Project Mon-Tor

A surface EMG recording device

Wu, Ting-Wei 105501539

Supervised by Chao-Min Wu Ph.D.

Abstract:

In this project I build a device that can record voluntary surface electromyography (sEMG) signal. When placing surface electrode on the location recommended by the SENIAM (Surface ElectroMyoGraphy for the Non-Invasive Assessment of Muscles) project, this device records the signal after it was being amplified, filtered and converted into a 10-bit digital signal. The recorded signal is saved to the flash memory as a comma separated value (CSV) file of a microcontroller (MCU). Operating in Wi-Fi station mode, this MCU sends the CSV file to the client on the web server hosted by the MCU. Using the website as a user interface, some option can be made, including changing sampling period (30s max for dual channels), changing sampling frequency (1k or 2k samples per second), calculate the RMS of the recorded signal etc. The device is powered by an 18650 li-ion battery with 5 hours continuous operation and 12 hours on standby.

Introduction:

While commercially available wireless surface electromyography device is often small, expensive and using complicated user interface, there exist little to none cheaper alternatives with bulkier appearance perhaps due to the lack of such market.

When resistance training athletes discuss which move set is best for increase maximum muscle strength or muscle mass, muscle activation is often the phrase brought upon said discussion. This project is aimed to provide the means to objectify such term to the mass by building a device that can record voluntary sEMG signal using surface electrode. The recorded analog signal is converted into digital signal and then sent to a phone via wireless communication.

Method:

According to the purpose of the project, I set some primary and secondary objectives.

Primary objective including:

1. Ability to measure sEMG
2. Sampling frequency of 1k at least
3. Battery powered
4. Send data by website server

Secondary objective including:

1. Real time measurement on the website

2. Multi-channel input
3. Make a printed circuit board version
4. Basic digital signal process on client side

The project is separated by time into three phases, first I did some research about how to obtain bio signals, muscle physiology, web server, html markup language and programming of MCU.

In the second phase I make prototypes on bread board after simulating the circuit on the computer. The third phase includes moving the system to perf board or printed circuit board and trying secondary objective.

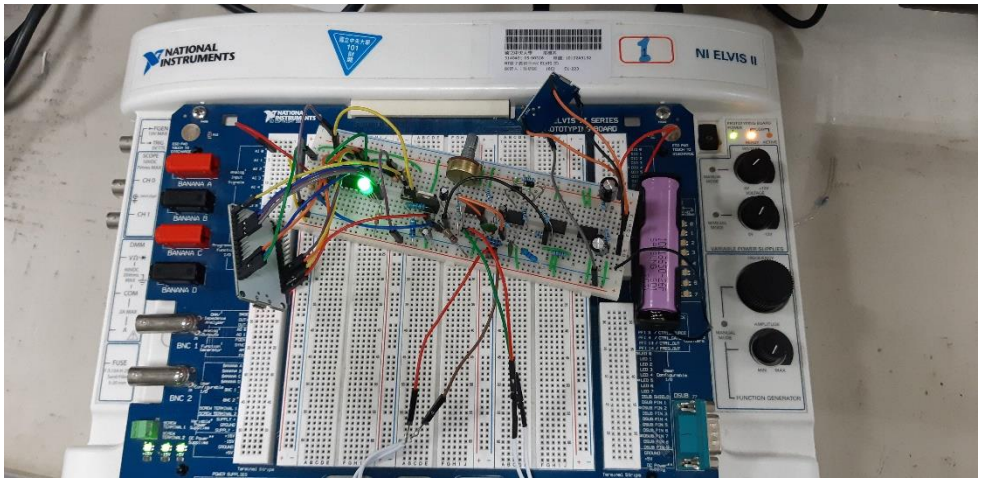


Figure 1. Build a prototype with breadboard and test it with ELVIS 2.

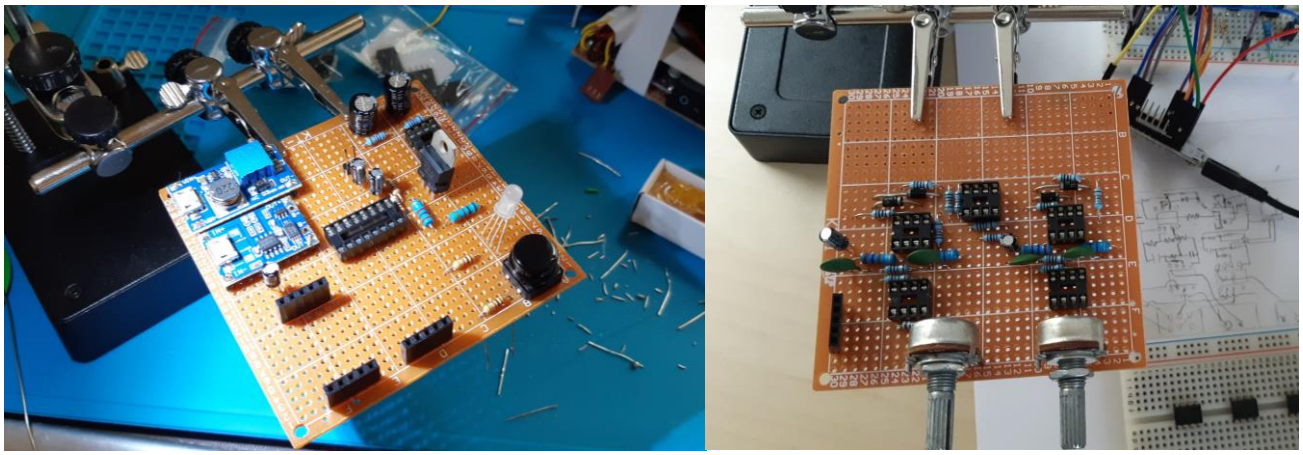


Figure 2. Building the prototype on perfboard.

