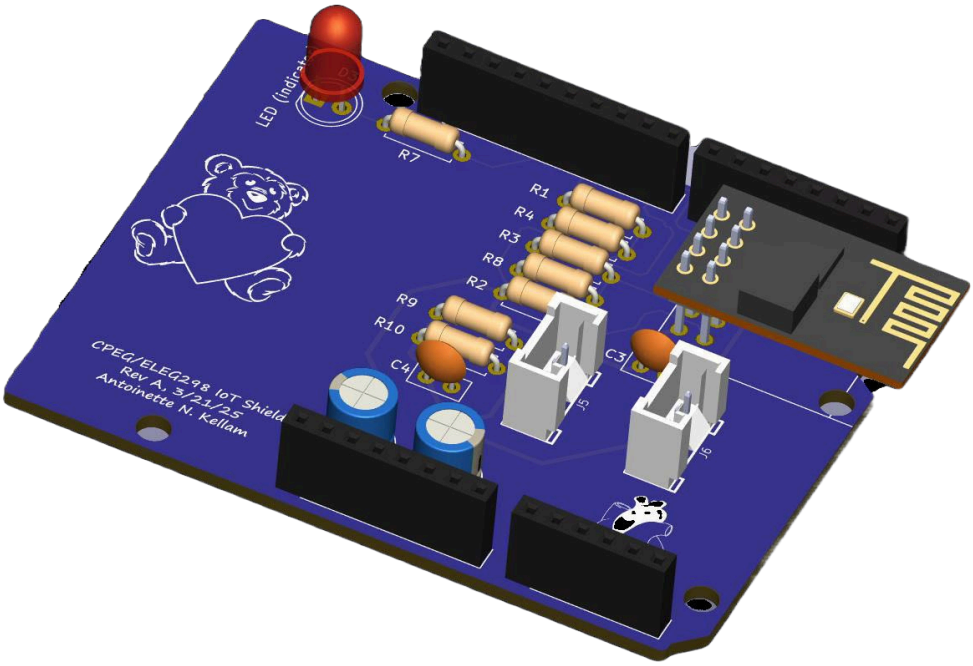


3D Image From KiCad

PulseSync IoT: Real-Time Heart-Rate Monitor w/LED Synchronization

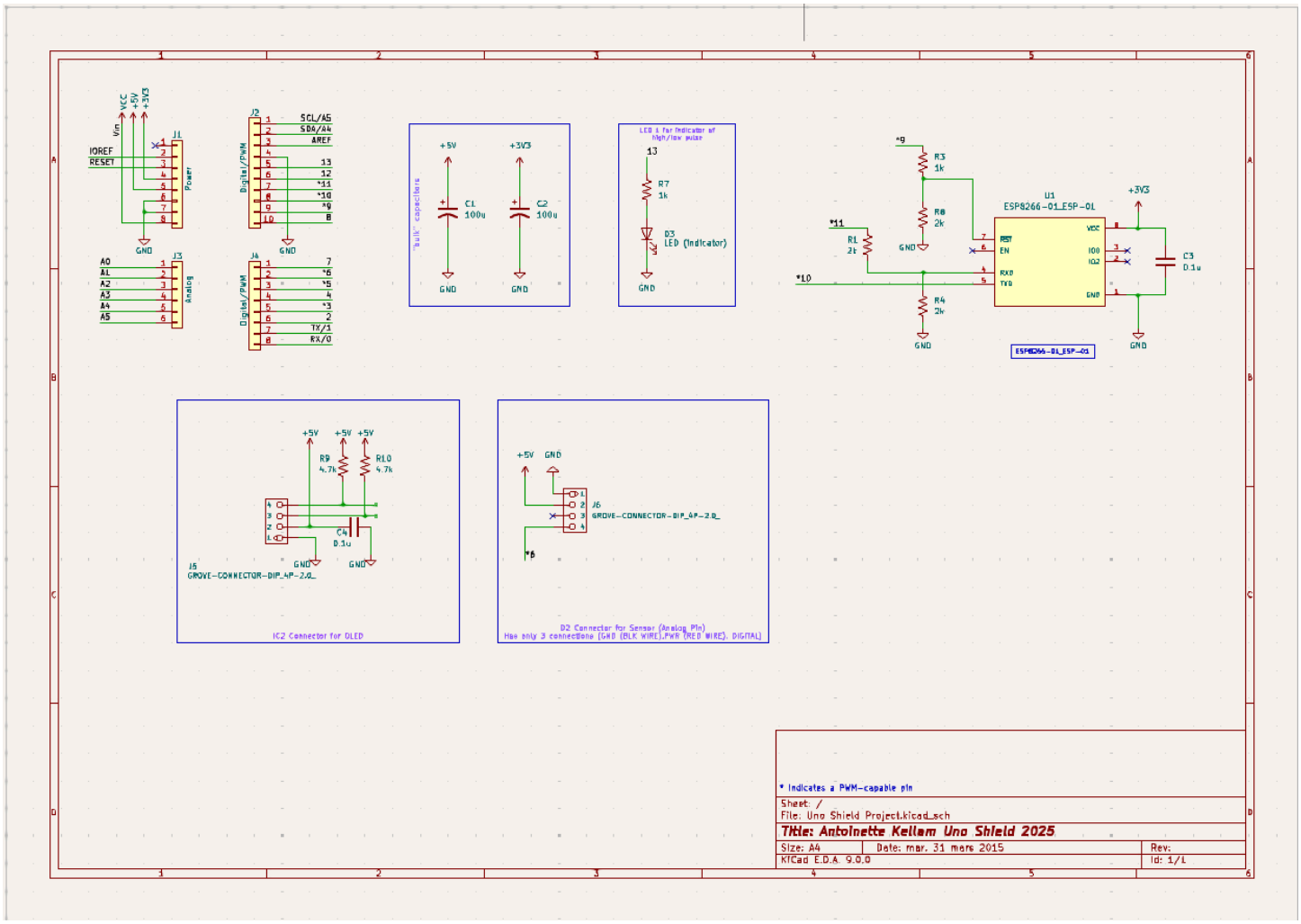
CPEG/ELEG298
Antoinette Kellam

Introduction/Project Description

The Heart Rate Sensor Pulse Sync IoT Device was designed to monitor and track one's heart rate. This device uses the **Grove-Ear-Clip Heart Rate Sensor.** The device will receive data and be sent to the cloud via the **ESP01 module, which** controls the WiFi communications. The sensor measures the pulse in real time using the **PPG (photoplethysmography),** which is an infrared LED. The clip can be worn on the earlobe or finger with this PPG technology. The module delivers PWM to Arduino Uno via interrupts (pin-specific). This power supply is a 3.3- 5V supply. The calculations are made using the code (references further in the report) to calculate the individual's instantaneous heart rate and update roughly every 1 sec for the heartbeat intervals and calculations (in BPM). I utilized an LED to correspond and flash for every beat of one's heartbeat. This is done simultaneously while still maintaining the calculations of BPM and sending the data via the cloud.

The Circuit/Schematic

The KiCad schematic (.kicad_sch) of my IoT Heart Rate device is shown below. The Arduino UNO connections, like power and I/O headers, are shown in the ***upper left quadrant***. In the **top-center,** I have 2 bulk capacitors (100uF on 5V and +3V3 rails). There was also a D13 LED which can be used to indicate and become in sync with one's heartbeats. The *upper-far-right* displays the **ESP8266-01** WiFi module with its 3.3V bypass capacitors and resistors that are in connections to the Arduino Uno 5V UART and EN lines. The **lower-left** houses the Grove I2C port connector for an OLED screen that could be used to display the current bpm (**this was not functioning properly).* **Centered** below my "bulk capacitors" is the Grove Senor port that houses the main sensor used in this project. This was a 3-wire connector for the **Grove-Ear-Clip Heart Rate Sensor: GND, PWR, and Digital Pin** that supplies 5V and GND.



