HW2 EE333 Digital Design

SECTION #2

Group 1

MEMBERS:

Raj Singh

Jordan McGhee

SUBMISSION DATE:

12/12/2022

Table of Contents

Introduction

We are building a custom Arduino PCB that is designed to scan for WiFi signals and display their strength for our project. The board uses the Atmega328p microcontroller for running code, the ESP8266 WiFi module to scan for signals, and the WC1602A LCD screen to display the results. The signal strength is also indicated by a strip of 10 WS2812B LED, which change color to show the strength of the signal. The PCB is powered by a USB C port, which provides the 5V rail needed to power the components.

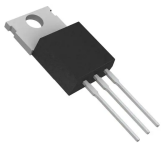
This project is a great way to explore the capabilities of the ESP8266 WiFi module and the Atmega328p microcontroller. By using the LCD screen and LED, we can easily monitor the signal strength of nearby networks and use this information to optimize our own WiFi setup. Additionally, the custom PCB allows us to easily integrate this functionality into our own projects and create internet-connected devices that can scan for WiFi signals. The Atmega328p microcontroller provides the processing power needed to run the WiFi scan and display the results.

Design Choices

1. **Power Supply**

The USB C port is used to provide power to the Arduino PCB because it is a convenient and widely available power source. Most computers and many other devices have a USB C port, which makes it easy to power the board using a standard USB cable.

Additionally, the USB C port provides a stable and reliable source of power for all of the other components in the circuit. The generic USB standard provides a 5v output, which is more than enough to power the components on our Arduino PCB as long as we regulate the voltage down to 3.3v for the WiFi module and the LCD. This ensures that the circuit will have a consistent and reliable power supply, which is important for its proper operation. Overall, using the USB C port as the power source for our Arduino PCB is a convenient and reliable choice that allows us to easily power the board using a standard USB cable.

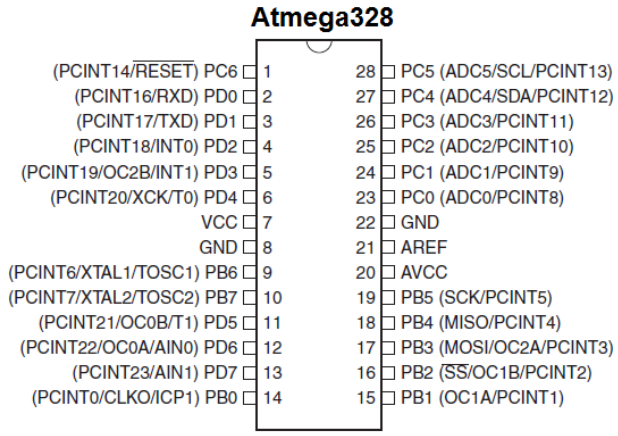


2. Atmega Pin Configuration

The Arduino PCB we are building uses a number of different pins that are connected to the various components. In our project, the following pins are used:

- The Atmega328p microcontroller is connected to the ESP8266 WiFi module using the RX and TX pins wich correspond to pins 0 and 1 on the Atmega. These pins are used for communication between the microcontroller and the WiFi module, while also providing support for serial interrupts, making reading and writing to the WiFi module more reliable than the other pins.
- The Atmega 328p microcontroller is connected to the WC1602A LCD screen using a set of digital pins. These pins are used to send data and commands to the LCD screen. In order for full functionality to work, we had to use a total of 6 digital pins, 4 of which were used for data.
- The WS2812B LED is connected to the Atmega328p microcontroller using a single digital pin. This pin is used to send data to the LED, which is used to control its color and brightness. One of the benefits of the WS2812B, is that it allows us to individually address each of the leds, while only requiring one pin to control.
- The USB C port is connected to the Atmega328p microcontroller using the VIN and GND pins. The VIN pin is used to provide power to the board, while the GND pin is used as a reference point for the other components. The VIN pin directly corresponds to our 5v rail, which later gets stepped down to 3.3V to generate our 3.3V rail.

Overall, the pin configuration for our project is designed to enable the various components to communicate with each other and work together to scan for WiFi signals and display the results. The Atmega328p microcontroller serves as the brains of the operation, while the ESP8266 WiFi module and the other components provide the input and output capabilities.



3. **LCD Screen**

The WC1602A LCD screen is a type of liquid crystal display (LCD) that is commonly used in Arduino projects. LCDs are a type of display technology that uses liquid crystals to produce images. They are known for their low power consumption, high resolution, and wide viewing angle.

In our project, the WC1602A LCD screen is used to display the results of the WiFi scan performed by the ESP8266 WiFi module. The screen is connected to the Atmega328p microcontroller using a set of digital pins, which are used to send data and commands to the LCD. This allows the microcontroller to control the display and show the signal strength of nearby WiFi networks.