Result:

This project manages to meet the primary objectives, including measures surface EMG, sampling frequency of 1k, operates on battery, and sends the result to remote devices through web server with Wi-Fi connection. Some secondary objectives are met, including multi-channel measuring surface EMG, make PCB version, and realize simple signal processing on client side, for instance, root mean square with window size 250.

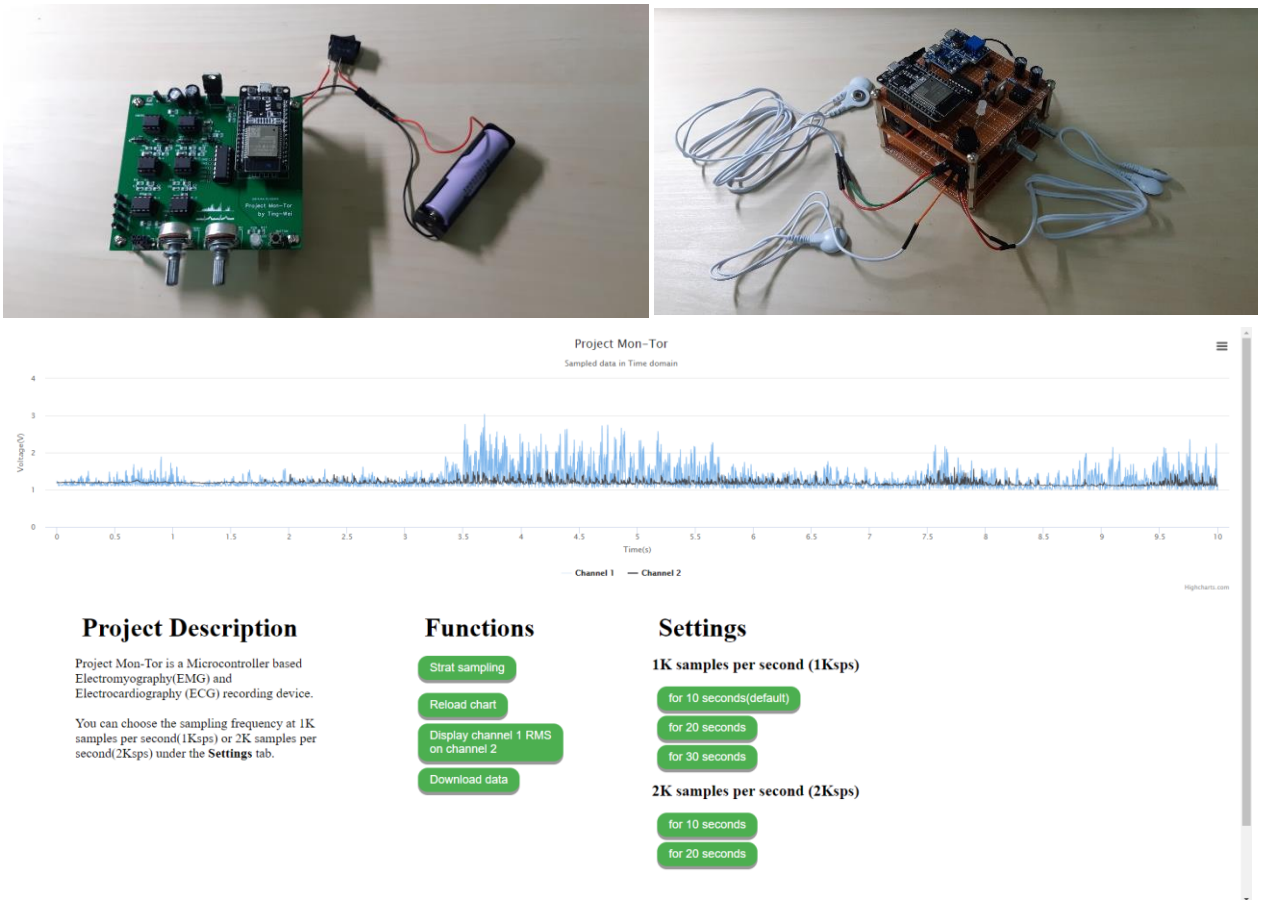


Figure 3. Device and interface of this project. Top left: PCB version, top right: perfboard version.