Below is a picture of the device, the PPG Sensor System. There is a picture and schematic of the earlobe/finger sensor **HRM-2511B**.

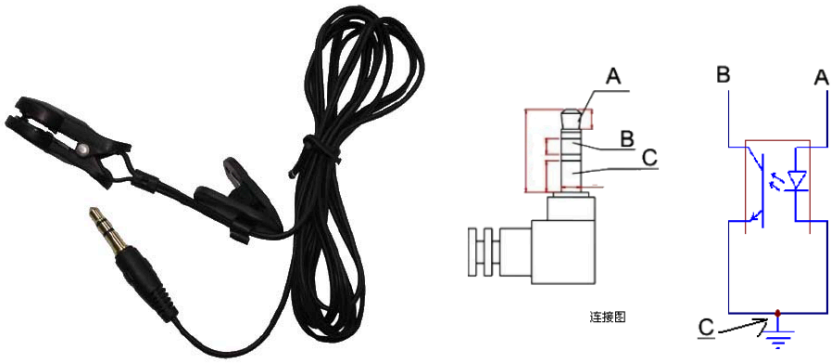


Figure A.1: Ear lobe pulse sensor HRM-2511B

Reference: PPG Sensor System PDF

Below is the functionality of the PPG sensor and its depictions of using the light transmitted functionality to measure the pulse.

A PPG sensor can be used in reflection mode (for example on the finger) or in transmission mode (for example on the ear) as shown in Figure 1.1. Normally, a wavelength in the near-infrared is used because there we have the strongest modulation of the signal due to light absorption in the haemoglobin in the blood[1].

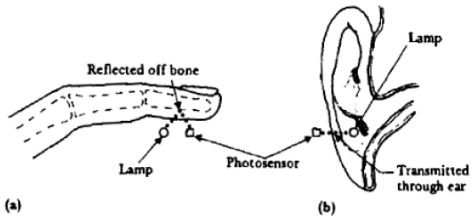


Figure 1.1: (a) Light transmitted into the finger pad is reflected off bone and detected by a photo sensor. (b) Light transmitted through the aural pinna is detected by a photo sensor [1]

Table A.1: Specifications of the two optoelectronic components in the HRM-2511B ear-lobe clip

	LED	Photo transistor
Manufacturer	Lucky Light Electronics Co. China	Lucky Light Electronics Co. China
Device code	LL-AR180IRC-2A	LL-AR180PTC-1A
Chip material	GaAlAs	InGaN
Max power	130mW	150mW
Forward current	1A (1/10 duty cycle, 0.1ms pulse width)	.
Continuous forward current	65 mA	.
Reverse voltage	5V	.
Forward voltage	1.2V (I_f = 70mA)	.
Reverse current max	10μA (V_r = 5V)	5-30nA (dark 0mW/cm^2, V_r = 10V)
Operating temperature range	-50°C to +85°C	-20°C to +80°C
Storage temperature	-40°C to 100°C	-40°C to +85°C
Peak wavelength	940nm (I_f=20mA)	940nm
Spectral bandwidth	45nm (I_f=20mA)	400-1200nm ($\lambda_{0.5}$)
Radiant intensity	3.0mW/sr (I_f=20mA)	
Open circuit voltage	.	0.41 V (5mW/cm^2, 940nm)
Short-circuit current	.	7 μA (5mW/cm^2, 940nm)
Reverse light current	.	4.5 μA (5mW/cm^2, 940nm, V_r = 5V)
Reverse breakdown voltage	.	32-50V (0mW/cm^2, 100μA)
Total capacitance	.	25pF (0mW/cm^2, V_r=3V, 1MHz)
Response time (rise and fall)	.	50ns (V_r=10V, R_L=1kΩ)

Reference: PPG Sensor System PDF

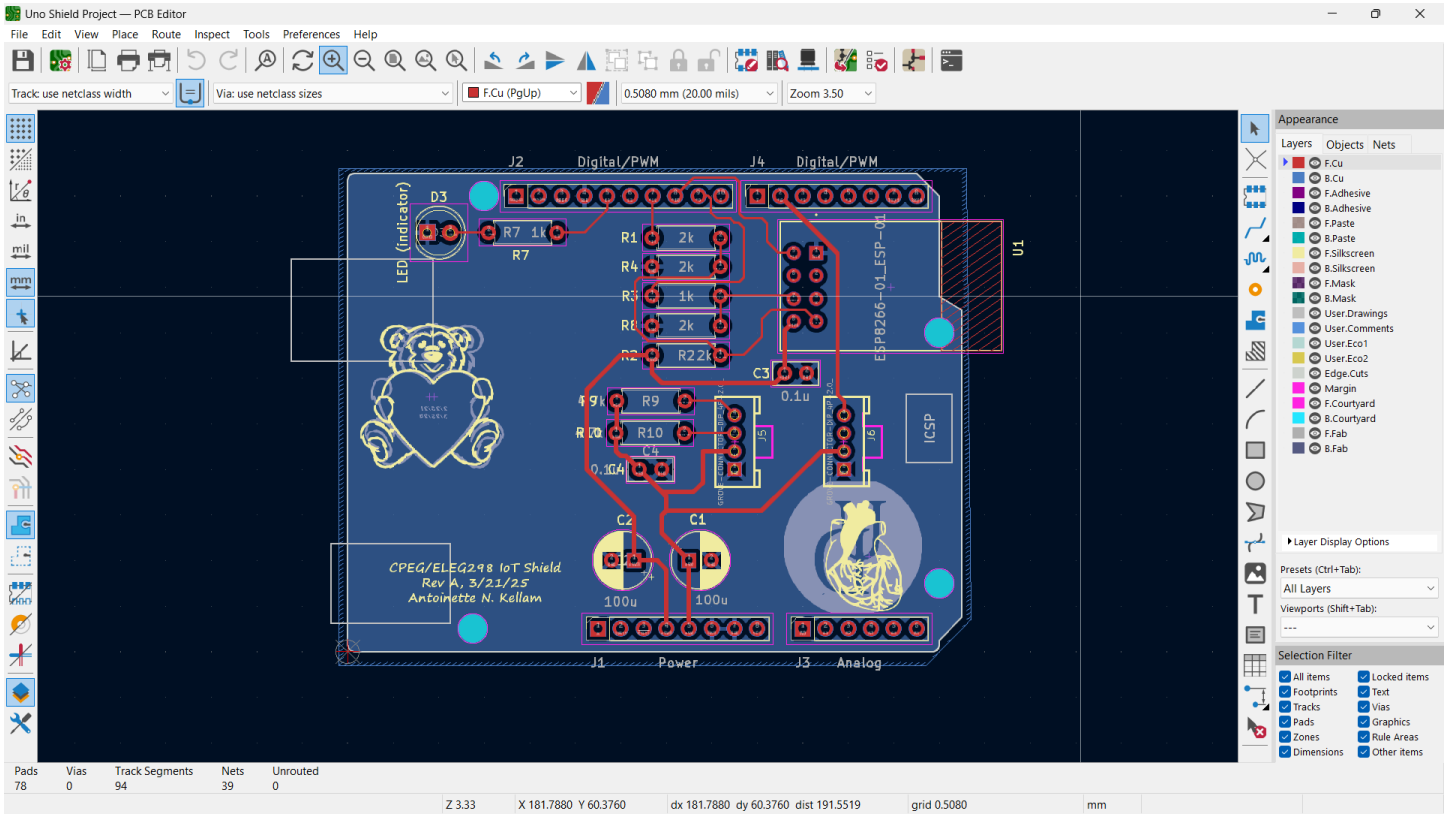
An **ESP-01S** module incorporating an **ESP8266** (*a 3.3V device*) was used to connect the shield to the WiFi and to allow the IoT device to send data to the cloud (Adafruit IO). The VCC (pin 8) and the EN/CH_PD (pin 6) are tied to the 3.3 V

DC supplied by the Arduino. The TX output (pin 5) is tied directly to GPIO 10 of the Arduino and the RX input (pin 4) has a 2KΩ/1K Ω voltage divider to reduce the 5 V UART signal from the Arduino (GPIO 11) to 3.3 V. Since the Arduino only has one hardware UART (GPIO0 and GPIO1) which is used for programming a printing to the serial monitor, a software serial UART is created using GPIO10 and 11 in the software. The AT Command firmware on the **ESP8266** has been replaced with special software to allow the device to connect to the internet and send data to Adafruit IO. Another *2KΩ/1KΩ* voltage divider allows digital output on GPIO8 to reset the ESP-01S model and have it boot up off the new firmware in the flash memory.