The WC1602A LCD screen has a resolution of 128x64 pixels, which allows it to display detailed information. It also has a backlight, which makes it easy to read in a variety of lighting conditions. Overall, the WC1602A LCD screen is a useful and versatile component that allows us to display information for our Arduino project. This screen was imperative for debugging as we didn't have any serial connections to the Atmega once it was on the board. So this LCD screen served as a boon for debugging our system as well as allowing us to have better insight on how things like initialization or failure conditions occurred.

4. **LED**

The WS2812B LED is a type of light-emitting diode (LED) that is commonly used in Arduino projects. LEDs are a type of semiconductor device that emits light when a current is applied to them. They are known for their low power consumption, high brightness, and long lifetime.

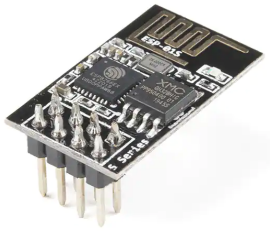
In our project, the WS2812B LED is used to indicate the strength of the WiFi signal being scanned by the ESP8266 WiFi module. The LED is connected to the Atmega328p microcontroller using a digital pin, which is used to send data to the LED. This allows the microcontroller to control the color and brightness of the LED,

which is used to show the strength of the WiFi signal.

The WS2812B LED has the ability to display a wide range of colors, which makes it useful for indicating different signal strengths. It also has a high brightness, which makes it easy to see even in bright lighting conditions. Overall, the WS2812B LED is a useful and versatile component that allows us to visually indicate information on our Arduino project. This particular LED also had a very good software library that allowed us to individually control the color of LED's at arbitrary locations within the completed strip. Using this library we were able to create a gradient of colors to show the relative signal strength even in conditions when the LCD screen may not be fully visible. One of the concerns we had with this LED was with the use of decoupling capacitors to reduce current spikes when we toggled them. On our board we had left space for each LED to have it's own decoupling capacitor, but when it came to soldering, we noticed that the pin pads were a little too close and we ended up not soldering them on since we didn't want to risk ruining board, and since we had a very low switching frequency (< 1 Hz) it wasn't vital to our boards operation.

5. Wifi Module

For our project we chose to use the ESP8266 to gather our wifi information. Since we did not have the ability to program the microcontroller on the ESP8266, we relied on the pre-packaged firmware that comes with every ESP8266 module. This module communicates over serial using what they called "AT" commands. These "AT" commands are string of information that enable anyone to send commands or configuration information to the board without having to adjust the wifi module in any way. This allowed us to easily connect our projects to the internet, which can be useful for a wide range of applications.



Some of the reasons we chose to use the Arduino ESP8266 include:

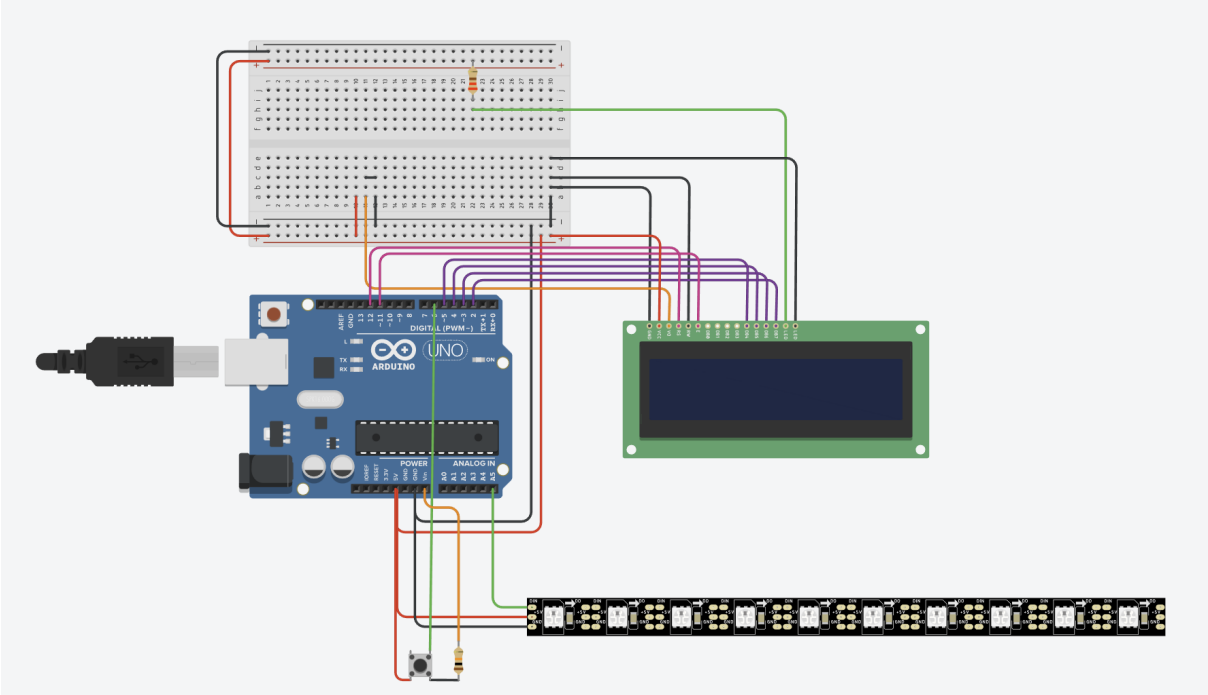
- Easy internet connectivity: The ESP8266 WiFi chip allows the Arduino ESP8266 to connect to the internet, which can be useful for a wide range of projects.