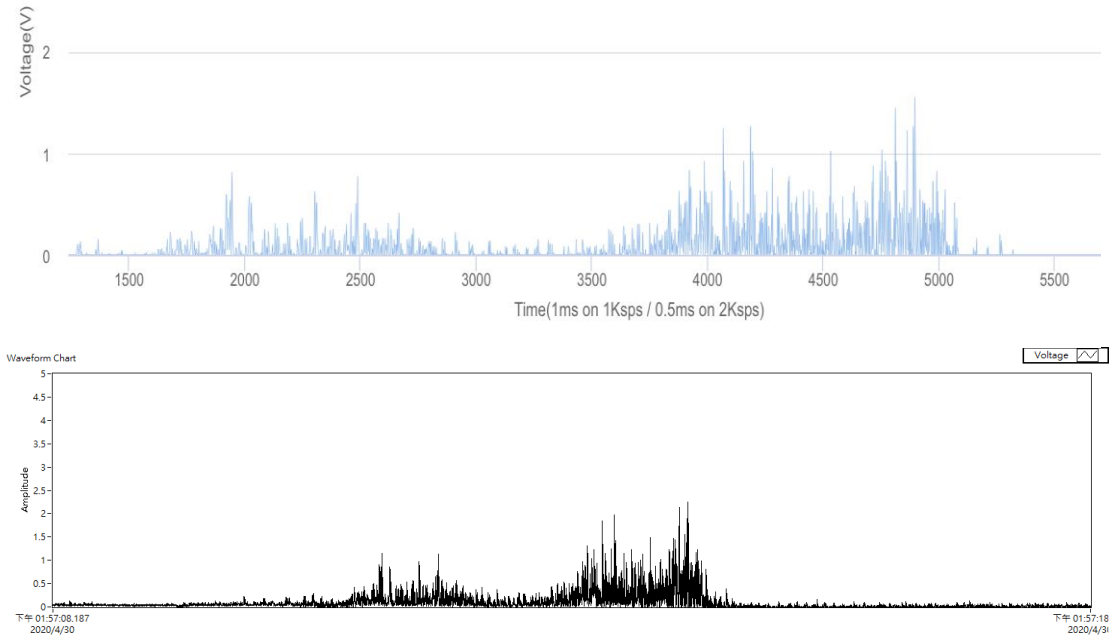


Figure 4. Comparison with LabView interface.

Implementation detail:

The microcontroller used in the project is ESP-32 from Espressif Systems. Software environment for MCU is Platform IO IDE with Arduino library in C++ programing language.

The MCU communicate with an 8-channel ADC integrated circuit with serial peripheral interface. The data receive during sampling period is saved to SRAM on the MCU with storage capacity of 520K bytes, once finish sampling, the data then being saved to flash memory with storage capacity of 4M bytes.

A web server is hosted by the MCU when it connects to a Wi-Fi access point. Static HTTP web serving is possible only when the MCU is not in sampling period or when the data is being saved to the flash memory. The files (e.g., index.html, myscript.js) being served to the client is stored on the flash memory. The webpage is written in html and JavaScript. The chart on the webpage is made possible with highcharts library.

The circuit is base on operational amplifier. The rail voltage is +-6V, and an active virtual ground provided by a LM358. The input signal is first fed to a instrumental amplifier with a signal gain of 250 (47.958dB). Then it is filter by a 2^{nd} order Butterworth lowpass filter in Sallen-Key topology with cutoff frequency 442Hz and passband gain approximately 1.58 for the quality factor to be Q=0.707. The filtered signal is then rectified and output to the analog to digital converter.

The device is power by an 18650 lithium-ion battery, which has a capacity of 3200 mAh and nominal voltage of 3.7V. The voltage is boost to 12V for the main circuit by a boost converter and then convert to 5V for the MCU with linear voltage regulator LM7805.

The device has a peak current of 300 mA and an idle current of 80 mA with Wi-Fi connection. Assuming the battery has 90% of its rated capacity, that is 9.6 hours for continuous sampling and 36 hours for idle.

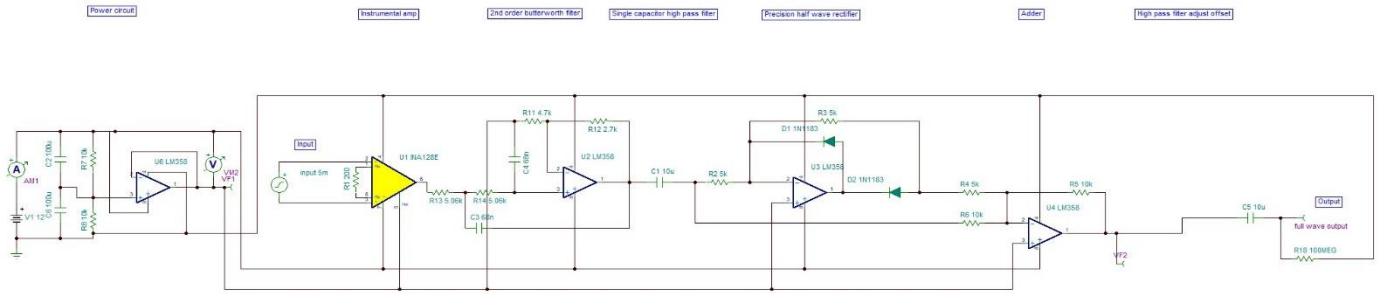


Figure 5. Amplifier and filter circuit.