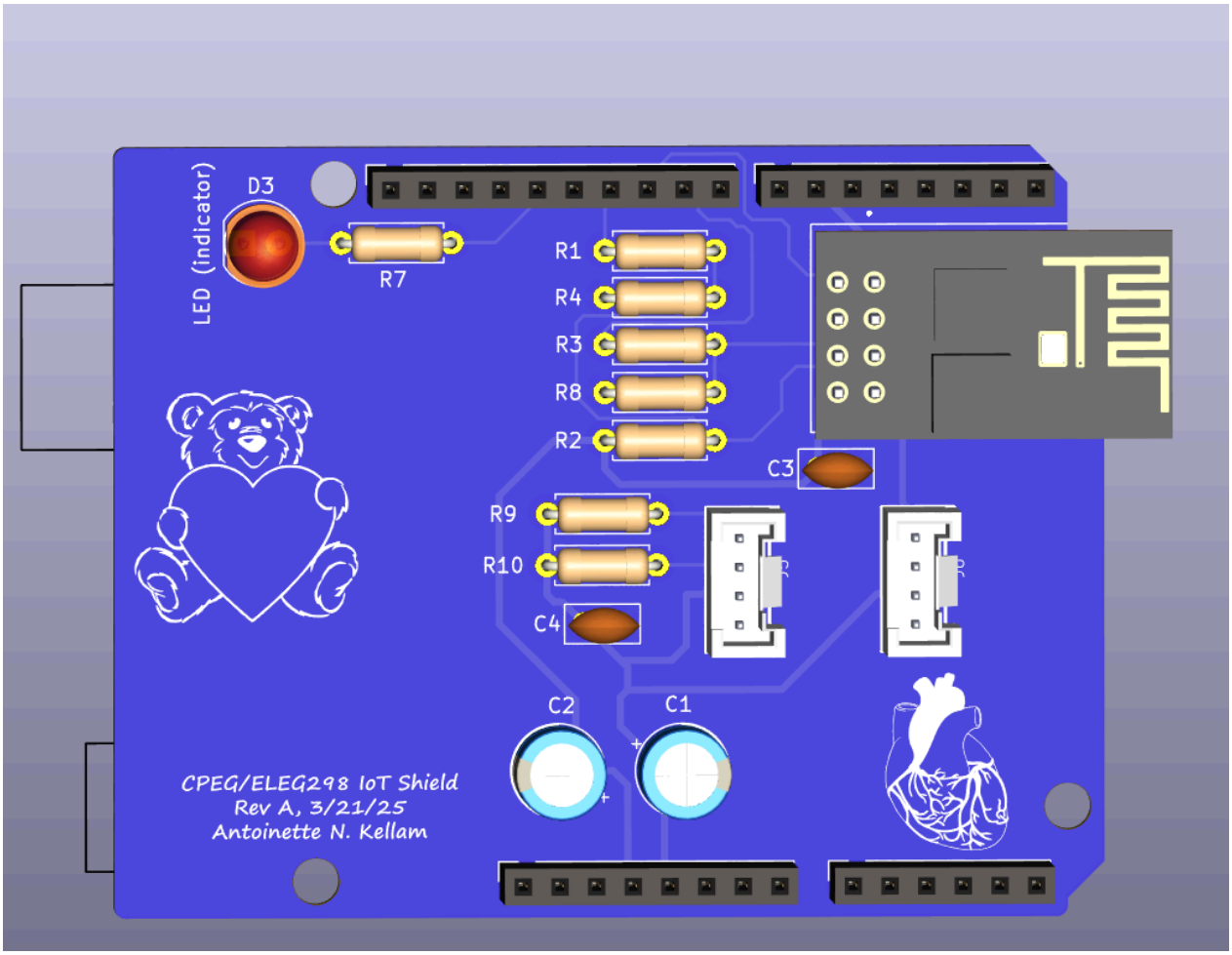
Two 2 mm pitch Grove connectors are used to attach the Grove I2C 0.96" OLED screen and the D2 connections for the sensor. The OLED uses the **Inter-Integrated Circuit (I2C)** serial communication interface, which is a 128×64-pixel passive display matrix module for various display information.

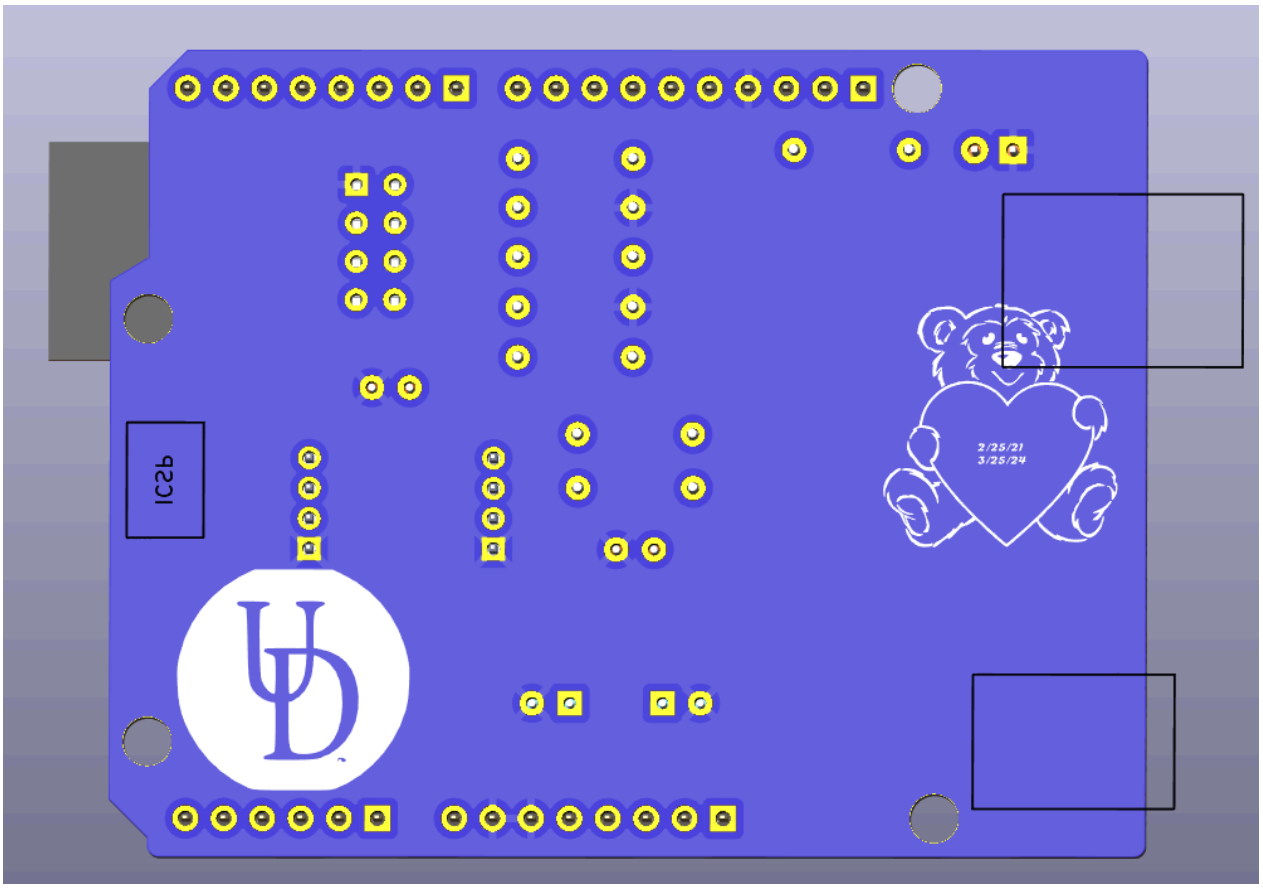
Printed Circuit Board Design

My KiCad PCB board is shown below. The board is a two-layer board with a ground plane as the bottom layer. I used the Arduino Uno Template to begin this project and built upon that. This was useful as the dimensions of the templates match for placement on the Arduino. I was then able to update and place parts/components in my schematic and bring them over into the PCB editor, and begin the process.