- Versatility: The Arduino ESP8266 is based on the popular Arduino platform, which means that it is compatible with a wide range of sensors and other components. This makes it easy to use the board for a variety of different projects.
- Large community: The Arduino platform has a large community of users and developers, which means that there is a wealth of information and support available for the Arduino ESP8266.
- Affordable: The Arduino ESP8266 is an affordable option for those who want to experiment with internet-connected projects without breaking the bank.

Overall, the Arduino ESP8266 is a versatile and affordable option for those who want to build internet-connected projects.

Simulation

Tinkercad is a online platform that allows us to design and simulate electronic circuits. To simulate our Arduino PCB in Tinkercad, we would first need to create a digital model of the circuit using the Tinkercad tools. This would involve adding the various components of our circuit, such as an arduino UNO (to simulate the Atmega328p microcontroller), the WC1602A LCD screen, the NeoPixel strip (to simulate the WS2812B LEDs).



Once the components are added to the design, we need to connect them using Tinkercad's virtual wires. This would involve connecting the pins of the various components together, as well as connecting power and ground.

Once the circuit is assembled in Tinkercad, we can use the platform's simulation tools to test the circuit and see how it behaves. In order to verify the functionality of our design, we tested with using real data that we gathered in lab. During our lab time, we were able to get raw information from the board by using an arduino uno, connecting the reset pin HIGH and then opening up the serial monitor in the arduino IDE. By forcing the arduino to constantly reset, we were able to send AT commands directly to the ESP8266 without the arduino interfering, this was important because were unable to get information using the arduino itself due to the high default baud rate of the ESP8266 and the low maximum baudrate from using software serial on an arduino. After we had gotten information from the ESP8266, we put that into a file to be read from when did our tinkercad simulations.

Overall, using Tinkercad to simulate our Arduino PCB allows us to quickly and easily test our circuit design and identify any potential issues before building the physical circuit. This can save time and effort and help ensure our circuit functions properly.

Code Design

To design the code for our Arduino project, we would need to use the Arduino Integrated Development Environment (IDE). The Arduino IDE is a software tool that allows us to write, compile, and upload code to our Arduino board.

In our project, the code would need to perform the following tasks:

- Initialize the various components of our circuit, such as the Atmega328p microcontroller, the ESP8266 WiFi module, the WC1602A LCD screen, and the WS2812B LED.
- Scan for nearby WiFi signals using the ESP8266 WiFi module and retrieve their signal strength.
- Display the signal strength of the WiFi signals on the WC1602A LCD screen and the WS2812B LED.

To accomplish these tasks, we would need to use the appropriate Arduino libraries and functions. The Arduino libraries are pre-written code that provides a set of functions that can be used to control and interact with various components and devices.