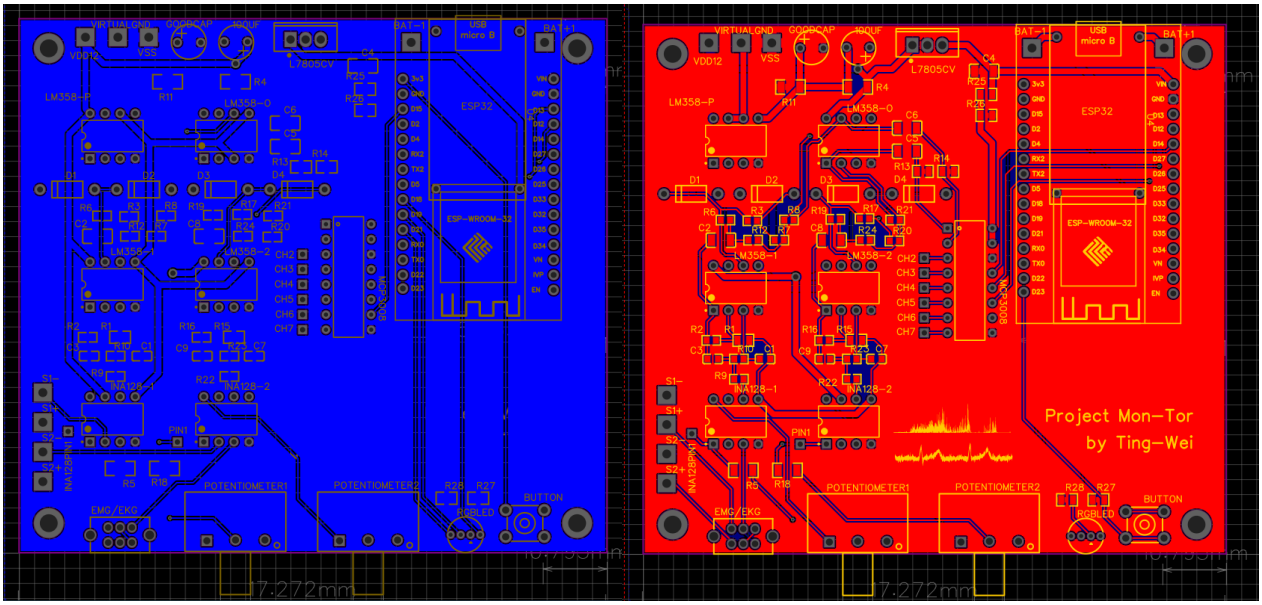


Figure 6. Two layers PCB layout. Bottom layer on the right with copper mark as blue, top layer on the left with copper mark as red and silk screen as yellow.

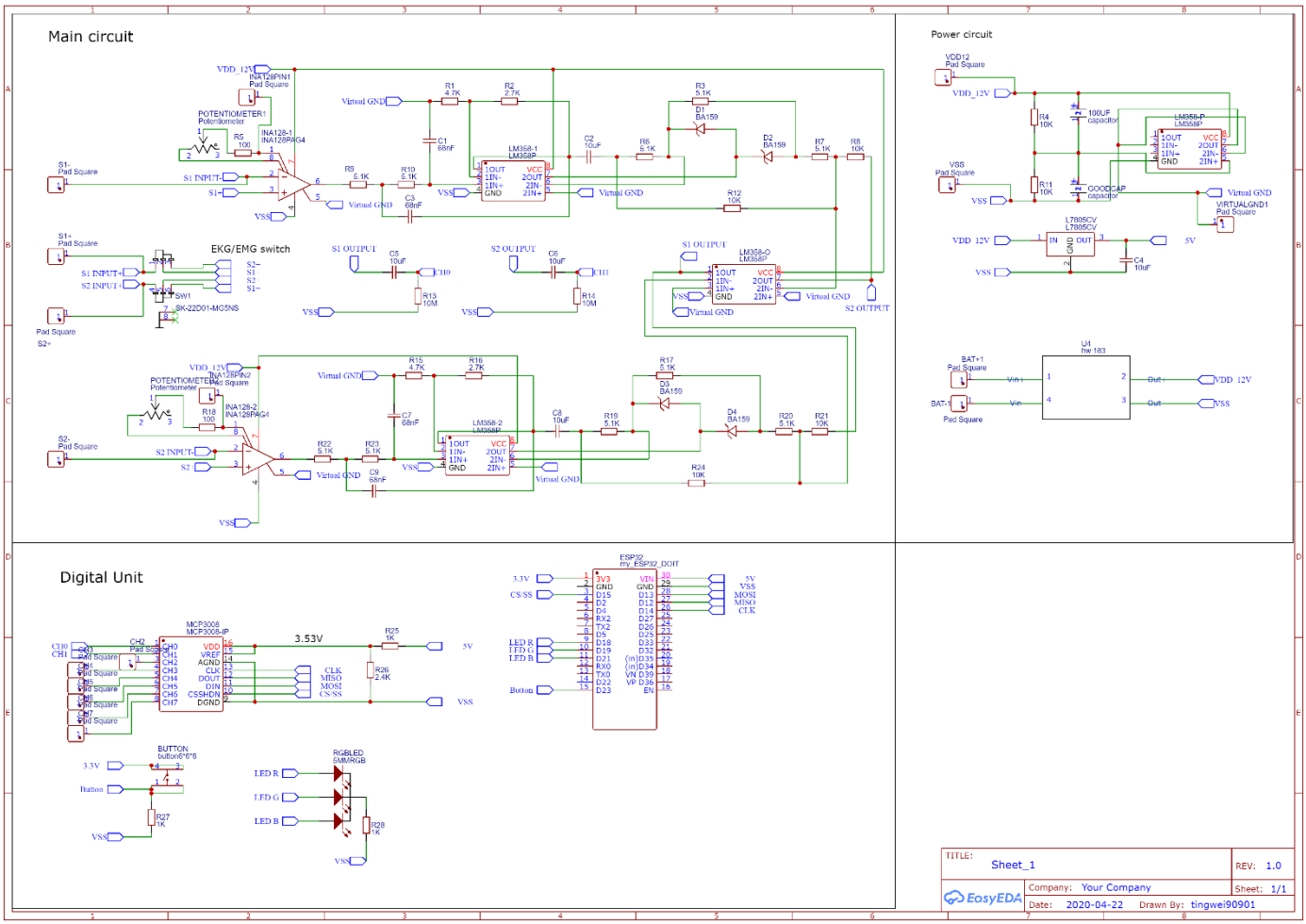


Figure 7. Schematic of the device.

Discussion:

The main design considerations are cost, accessibility and functionality, which specified in method section. The choice of MCU, power solution and analog circuit are all based on these considerations.

The alternative choices of MCU would be STM32F4 made by STMicroelectronics with ESP-01 Wi-Fi connection, unfortunately the development board is at least three times as expensive as ESP-32 development board. The combination of FPGA and ESP-01 is also a possibility if cost is not considered.