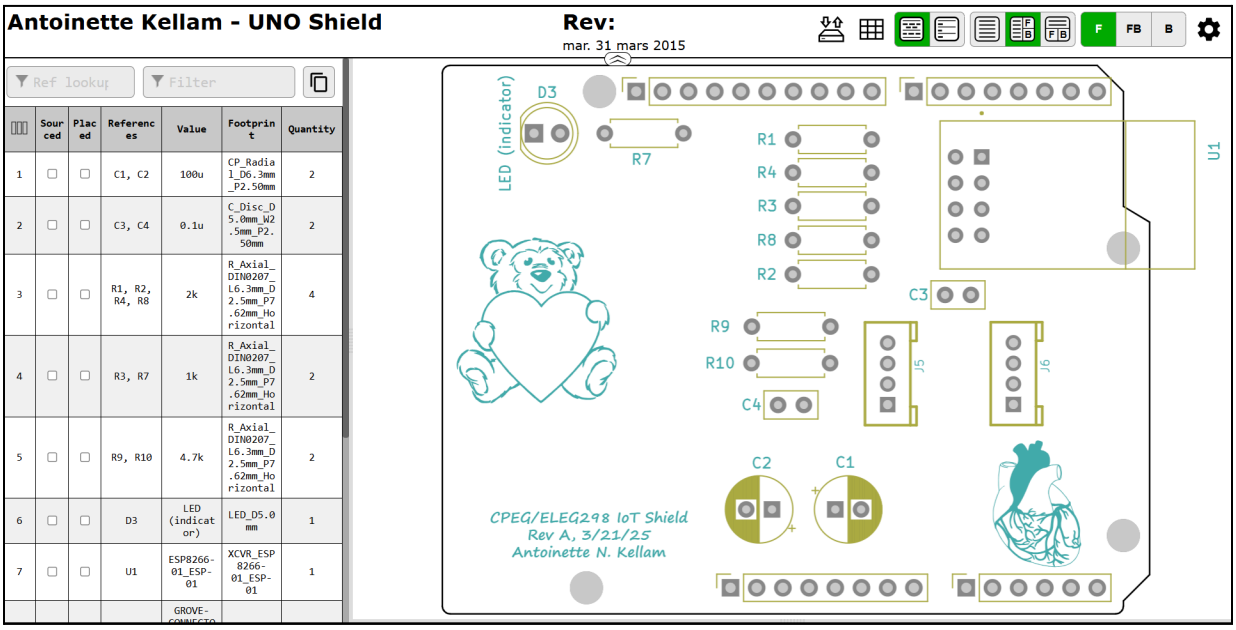
Manufacturing images of the top and back of the PCB taken from the KiCad 3D viewer are shown below. A purple solder mask and white silkscreen were the colors I wanted. The boards were manufactured by JLCPCB (https://www.jlcpcb.com).

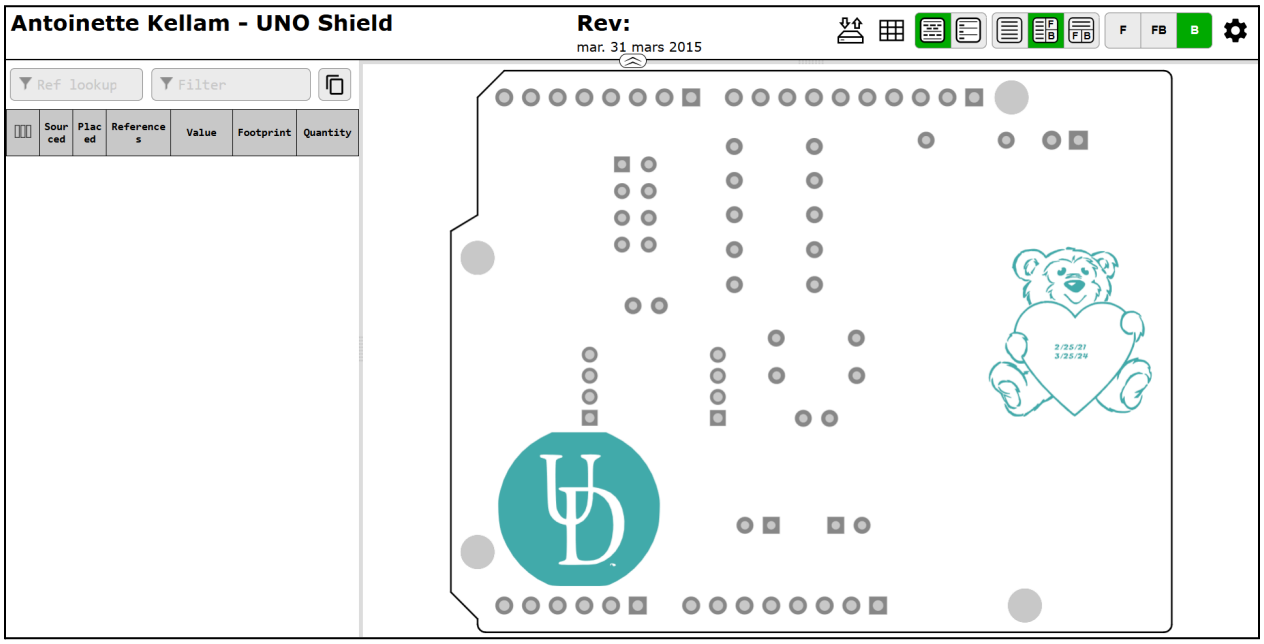




Component List/ Bill of Materials

All components are on the top side of the printed circuit board. A component placement diagram, or a “stuffing” diagram, is shown below. The BOM table is listed below as well.





Bill of Materials

Id	Designator	Footprint	Quantity	Designation	Supplier and Ref
1	J1	PinSocket_1x08_P2.54mm_Vertical	1	Power	
2	J3	PinSocket_1x06_P2.54mm_Vertical	1	Analog	
3	J2	PinSocket_1x10_P2.54mm_Vertical	1	Digital/PWM	
4	J4	PinSocket_1x08_P2.54mm_Vertical	1	Digital/PWM	
5	R4,R1,R2,R8	R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal	4	2k	
6	U1	XCVR_ESP8266-01_ESP-01	1	ESP8266-01_ESP-01	
7	C3, C4	C_Disc_D5.0mm_W2.5mm_P2.50mm	2	0.1u	
8	D3	LED_D5.0mm	1	LED (indicator)	
9	J6, J5	GROVE-HW4-2.0	2	GROVE-CONNECTOR-DIP_4P-2.0	
10	R7, R3	R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal	2	1k	
11	R10,R9	R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal	2	4.7k	
12	C2, C1	CP_Radial_D6.3mm_P2.50mm	2	100u	

The parts were sourced from the University of Delaware undergraduate labs and DigiKey.
The unique parts and prices are shown in the table below:

Component	**DigiKey part #**	**Price/each**	**Description**
ESP8266-01/ESP-01	3647-Ai-ThinkerESP-01S ESP8266-ND	$3.96	Ai-Thinker ESP-01S ESP8266 Wi-Fi
ADS1115IDGS	296-38849-1-ND	$5.30	IC ADC 16BIT SIGMA-DELTA 10VSSOP
Conn_01x04	1597-1082-ND	$1.70 [10]	GROVE 2MM 4PIN VERT CONN 10PCS
SJ1-3535NG	CP1-3535NG-ND	$1.23	CONN JACK STEREO 3.5MM R/A
PJ-047AH	CP-047A-ND	$0.88	CONN PWR JACK 2X5.5MM SOLDER
SK6812 NeoPixel	1528-1104-ND	$4.06 [10]	ADDRESS LED DISC SERIAL RGB 1=10
Grove DHT20 sensor	1597-101020932-ND	$5.86	GROVE TEMP+HUM SENSOR V2.0 DHT20
Grove 0.96” OLED	1597-104020208-ND	$4.96	GROVE - OLED DISPLAY 0.96" (SSD1313)
YHDC SCT013-000	1597-1307-ND	$10.36	CURR SENSE XFMR 100A:0.05A 100A
WAU12-200	237-1880-ND	$9.03	AC/AC WALL MNT ADPT 12VAC 200MA

Software/Arduino Code and Libraries

The complete Arduino sketch is included in the Appendix of this report. The sketch is entitled **HeartRate_Sensor_AKEL** and is well documented. The following libraries were used:

```
#include <Arduino.h>
#include <SoftwareSerial.h>  // Allows us to use two GPIO pins for a second UART
```

The following pre-directives were used to define the pins and addresses of sensors and LED pin:

```
// ——— Pin definitions ———
#define RESET_PIN        9   // ESP-01S CH_PD/RESET
#define UART_RX_PIN      11  // ESP-01 TX → D11
#define UART_TX_PIN      10  // ESP-01 RX ← D10
#define HEART_SENSOR_PIN 6   // Grove sensor → D6
#define LED_PIN          13  // Indicator LED net "13" in your schematic
```

The setup() software establishes the serial communications, GPIO pins, LED, and Heart Rate sensor. It then establishes the internet connection and sets up the data feeds for Adafruit IO.

```
  // — Serial setups —
  Serial.begin(115200); // for the monitor
  espSerial.begin(9600); // for the commands

  // — Reset ESP-01S —
  pinMode(RESET_PIN, OUTPUT);
  digitalWrite(RESET_PIN, LOW);
  delay(1000);
  digitalWrite(RESET_PIN, HIGH);
  delay(2000); // 2 sec loop reading

  // — Sensor & LED pins —
  pinMode(HEART_SENSOR_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);

  // — Drain any stray ESP bytes —
  unsigned long t0 = millis();
  while (millis() - t0 < 2000) {
    while (espSerial.available()) espSerial.read();
    pollHeartSensor();
  }

  // — Adafruit IO handshake —
  Serial.println("\n\n--- Heart Rate → Adafruit IO ---");
  espSend("get_macaddr");
  espSend("get_version");
  espSend("wifi_ssid=" + WIFI_SSID);
  delay(500);
  espSend("wifi_pass=" + WIFI_PASS);
  delay(500);
  espSend("io_user=" + IO_USERNAME);
  espSend("io_key="  + IO_KEY, 2000, false);

  String r = espSend("setup_io", 30000);
  if (r.indexOf("connected") < 0) {
    Serial.println("Adafruit IO Connection Failed");
    while (1);
  }
  espSend("setup_pubfeed=1,HeartRate");
```

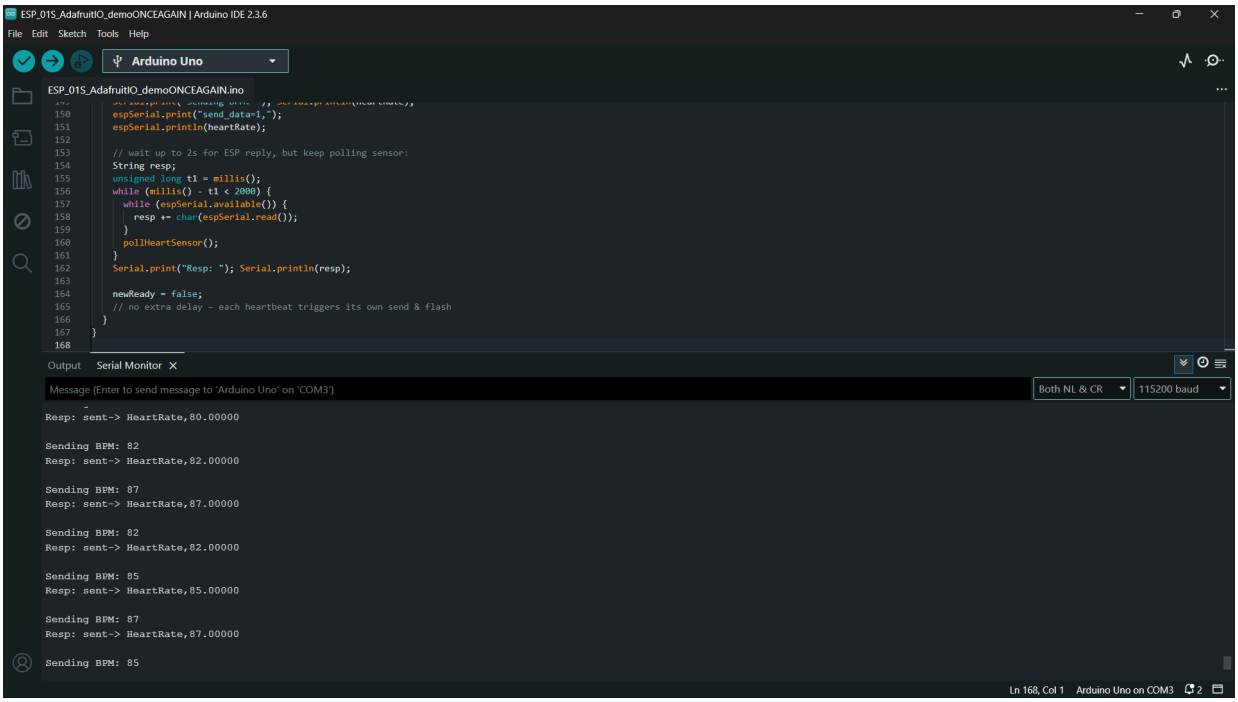
```
  Serial.println("------ Setup Complete ------");
}
```

If the setup succeeds, the neopixels will be green, and a starting millisecond reading is obtained before starting the main program loop.