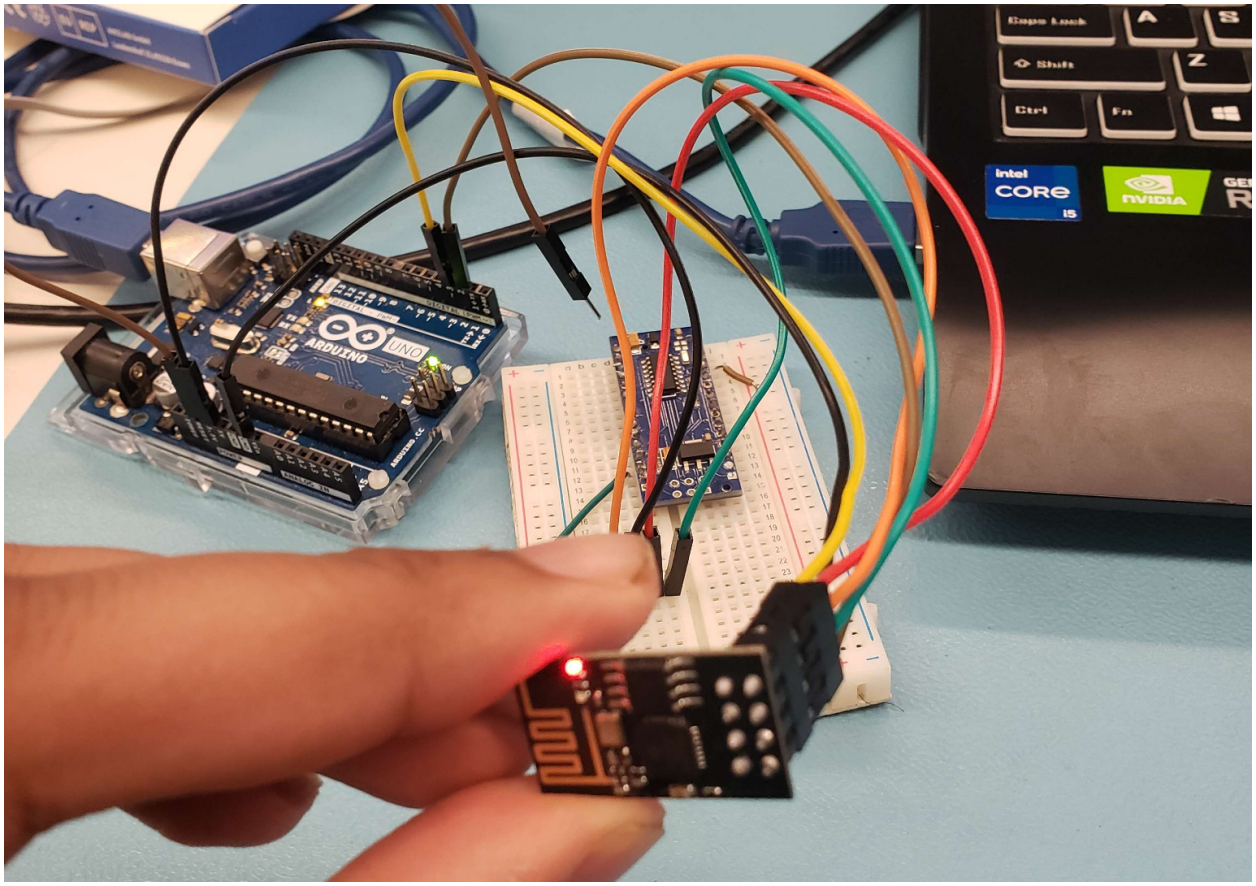
For example, to initialize and control the ESP8266 WiFi module, we would need to use and understand the various AT commands provided by the ESP8266 manufactuerers. This command specification provides a set of functions that allow us to reset the device, scan for WiFi signals, connect to a network, and retrieve information about signal strength.

To initialize and control the WC1602A LCD screen, we would need to use the Arduino LiquidCrystal library. This library provides a set of functions that allow us to send data and commands to the LCD screen and control its display.

Overall, the code design for our project would involve using the appropriate Arduino libraries and functions to control the various components of our circuit and perform the desired tasks. Our finalized code can be found in the appendix.

Breadboard Testing

Breadboard testing is a common method for prototyping and testing electronic circuits. It allows us to quickly and easily build a circuit and test it without having to commit to a permanent design. Overall we were only able to verify two of the main components of the design using breadboading since a lot of our other pieces were Surface Mount Devices (SMD) which were difficult to connect to a breadboard without proper tools. Of the components we had, we verified the 5v DC to 3.3V DC regulator as well as the ESP8266. The ESP8266 was able to verified by skipping over the arduino by resetting it, and using the serial monitor in the arduino IDE in order to send/recieve commands directly from the host computer to the ESP8266.