The battery choices considered other than lithium-ion is lithium polymer, which has the advantage of being smaller in size but higher price tag. A really good alternative choice, which I discover too late, would be lithium iron phosphate battery ($LiFePO_4$). It has a nominal voltage of 3.2V that can directly supply to the MCU without extra voltage regulation. $LiFePO_4$ is relatively safer compare to Li-ion and Li-PO batteries

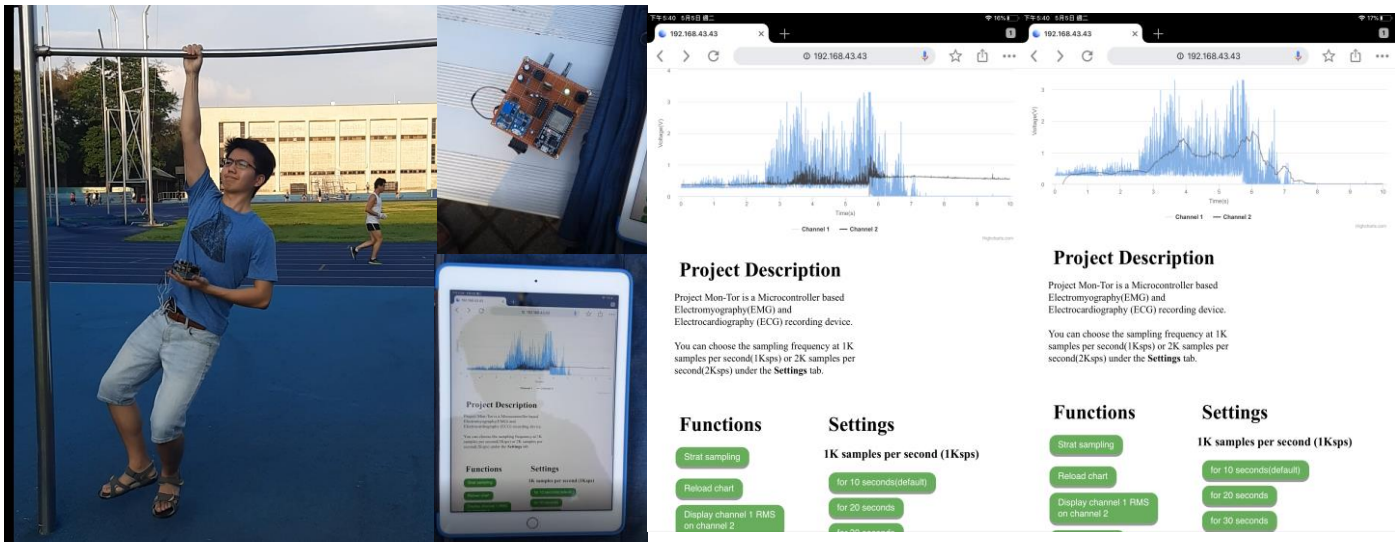


Figure 8. Holding device on one hand and use one arm scapular pull as demonstration since there's no housing yet. The electrodes are place on bicep long head (blue) and lower end of latissimus dorsi (black). The signal shift in the middle part is due to movement of reference electrode when dropping down from the bar.

and it's cheaper than Li-PO.

Analog circuit design is based on a university experimental course which is taught by professor Chao-Min Wu, who is the supervisor of this project. The external 10-bit ADC could be replaced with more accurate one if needed, or with less channel for exchange of less cost. MCP3008 is more accessible when I work on this project. In hindsight, I would not rectify the analog signal, but rather do it after data acquisition. The trade of is having less gain but more flexibility of what signal is sampled.

With limited time and knowledge, I did not achieve showing signal in real time on a webpage and more complex signal process at client-side browser like moving window fast Fourier transform.

Other than HTTP 1.1 version, I've spent an extensive amount of time trying WebSocket protocol. Although it is possible to receive message every 10 millisecond on the client side with each message contain 10 data points. I could not render these messages due to my limited knowledge of JavaScript.

The possible applications of this project are providing cheaper option for athletes to compare training method based on surface EMG, and if one gather enough data for specific movement it's possible to do machine learning with the dataset.