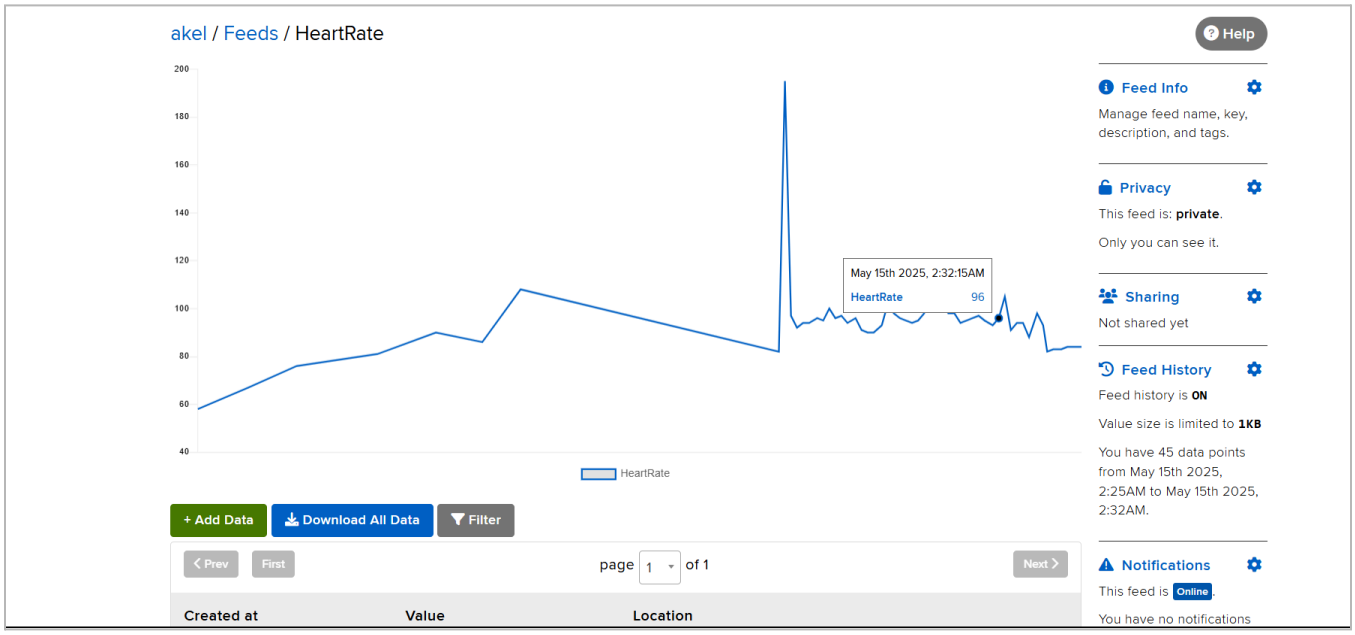
The main loop continuously polls the heart rate sensor while simultaneously having the RED LED flash on each beat.

```
// always poll sensor first (handles LED off-timer too)
  pollHeartSensor();

  if (newReady) {
    // send immediately on each beat
    Serial.print("Sending BPM: "); Serial.println(heartRate);
    espSerial.print("send_data=1,");
    espSerial.println(heartRate);

    // wait up to 2s for ESP reply, but keep polling sensor:
    String resp;
    unsigned long t1 = millis();
    while (millis() - t1 < 2000) {
      while (espSerial.available()) {
        resp += char(espSerial.read());
      }
      pollHeartSensor();
    }
    Serial.print("Resp: "); Serial.println(resp);

    newReady = false;
    // no extra delay – each heartbeat triggers its own send & flash
  }
```

A screenshot of the Arduino IDE in progress of calculating and sending heart rate BPM to AdaFruit



Here was some data of the BPM of my heart rate at various times on the AdaFruit Feed & Dashboard



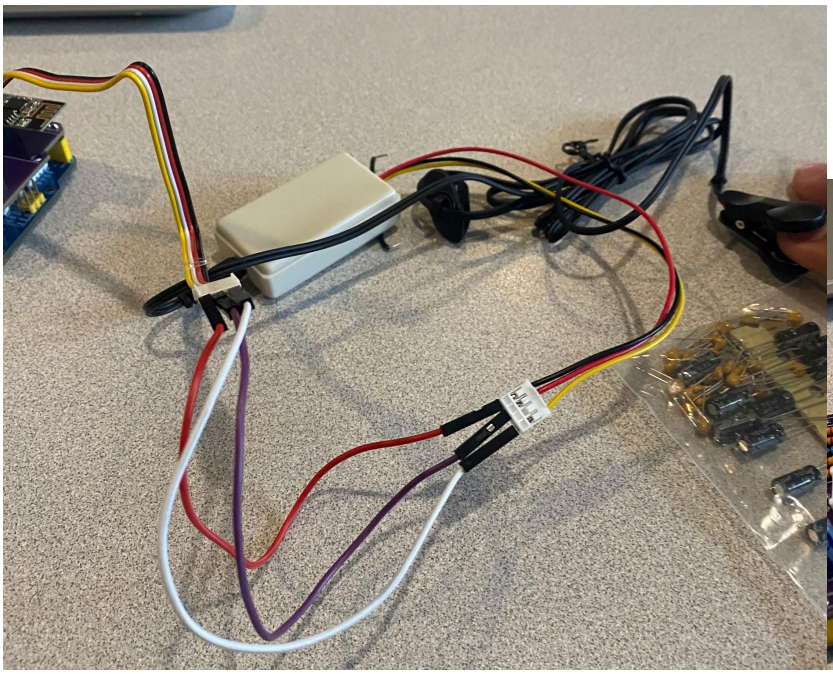
Adafruit IO dashboard and cloud data

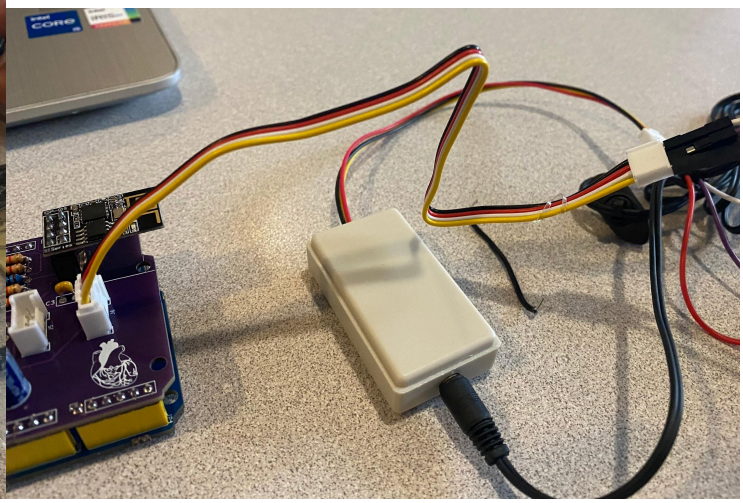
A screenshot of the Adafruit IO dashboard is shown above. The values are sent roughly every 1 second to the cloud service, and the display is updated. The periodic spikes are from movement of one's finger (while removing the sensor) or an increase in heart rate.

Conclusion/Possible Changes

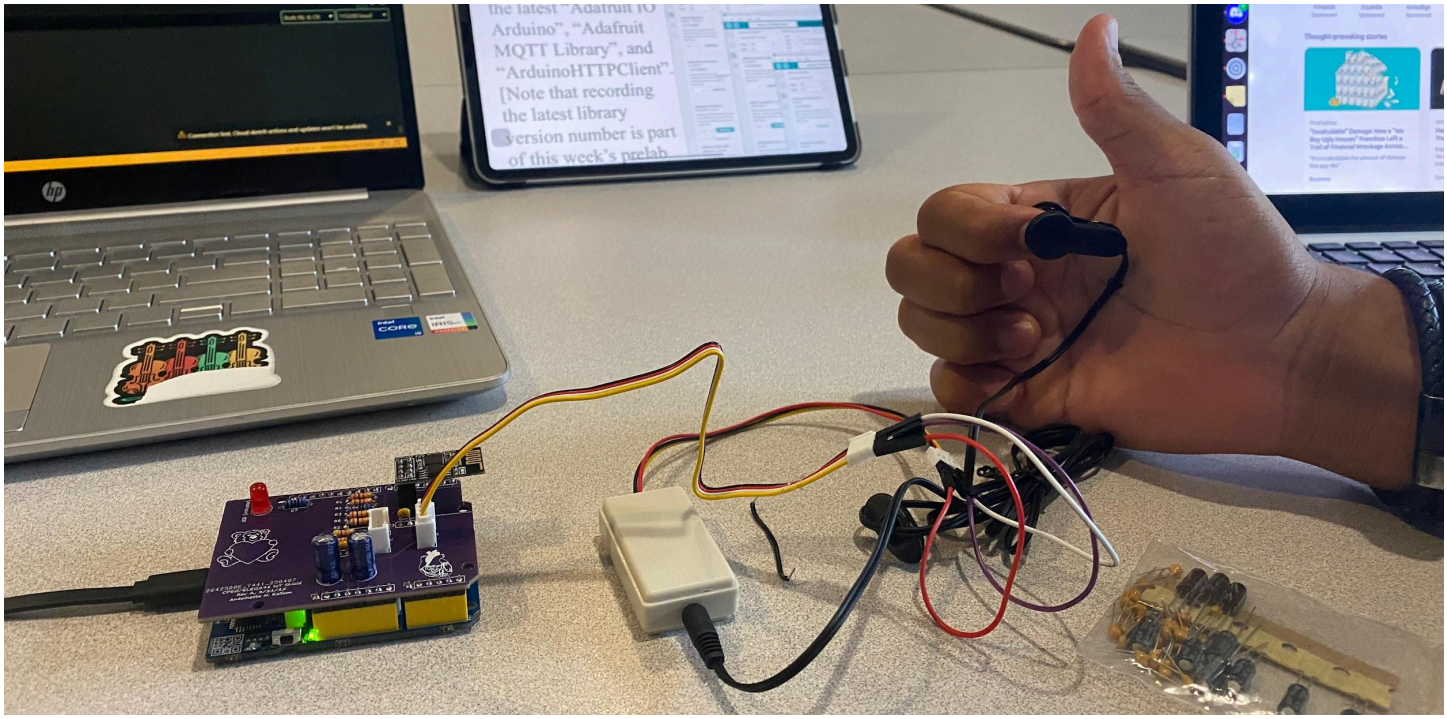
The IoT shield overall works somewhat well. There were a lot of issues I ran into after altering my code. There were errors, I do believe, with the overall PCB that can be fixed within the next design. When trying to connect with the Grove connector for the sensor. I was not getting stable answers with the calculations of the heart rate at first. The erratic heart rate and miscalculations (like 65366 bpm) were being read from the sensor. But then realized that my Grove connector was incorrectly initialized and the pins were not matching with the PCB. I then had to manually wire the correct connections, and the calculations were stable and present. The correct connections that I had were for D6 and not D2, as I had originally. So rewiring corrected this issue.