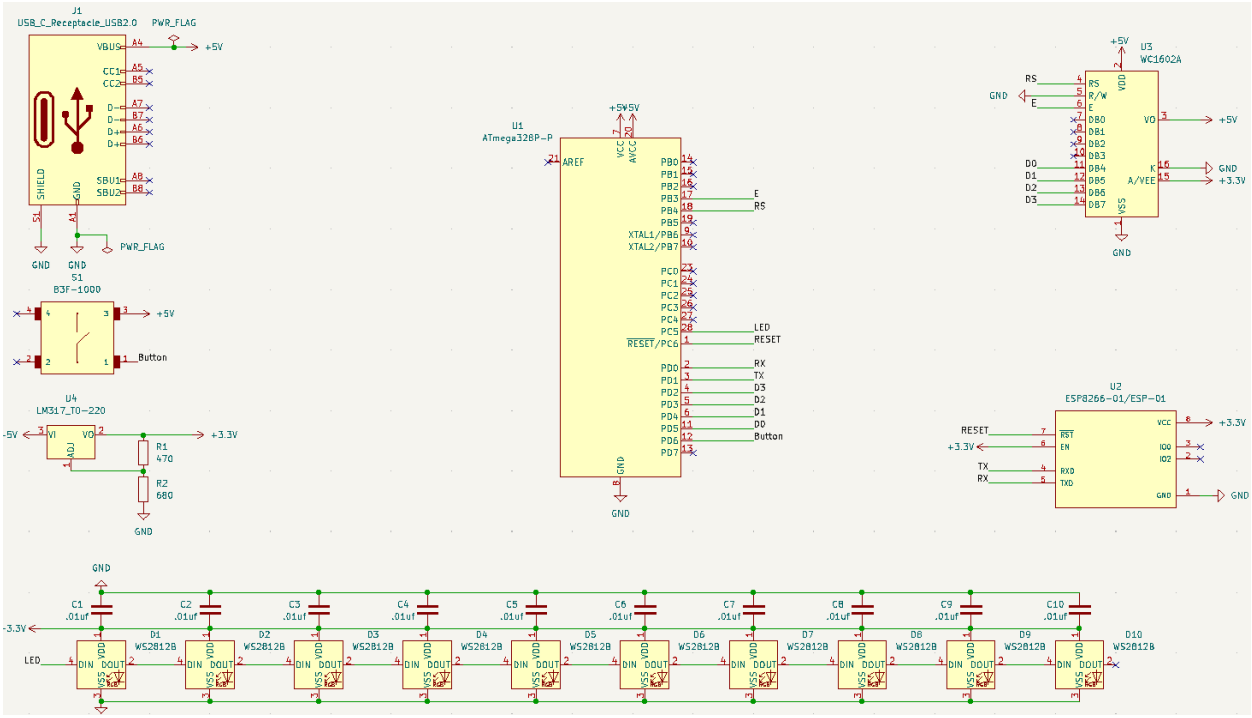
KiCad Design

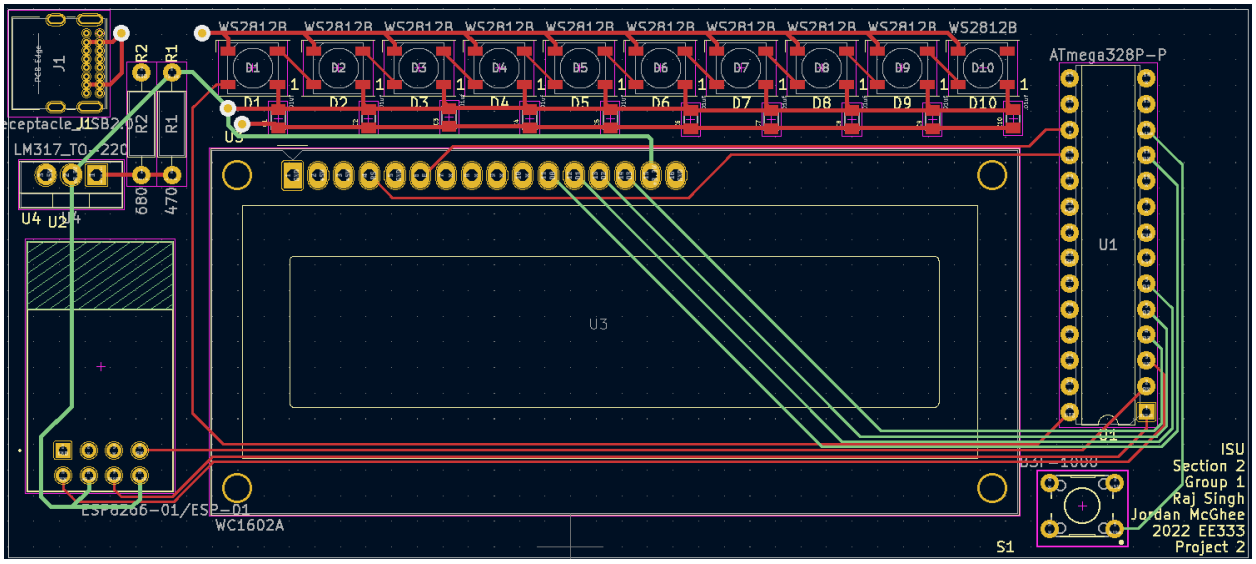
The KiCad design for our project includes a PCB with a layout that is optimized for the components we are using. The PCB accommodates the Atmega328p microcontroller, the ESP8266 WiFi module, the WC1602A LCD screen, the WS2812B LED, and the USB C port.

