, 2019. https://www.arduino.cc/reference/en/ p-themes, “WS2812B Led Pixel Strip Individually Addressable 30/60/74/96/100/144 pixels/leds/m 5V,” *BTF-LIGHTING*. https://www.btf-lighting.com/products/ws2812b-led-pixel-strip-30-60-74-96-100-144-pixels-leds-m (accessed May 02, 2023).
[14] L. P. last updated, “How to Use an I2C LCD Display With Raspberry Pi Pico,” *Tom’s Harware*, Jun. 27, 2021. https://www.tomshardware.com/how-to/lcd-display-raspberry-pi-pico (accessed May 02, 2023).
[15] Gareth Halfacree and B. Everard, *Get started with MicroPython on Raspberry Pi Pico*. Cambridge [Uk] Raspberry Pi Press, 2021.
[16] Texas_Instruments, “1A Low Dropout Regulator”, LM2940CT-5.0_NOPB datasheet, 2006
[17] Linear Technology, “Micropower Dual 10-bit DAC in MSOP,” LTC1661 datasheet, 1999.
[18] “LM386,” Ti.com, 2017. https://www.ti.com/product/LM386
[19] Cortland Bailey, “Amplifier Module Report”, 2023

Appendix

Pico Code:

```
from machine import I2C, ADC, Pin, SPI, Timer
import time
from pico_i2c_lcd import I2cLcd
from neopixel import Neopixel
import random


spi = SPI(0, baudrate=20000000,polarity = 0, phase = 0, bits = 8, sck = Pin(18), mosi
= Pin(19))
cs = Pin(17, mode=Pin.OUT, value=1)
HB = 0
LB = 0
frequency = 44000
sin_wave =
[512,639,758,862,943,998,1022,1014,974,906,812,700,576,447,323,211,117,49,9,1,25,80,1
61,265,384]
def generate_sine_wave(timer):
    global sin_wave, spi, cs
```

```
    w = int(1 / frequency * 100000)
    for i in range(25):
        LB = (sin_wave[i] << 2)
        HB = (144 | (sin_wave[i] >> 6))
        txdata = bytearray([HB, LB])
        cs(0)
        spi.write(txdata)
        cs(1)
        time.sleep_us(w)

# Create a Timer object and start the timer
sine_wave_timer = Timer()
sine_wave_timer.init(period=10, mode=Timer.PERIODIC, callback=generate_sine_wave)

#button setup
button0 = Pin(14, Pin.IN, Pin.PULL_UP)
button1 = Pin(15, Pin.IN, Pin.PULL_UP)
debounce_time0 = 0
interrupt_flag0 = 0
debounce_time1 = 0
interrupt_flag1 = 0

#Light sensor
light=ADC(machine.Pin(28))
conv_factor = 5 / 65535


#LCD Setup
i2c=I2C(0,scl=Pin(1),sda=Pin(16),freq=400000)
I2C_ADDR=i2c.scan()[0]
lcd = I2cLcd(i2c, I2C_ADDR, 2, 16)
lcd.clear()

#Wifi board connections
Motion_detect=machine.Pin(10, Pin.IN)
W_EN=machine.Pin(11)
W_RST=machine.Pin(12)
W_WAKE=machine.Pin(21)
LED_Out=machine.Pin(22, Pin.OUT)
LED_Out.value(0)

# LED Strip setup
numpix = 20
strip = Neopixel(numpix, 0, 9, "RGB")
white = (255, 255, 255)
red = (255, 0, 0)
orange = (255, 50, 0)
yellow = (255, 100, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
indigo = (100, 0, 90)
violet = (200, 0, 100)
colors_rgb = [white, red, orange, yellow, green, blue, indigo, violet]
delay = 300
strip.brightness(42)
```

```
blank = (0,0,0)



#Button interrupt routine for debouncing:
def callback0(button0):
    global interrupt_flag0, debounce_time0
    if (time.ticks_ms()-debounce_time0)>250:
        debounce_time0=time.ticks_ms()
        if interrupt_flag0==1:
            sine_wave_timer.init(period=10, mode=Timer.PERIODIC,
callback=generate_sine_wave)
        if interrupt_flag0==0:
            sine_wave_timer.deinit()
        interrupt_flag0 = not interrupt_flag0

def callback1(button1):
    global interrupt_flag1, debounce_time1
    if (time.ticks_ms()-debounce_time1)>250:
        debounce_time1=time.ticks_ms()
        if interrupt_flag1==1:
            lcd.backlight_off()
        elif interrupt_flag1==0:
            lcd.backlight_on()
        interrupt_flag1= not interrupt_flag1

button0.irq(trigger=Pin.IRQ_FALLING, handler=callback0)
button1.irq(trigger=Pin.IRQ_FALLING, handler=callback1)
status = "Startup"


while True:
    w = int(  (1/frequency)*100000  )
    strip.clear()
    room_light=0.0
    old_status = status
    for i in range(100):
        room_light = room_light+light.read_u16() * conv_factor


    if Motion_detect.value()==1:
        LED_Out.value(1)


        if room_light > 100:
            strip.set_pixel(0, colors_rgb[0])
            strip.set_pixel(1, colors_rgb[0])
            strip.set_pixel(2, colors_rgb[0])
            strip.set_pixel(3, colors_rgb[0])
            strip.set_pixel(4, colors_rgb[0])
            strip.show()
            frequency = 10000
            status = "LED Active"

        elif room_light < 30:
```

```
            strip.clear()
            strip.show()
            sine_wave_timer.init(period=1, mode=Timer.PERIODIC,
callback=generate_sine_wave)
            status = "Music Only"
        else:
            strip.set_pixel(random.randint(0, numpix-1), colors_rgb[random.randint(0,
len(colors_rgb)-1)])
            strip.set_pixel(random.randint(0, numpix-1), colors_rgb[random.randint(0,
len(colors_rgb)-1)])
            strip.set_pixel(random.randint(0, numpix-1), colors_rgb[random.randint(0,
len(colors_rgb)-1)])
            strip.set_pixel(random.randint(0, numpix-1), colors_rgb[random.randint(0,
len(colors_rgb)-1)])
            strip.set_pixel(random.randint(0, numpix-1), colors_rgb[random.randint(0,
len(colors_rgb)-1)])
            strip.show()
            time.sleep_ms(delay)
            strip.fill((0,0,0))
            frequency = 100000
            status = "Light show!"
            sine_wave_timer.init(period=50, mode=Timer.PERIODIC,
callback=generate_sine_wave)


    else:
        LED_Out.value(0)
        strip.clear()
        strip.show()
        status = "Standby"

    if old_status != status:
        lcd.clear()
        lcd.putstr("Status: \n")
        lcd.putstr(status)
        time.sleep_ms(200)
```