Here, the rewriting of the sensor connections back to the shield on the PCB



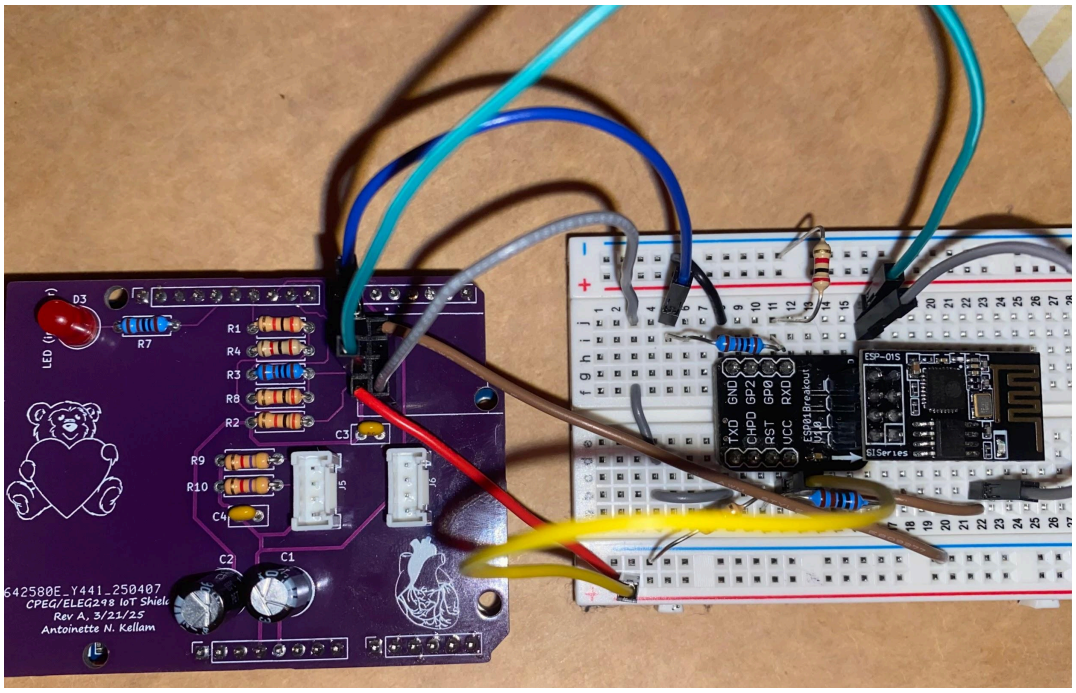


A photo of my project in progress, with the sensor working



I also had issues with the ESP01 Wifi Module and connections. After long hours of troubleshooting, I then realized that my PCB (possibly the traces) was not supplying the correct commands to it. I had to reflash the module and breadboard and use jumper wires to make the necessary connections to my PCB. This then worked.

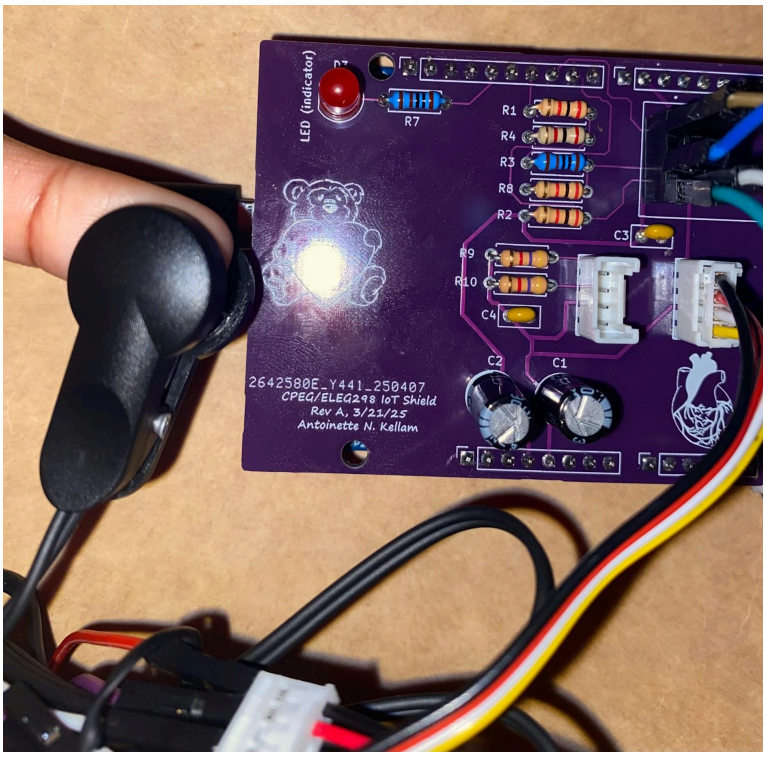
A picture of the ESP01 module connected to my Uno Shield