The Atmega328p microcontroller and the ESP8266 WiFi module are placed on opposite sides of the PCB, with the ESP8266 WiFi module connected to the microcontroller using the RX and TX pins. The LCD screen is also connected to the microcontroller using a set of digital pins, while the LED is connected to a separate digital pin. The USB C port is connected to the microcontroller using the VIN and GND pins.

The KiCad design also includes the necessary connections and traces to enable the components to communicate with each other and function properly. The layout of the PCB is optimized for efficiency and ease of use, making it easy to assemble the board and get our project up and running. Overall, the

KiCad design for our project provides a solid foundation for building a WiFi scanning device. Below is a picture of the finalized layout.





Final Outcome

Overall our design worked splendidly. We were able to incorporate all of the pieces that we had intended, and even added smaller quality of life changes like showing an initialization screen, and creating a color gradient to improve our design by that little bit more. All of the issues that we had with

either our pcb or our code were able to be resolved without visually affecting design or its functionality, so we believe that to be a cause for celebration.

Notable Issues

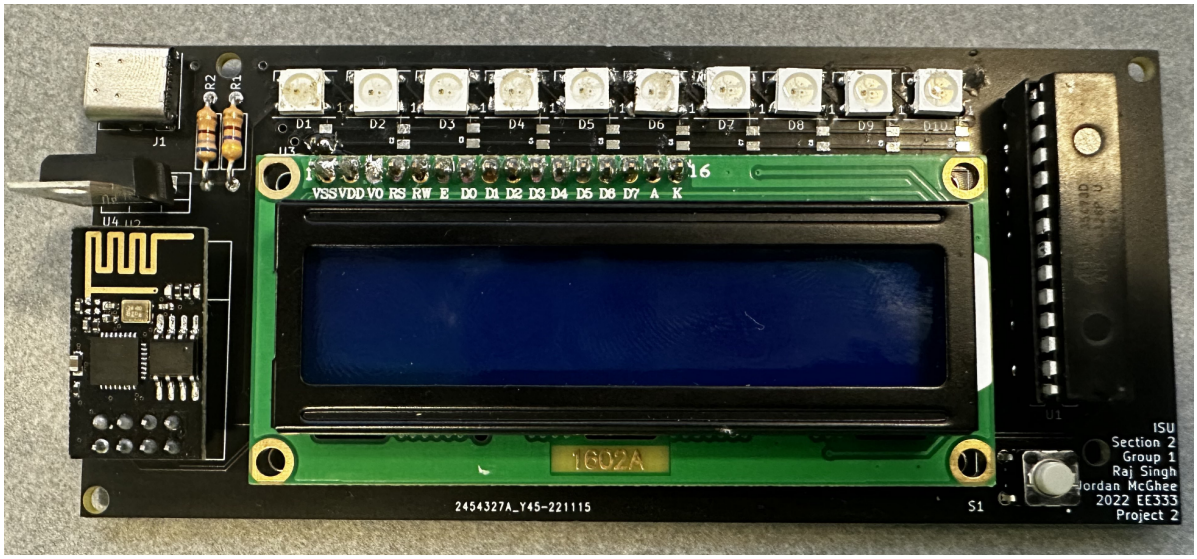
There were quite a few issues that occurred over the course of this project. The most notable issues we had were with the baud rate on the ESP8266, contrast pin mapping on the LCD, soldering the, and the button connections. The first issue occurred when we were trying to test out our ESP8266 on the bread board. We wanted to be able to see the output of ESP8266 directly, however using the software serial library to communicate with the ESP8266 proved to show a lot of noise, and in a lot of cases unreadable output. We deduced that the issue was the fact that hardware serial had interrupts to ensure fluent communication, but in the software serial library they only had polling, which was less reliable. We were able to verify that the ESP8266 worked by connecting the serial connectors directly to computer, wich proved to be much more reliable. The next issue we faced had to deal with the contrast pin being driven high in our initial design. If the contrast pin is set to 5V, then that means the output will have 0 contrast, making our image unreadable, but we didn't realize this until after the PCB was delivered to us, so in order to remedy this, we had to solder a small wire from the contrast pin on the led, to an exposed ground connection, which luckily there was one immediately next to it. Finally, our most critical oversight was with the button connections. In our initial design we had forgotten to create a path for the button output to drain from, this caused a floating output when we tried to read wether or not the button was pressed, giving us non-sensical results. In order to fix this, we had soldered an extra 10k resistor from the pin we read the button from to ground, which worked perfectly.