Wifi Access Point code:

```
/*
   Copyright (c) 2015, Majenko Technologies
   All rights reserved.

   Redistribution and use in source and binary forms, with or without modification,
   are permitted provided that the following conditions are met:

 * * Redistributions of source code must retain the above copyright notice, this
     list of conditions and the following disclaimer.

 * * Redistributions in binary form must reproduce the above copyright notice, this
```

```
     list of conditions and the following disclaimer in the documentation and/or
     other materials provided with the distribution.
*/

/* Create a WiFi access point and provide a web server on it. */

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

#ifndef APSSID
#define APSSID "ESP_ap"
#define APPSK "thereisnospoon"
#endif

/* Set these to your desired credentials. */
const char *ssid = APSSID;
const char *password = APPSK;

ESP8266WebServer server(80);

/* Just a little test message.  Go to http://192.168.4.1 in a web browser
   connected to this access point to see it.
*/
void handleSentVar() {
  if (server.hasArg("sensor_reading")) {
    int readingInt = server.arg("sensor_reading").toInt();
    if(readingInt==HIGH) {
      digitalWrite(D5, HIGH);
    }
    else digitalWrite(D5, LOW);
    Serial.println(readingInt);
    server.send(200, "text/html?", "Data Received");
  }
}

void setup() {
  delay(1000);
  Serial.begin(115200);
  /* You can remove the password parameter if you want the AP to be open. */
  WiFi.softAP(ssid, password);
  pinMode(D5, OUTPUT);
  server.on("/data/", HTTP_GET, handleSentVar);
  server.begin();
}

void loop() {
  server.handleClient();
}
```

Wifi Client Code:

```
#include <ESP8266WiFi.h>
```

```
const char *ssid = "ESP_ap";
const char *password = "thereisnospoon";
int sensorValue = 13;        // value read from the PIR
int outputValue;        //value to send to the server

void setup() {
  Serial.begin(115200);
  delay(10);

  // Explicitly set the ESP8266 to be a WiFi-client
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  pinMode(sensorValue, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);

}



void loop() {
  long val = digitalRead(sensorValue);
  // builtin led to debug motion sensor, activates when motion sensor sends signal,
deactivates when motion sensor is off
  if (val == HIGH) {
    digitalWrite(LED_BUILTIN, LOW);
    Serial.println("Motion Detected!");
    Serial.println(sensorValue);
    outputValue = 1;
  }
  else if (val == LOW) {
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("Motion not detected :(");
    Serial.println(sensorValue);
    outputValue = 0;
  }

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const char * host = "192.168.4.1";
  const int httpPort = 80;

  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }


  // We now create a URI for the request. Something like /data/?sensor_reading=123
  String url = "/data/";
```

```
  url += "?sensor_reading=";
  url += outputValue;

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");


//  unsigned long timeout = millis();
//  while (client.available() == 0) {
//    if (millis() - timeout > 5000) {
//      Serial.println(">>> Client Timeout !");
//      client.stop();
//      return;
//    }
//  }

  delay(500);
}
```