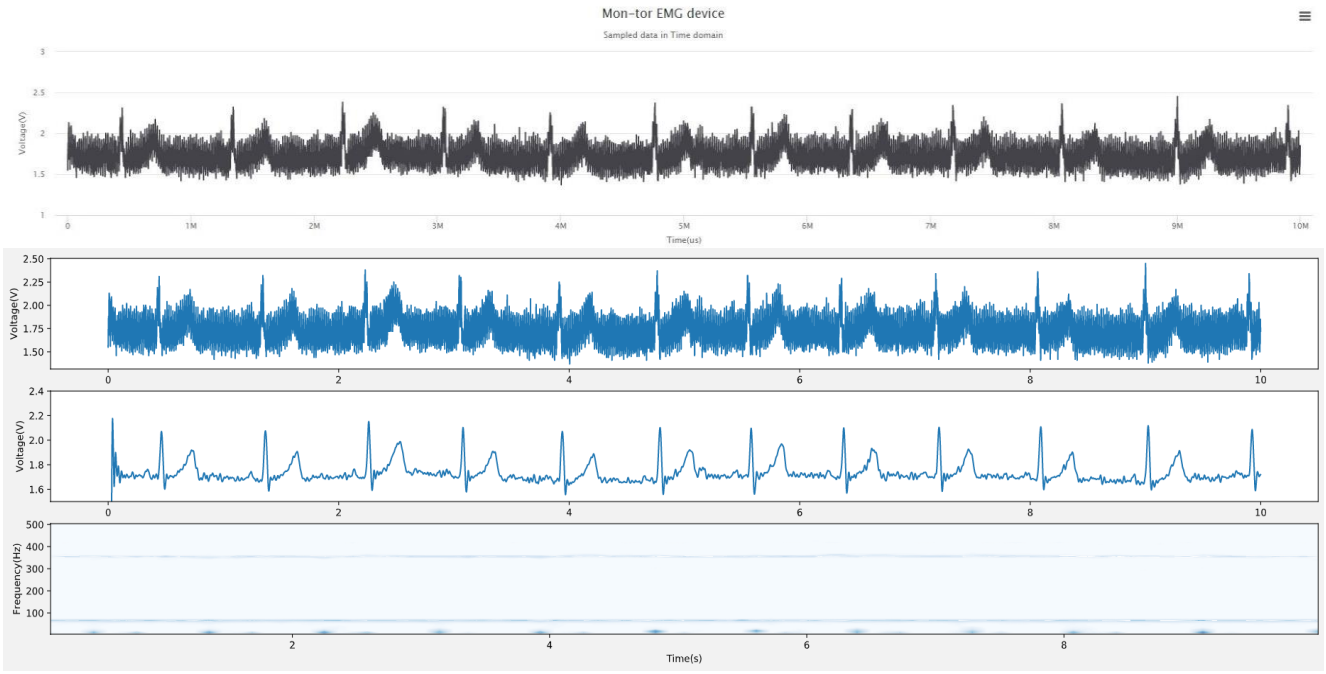


Figure 9. Lead 1 EKG, using python requests library download data from the server and process signal. The first graph from the top is the raw signal with webpage interface. The second graph is the raw signal drawn with python. The third is applying low pass filter to the signal. The fourth is the spectrogram of the signal.

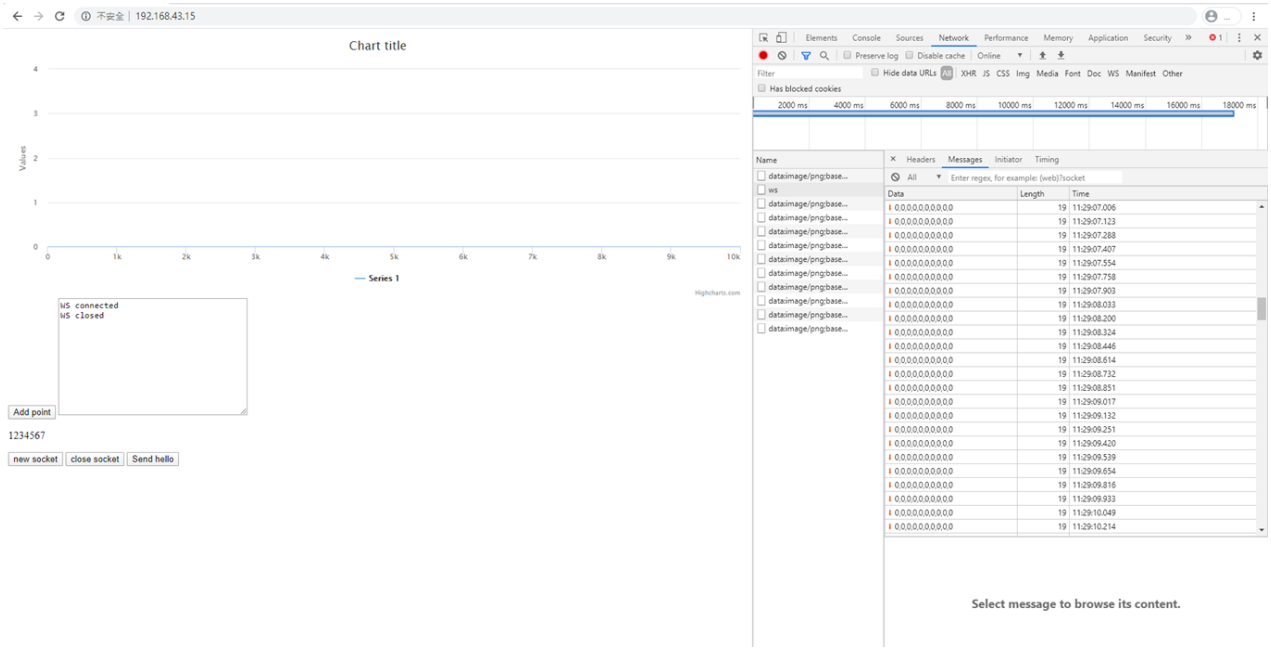


Figure 10. Testing WebSocket protocol. It's possible to stream data in real time, but unfortunately, I was not able to render the chart in real time

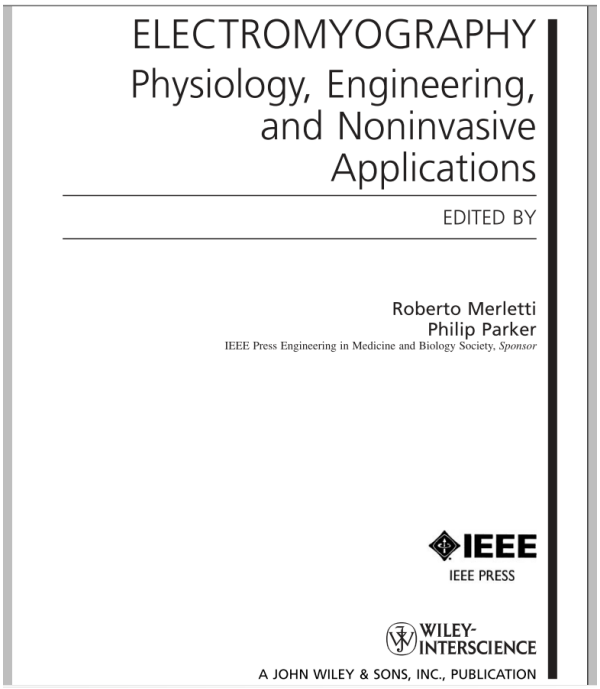


Figure 11. Electromyography: Physiology, Engineering, and Non-Invasive Applications. Available with NCU library login to IEEE website.