Summary

In this project, you have created a custom Arduino PCB that is designed to scan for WiFi signals and display their strength. The board uses the Atmega328p microcontroller, the ESP8266 WiFi module, the WC1602A LCD screen, and the WS2812B LED to accomplish this task. The PCB is powered by a USB C port, which provides the 5V rail needed to power the components.

Through breadboard testing and KiCad design, we have developed a circuit that is capable of scanning for WiFi signals and displaying the results on the LCD screen and showing the equivalent signal on an LED strip. This project proved to be a great way to explore the capabilities of the ESP8266 WiFi module and the Atmega328p microcontroller, and resulted in a useful tool for monitoring the strength of WiFi signals.

Overall, this project has been a successful exploration of internet-connected Arduino projects and the use of various components to create a functional device. We were able to create a useful and versatile tool that allowed us to explore the capabilities of the ESP8266 WiFi module and the Atmega328p microcontroller, providing a practical way to monitor the strength of WiFi signals for optimizing our very own WiFi setup.



Appendix:

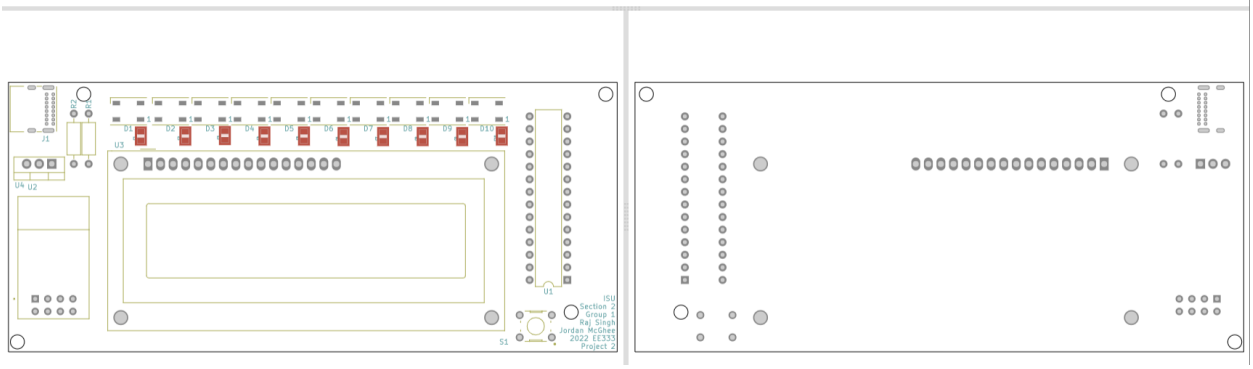
[Video Link]

BOM:

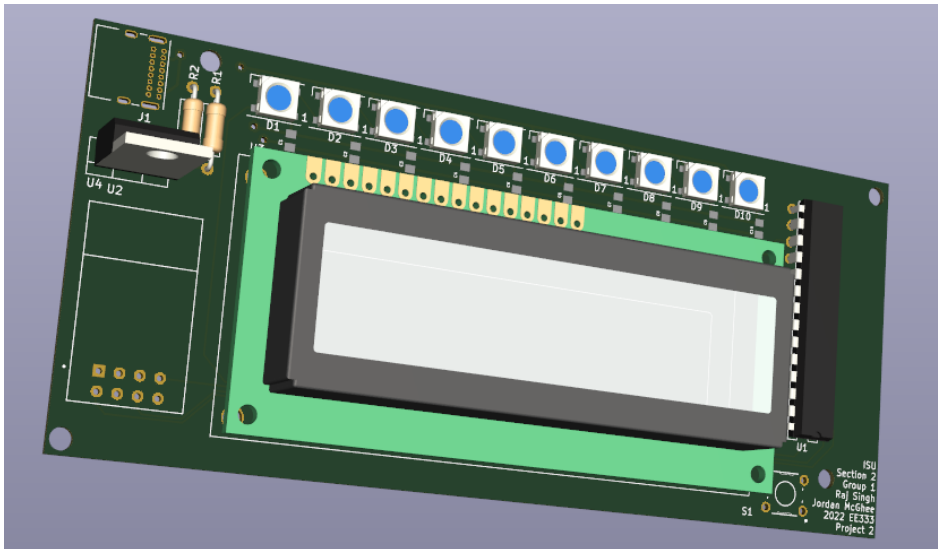
Project2 **Rev:** 2022-11-14 15:16:33

Ref lookup | Filter

	Sourced	Placed	Value	References	Footprint	Quantity
1	☐	☐	.01uf	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10	CAPC2012X140N	10
2	☐	☐	470	R1	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal	1
3	☐	☐	680	R2	R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal	1
4	☐	☐	WS2812B	D1, D2, D3, D4, D5, D6, D7, D8, D9, D10	LED_WS2812B_PLCC4_5.0x5.0mm_P3.2mm	10
5	☐	☐	ATmega328P	U1	DIP-28_W7.62mm	1
6	☐	☐	ESP8266	U2	XCVR_ESP8266-01_ESP-01	1
7	☐	☐	WC1602A	U3	WC1602A	1
8	☐	☐	LM317	U4	TO-220-3_Vertical	1
9	☐	☐	B3F-1000	S1	SW_B3F-1000	1
10	☐	☐	USB C	J1	USB_C_Receptacle_GCT_USB4085	1