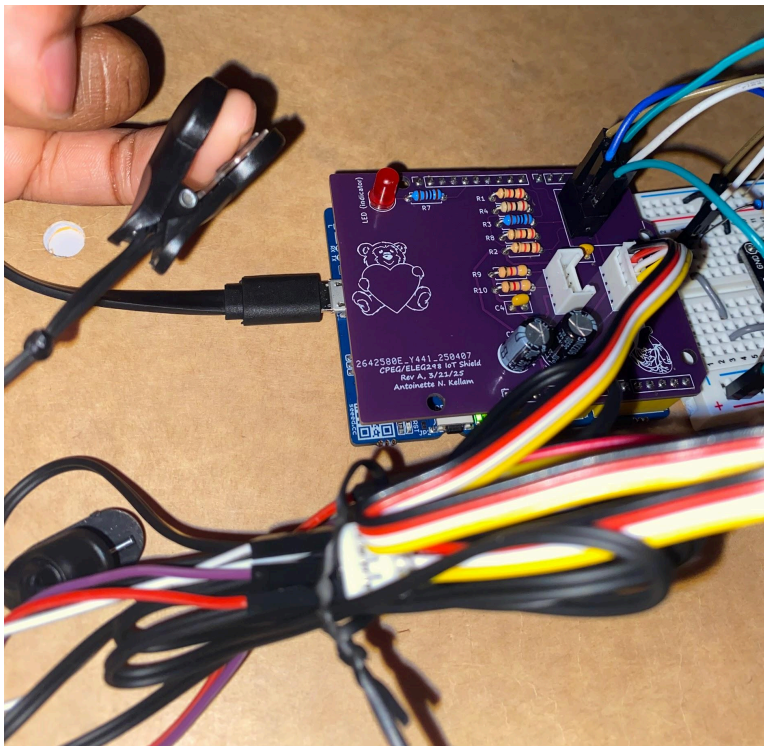


Once this issue with the ESP01 module was fixed, I then encountered the issue of having both the D6 sensor and the module together. There were firmware issues with having the AT firmware only sent back basic commands (e.g., setup_io), and no resp. Using the SoftwareSerial on D10/D11 had issues with the pin change of D6 (the correct heart rate sensor pin), so I had to switch things up in the code loop (polling). I then went on to incorporate my LED (D13), which flashes synchronized to one's heartbeat while continuously polling. By reconstructing the code, editing the pins, and using jumper wires, the project became a success (even though the wires are a little messy).

The overall sensor worked great. In comparison to my Apple Watch, the bpm values were close, with maybe a discrepancy of about 3 or 4 beats ahead. If no cloud software were being used, I would highly recommend (besides using the serial monitor) incorporating the OLED screen for maximum use of Some things that I think I would change in my design would be to have this device also track respiration and have that displayed on the OLED, along with the other relevant information. I would also change the connectors to the 90-degree angle connectors for easier placement of the **Grove Connector** connections. I would change it to a status LED, which would be bright or change color when connections are successfully established. I would also consider firmware specifications and clear routing to make sure this project ran more smoothly and without the use of jumper wires (especially the ESP01 Wifi Module).

A photo of my completed project is shown below.





Appendix: Arduino Code

//***

```C/C++
/*************************************************************************
 Antoinette Kellam - CPEG298 - Spring 2025

    ESP-01S + Grove Ear-Clip Heart Sensor on D6 → Adafruit IO
  • SoftwareSerial on D10/D11 for ESP-01
  • Poll D6 for beats, flash D13 at each beat
  • Publish instantaneous BPM immediately (and keep polling during send)
*************************************************************************/
#include <Arduino.h>
#include <SoftwareSerial.h>  // Allows us to use two GPIO pins for a second UART

// --- Pin definitions ---
#define RESET_PIN         9   // ESP-01S CH_PD/RESET
#define UART_RX_PIN      11   // ESP-01 TX → D11
#define UART_TX_PIN      10   // ESP-01 RX ← D10
#define HEART_SENSOR_PIN  6   // Grove sensor → D6
#define LED_PIN          13   // Indicator LED net “13” in your schematic

// --- SoftwareSerial for ESP-01 ---
SoftwareSerial espSerial(UART_RX_PIN, UART_TX_PIN);

// --- Adafruit IO credentials ---
const String IO_USERNAME = "YOUR_ADAFRUIT_IO_USERNAME";
const String IO_KEY      = "YOUR_ADAFRUIT_IO_KEY";
const String WIFI_SSID   = "YOUR_WIFI_SSID";
const String WIFI_PASS   = "YOUR_WIFI_PASSWORD";

// --- Heart-beat detection globals ---
unsigned long lastBeatTime = 0;      // timestamp of last beat
unsigned int  heartRate    = 0;      // computed BPM
bool          newReady     = false;  // flag: a new BPM is ready

// --- LED flash timing (non-blocking) ---
unsigned long ledOnTime       = 0;
const unsigned long ledFlashDuration = 50; // ms

// --- For edge-detecting the sensor pin ---
bool lastSensorState = LOW;

// --- Helper: send AT-style cmd & collect response ---
String espSend(const String &cmd, int timeout=2000, bool dbg=true) {
```

```
  String resp;
  Serial.print("> "); Serial.println(cmd);
  espSerial.println(cmd);
  unsigned long start = millis();
  while (millis() - start < timeout) {
    while (espSerial.available()) {
      resp += char(espSerial.read());
    }
    // keep heart polling alive while waiting:
    // (so LED still flashes on beats during long publishes)
    bool now = digitalRead(HEART_SENSOR_PIN);
    if (now && !lastSensorState) {
      unsigned long t = millis();
      unsigned long delta = t - lastBeatTime;
      lastBeatTime = t;
      if (delta < 2000) {
        heartRate = 60000UL / delta;
        newReady  = true;  // allow immediate send back in loop
      }
      // schedule LED flash
      digitalWrite(LED_PIN, HIGH);
      ledOnTime = millis();
    }
    lastSensorState = now;
  }
  resp.trim();
  if (dbg) {
    Serial.print("Resp: "); Serial.println(resp);
  }
  return resp;
}

// --- Poll the heart sensor & LED off-timer ---
void pollHeartSensor() {
  bool now = digitalRead(HEART_SENSOR_PIN);

  // on rising edge → record a beat
  if (now && !lastSensorState) {
    unsigned long t = millis();
    unsigned long delta = t - lastBeatTime;
    lastBeatTime = t;

    if (delta < 2000) {
```

```
      heartRate = 60000UL / delta;  // instantaneous BPM
      newReady  = true;
    }

    // schedule LED flash
    digitalWrite(LED_PIN, HIGH);
    ledOnTime = millis();
  }
  lastSensorState = now;

  // turn LED off once its flash-duration is up
  if (ledOnTime && (millis() - ledOnTime >= ledFlashDuration)) {
    digitalWrite(LED_PIN, LOW);
    ledOnTime = 0;
  }
}

void setup() {
  // – Serial setups –
  Serial.begin(115200); // for the monitor
  espSerial.begin(9600); // for the commands

  // – Reset ESP-01S –
  pinMode(RESET_PIN, OUTPUT);
  digitalWrite(RESET_PIN, LOW);
  delay(1000);
  digitalWrite(RESET_PIN, HIGH);
  delay(2000); // 2 sec loop reading

  // – Sensor & LED pins –
  pinMode(HEART_SENSOR_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);

  // – Drain any stray ESP bytes –
  unsigned long t0 = millis();
  while (millis() - t0 < 2000) {
    while (espSerial.available()) espSerial.read();
    pollHeartSensor();
  }

  // – Adafruit IO handshake –
  Serial.println("\n\n--- Heart Rate → Adafruit IO ---");
  espSend("get_macaddr");
```

```
  espSend("get_version");
  espSend("wifi_ssid=" + WIFI_SSID);
  delay(500);
  espSend("wifi_pass=" + WIFI_PASS);
  delay(500);
  espSend("io_user=" + IO_USERNAME);
  espSend("io_key="  + IO_KEY, 2000, false);

  String r = espSend("setup_io", 30000);
  if (r.indexOf("connected") < 0) {
    Serial.println("Adafruit IO Connection Failed");
    while (1);
  }
  espSend("setup_pubfeed=1,HeartRate");
  Serial.println("------ Setup Complete ------");
}

void loop() {
  // always poll sensor first (handles LED off-timer too)
  pollHeartSensor(); //edge detects rising pulse from D6

  if (newReady) {
    // send immediately on each beat
    Serial.print("Sending BPM: "); Serial.println(heartRate);
    espSerial.print("send_data=1,");
    espSerial.println(heartRate);

    // wait up to 2s for ESP reply, but keep polling sensor:
    String resp;
    unsigned long t1 = millis();
    while (millis() - t1 < 2000) {
      while (espSerial.available()) {
        resp += char(espSerial.read());
      }
      pollHeartSensor();
    }
    Serial.print("Resp: "); Serial.println(resp);

    newReady = false;
    // no extra delay — each heartbeat triggers its own send & flash
  }
}
```