3D View



TinkerCad Link

Code:

```
#include <LiquidCrystal.h>
#include <SoftwareSerial.h>
#include <Adafruit_NeoPixel.h>

// initialize the LCD
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// Specify the number of LEDs on the strip
const int max_Leds = 10;

// Specify the pin that is connected to the data pin of the LED strip
const int ledStripPin = A5;

Adafruit_NeoPixel strip = Adafruit_NeoPixel(max_Leds, ledStripPin, NEO_GRB
+ NEO_KHZ800);

void setup() {
  // set up the LCD's number of columns and rows
  lcd.begin(16, 2);

  pinMode(6, INPUT);

  // set up the SoftwareSerial port
  Serial.begin(9600);

  // Initialize the NeoPixel library
  strip.begin();

  // initialize the ESP8266
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Initializing");
  lcd.setCursor(0, 1);
  lcd.print("ESP8266...");
  delay(1000);
  lcd.clear();
  lcd.setCursor(0, 0);
```

```
  lcd.print("Resetting ESP8266");
  Serial.println("AT+RST");
  delay(1000);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Setting CWMODE");
  Serial.println("AT+CWMODE=1");
  delay(1000);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Setting CIFSR");
  Serial.println("AT+CIFSR");
  delay(1000);
}

void loop() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Scanning");
  lcd.setCursor(0, 1);
  lcd.print("Networks");
  // scan for networks
  Serial.println("AT+CWLAP");

  // read the response from the ESP8266
  String response = Serial.readString();


  if (response.indexOf("No AP") != -1) {
    // if no networks are found, print "No networks found" on the LCD
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("No networks");
    lcd.setCursor(0, 1);
    lcd.print("found");
    delay(1000); // delay 1 second between each network
  } else {
    // if networks are found, print the SSID and RSSI of each network on
the LCD
    lcd.clear();
```

```
    int index = 0;
    while (index != -1) {
      // find the next SSID in the response string
      int ssidStartIndex = response.indexOf("\"", index);
      if (ssidStartIndex == -1) break;
      int ssidEndIndex = response.indexOf("\"", ssidStartIndex + 1);
      String ssid = response.substring(ssidStartIndex + 1, ssidEndIndex);

      // find the next RSSI in the response string
      int rssiStartIndex = response.indexOf(",", ssidEndIndex);
      int rssiEndIndex = response.indexOf(",", rssiStartIndex + 1);
      String rssi = response.substring(rssiStartIndex + 1, rssiEndIndex);

      if(rssi.length() == 0 || ssid.length() == 0){
        index = response.indexOf("\n", rssiEndIndex);
        continue;
      }

      int numLeds = mapRssiToNumLeds(rssi.toInt());

      // Set the first "numLeds" LEDs on the strip to green
      for (int i = 0; i < numLeds; i++) {
        strip.setPixelColor(i, (((float)(max_Leds - i))/max_Leds) * 255,
(((float)i)/max_Leds) * 255, 0);
      }

      // Set the remaining LEDs on the strip to black (off)
      for (int i = numLeds; i < 10; i++) {
        strip.setPixelColor(i, 0, 0, 0);
      }

      // Show the updated pixel colors on the strip
      strip.show();

      // print the SSID and RSSI on the LCD
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print(ssid);
      lcd.setCursor(0, 1);
      lcd.print(rssi);
```

```
      delay(1000); // delay 1 second between each network

      while(!digitalRead(6))
        delay(10);

      // set the index for the next iteration
      index = response.indexOf("\n", rssiEndIndex);
    }
  }
}

// Function to map an RSSI value to a number of LEDs
int mapRssiToNumLeds(int rssi) {
  // Set the minimum and maximum RSSI values
  // that will be used for mapping
  const int minRssi = -100;
  const int maxRssi = -50;

  // Map the RSSI value to the range 0-10
  int numLeds = map(rssi, minRssi, maxRssi, 0, 10);

  // Limit the number of LEDs to the maximum number
  // of LEDs on the strip
  numLeds = min(numLeds, 10);

  return numLeds;
}
```