中文版

Abstract:

這個專題做出一個可以量測兩組表面肌電圖和心電圖的行動裝置。使用電極貼片，在肌腱和肌肉交界處記錄接收到的肌電類比訊號，經類比電路放大和濾波後，由 10 bit 類比轉數位器轉換成數位訊號，透過序列周邊介面（SPI）傳送到微控制器（Microcontroller）ESP-32。當作一個 Wi-Fi 工作站（STA 模式），該微控制器可以將訊號彙整成逗點分隔檔案，將檔案傳到網頁客戶端。網頁客戶端會顯示紀錄的肌電圖以及一些其他選項，包含下載逗點分隔檔案、設定紀錄時間（雙通道最長三十秒）、設定取樣頻率（1K/2K 取樣每秒）、計算肌電圖的方均根（RMS）等等。裝置是由 18650 鋰離子電池供電，連續使用時間約為五小時，待機時間約十二小時。

Introduction:

每當阻力訓練的運動員在討論哪一種訓練動作最適合增加肌肉量或是最大肌力時，肌肉啟動程度（Muscle activation）經常是討論的主題之一。哪種動作最有感覺、全神貫注於要訓練的肌群可以增加肌肉啟動程度等等建議，在業餘阻力訓練圈並不少見，惟這些較主觀或根據個人經驗的的敘述難以被客觀測量及記錄。

這個專題的目的在於做出一個可以記錄和測量主動表面肌電圖的行動裝置，方

便運動員量化自己的訓練成果。
該裝置會使用電極貼片，在肌腱和肌肉交界處記錄接收到的肌電類比訊號，經放大和濾波後，轉換成數位訊號，可以由微控制器傳送至網路裡的其他裝置。將微控制器當作一個網頁伺服器（Web server），網路裡的其他裝置可以藉由 HTTP 和微控制器互相溝通，觀察記錄到的表面肌電圖和下載該紀錄檔。

Method:

根據專題目的，我先擬定了幾個主要目標：能夠測量肌電圖、取樣頻率至少 1K 每秒、能夠以電池供電和能夠傳送到網頁伺服器客戶端；以及次要目標：能夠即時傳送肌電圖、能夠多通道測量肌電圖、畫 PCB 並請人打樣、能夠在網頁實現一些基本的數位訊號處理包含 RMS 以及快速傅立葉轉換（FFT）的頻譜圖（spectrogram）。
專題主要分成三個階段，第一階段為資料蒐集，包含量如何測肌電圖的方法以及肌電圖的訊息意義、學習用 C 語言操作微控制器、了解基礎 HTML 和 JavaScript 來建立使用者介面。第二階段為原型測試（prototype），使用電腦軟體設計以及模擬類比電路，並在麵包版上測試、撰寫微控制器的程式以及網頁的使用者介面。第三階段為實體設計以及測試次要目標。

Result:

這個專題達成設定的主要目標，包含能夠測量肌電圖、取樣頻率至少 1K 每秒和能夠以電池供電和能夠傳送到網頁伺服器客戶端。以及部分次要目標：能夠多通道測量肌電圖、畫 PCB 並請人打樣、能夠在網頁實現肌電圖的 RMS。
透過 INA128 儀器放大器、整流器和 5 – 400Hz 的濾波器，這個裝置能夠測量和紀錄原始訊號大小約為 5mV - 10mV p-p 的肌電訊號和心電訊號（電極貼片位置會影響測量導程）。
使用樂鑫資訊科技（Espressif Systems）的 ESP32 微控制器，在 timer interrupt 發生時，微控制器會和擁有八通道的外部 ADC 使用序列周邊介面（SPI）接收訊號。這些資料會被存到 SRAM 中，直到紀錄時間結束後再存到 Flash 記憶體中，等待下一次紀錄或是根據 HTTP request 將其發送到客戶端。
控制裝置的使用者介面是微控制器架設的網頁伺服器，在區域網路內的其他裝置可以透過網頁瀏覽器觀看紀錄下來的訊號以及調整裝置設定，也可以下載到電腦做更進階的數位訊號處理。
若周圍沒有 Wi-Fi 無線接入點（AP）可以連接，則可以用按鈕和 LED 和裝置溝通。網頁是使用簡易的 html 和 JavaScript，繪圖是使用 highcharts 的 JavaScript library。

Discussion:

裝置設計上最主要的考量是取樣頻率要至少一千取樣每秒（1Ksps）、原始訊號的完整度以及裝置成本，包括微控制器、電源的解決方案和裝置中類比 IC 的選擇都是根據上述考量決定的。
微控制器除了這次使用的 ESP32 之外有考慮過：ESP-12，但 SRAM 太小不符合應用要求；Arm 架構的 STM32F4 搭配 ESP-01，惟價格和 ESP32 有一段差距；FPGA 搭配 ESP-01，同樣是價格過高（但大規模製作使用 FPGA 測試再客制化特殊應用積體電路 ASIC 材料成本會是最低的）。電源的解決方案除了這次使用的 18650 Li-ion 之外還有考慮過體積較小的 Li-po 電池，但是價格高一到兩倍左右。類比 IC 的選擇，包括儀器放大器 INA128、運算放大器 LM358 和 ADC MCP3008，是根據取得較容易和符合本次專題需求；ADC 根據需求可以換成較便宜、通道較少的 IC，或較昂貴、取樣位元數較高的 IC。
本次專題的次要目標，在有限的時間和知識下，沒有辦法達到在網頁中即時的紀錄和顯示訊號以及對訊號做 FFT 並繪製頻譜圖。
除了 HTTP /1.1 當作應用層協定之外，我有嘗試過較精簡、可以全雙工通訊的 WebSocket 協定。在 WebSocket 測試中裝置有辦法每 10ms 傳送資料到客戶端，但基於對 JavaScript 的知識不足，沒有辦法即時 render 這些資料。
此專題的應用面包括：方便紀錄阻力訓練時肌肉的表面 EMG，並提供可以客觀比較訓練方法的手段；如果擁有足夠多組同一動作相關肌群的表面 EMG，研究人員可以將資料使用深度神經網路分析並擷取特徵。

References:

[1] Electromyography: Physiology, Engineering, and Non-Invasive Applications. By Roberto Merletti, Philip J. Parker. ISBN: 978-0-471-67580-8
[2] For HTML and JavaScripts template: https://www.w3schools.com/
[3] For highcharts library and API: https://www.highcharts.com/

Appendix

Tools needed for this project:

Hardware:

Instrumental amplifier INA128

Multiple LM358 OP amp

Multiple 100, 1K, 2.4K, 4.7K, 5.1K, 10K, 10M ohm resistors

Multiple 68nf, 10uf, 100uf capacitors

External ADC MCP3008

Dupont wires

18650 Lithium ion battery

Electrodes

ESP32

- Operate at 3.3-3.6V
- Input Voltage: 6-20 V, 7-12V recommended
- Peak current draw on start-up: 320mA
- Normal operation with AP connected: 35mA average, 300mA peak
- One 10 bit ADC, 0~3.3V on nodeMCU.

ELVIS II for comparison

Software:

VScode and platformIO for programming

TINA-TI for simulation

python for flashing microcontroller

LabView for comparison

Considerations:

The choices of hardware and software above is made based on price, accessibility and functionality. Learn more on the discussion part for more detail.

Design flow:

1. Obtain background knowledge – 2019/9-11
   Including general idea of how to obtain bio signals, muscle physiology, web server, html markup language, esp8266.
2. Conduct experiments – 2019/11-2020/1
   Including experiments design and how to approach this problem.
3. Building prototypes – 2020/2
4. Debug and assembly – 2020/2-4

Background knowledge:

1. EMG – electromyography
   EMG is a technique for evaluating and recording electrical activity created by skeletal muscle. Electric potential is generated by the skeletal muscle fiber membrane.
2. Motor unit

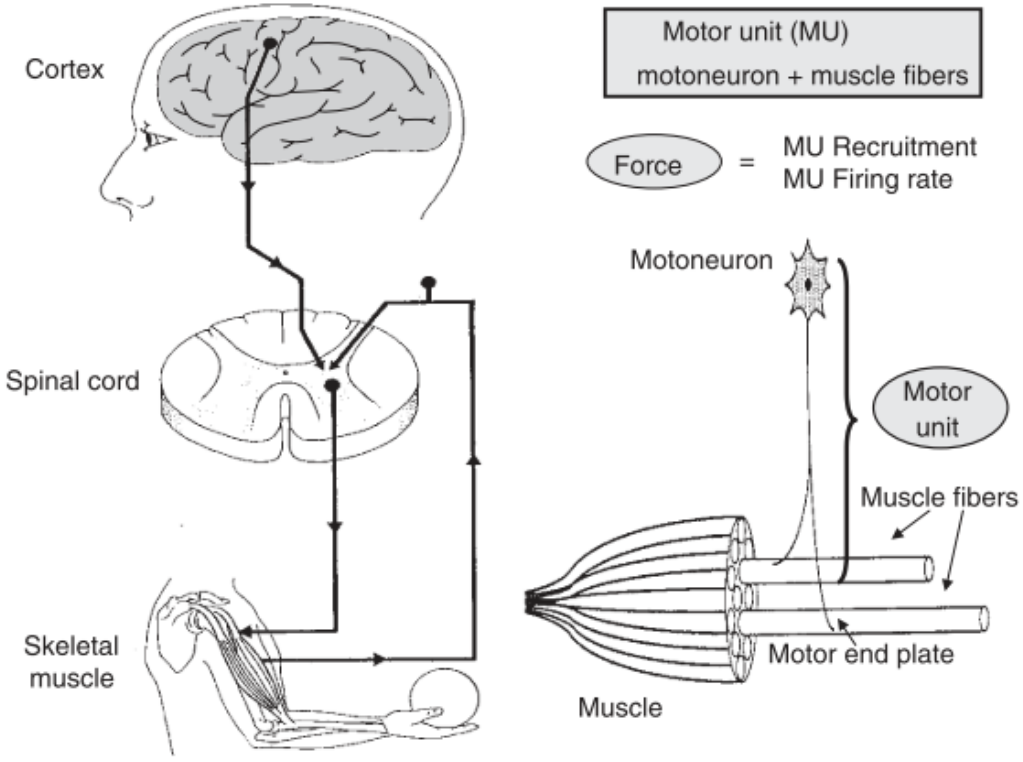


Fig 1. A schematic representation of basic motor control mechanisms and of the motor unit and its components.

A motor unit (MU) consists of an α-motoneuron in the spinal cord and the muscle fibers it innervates. The net membrane current induced in this motoneuron by the various synaptic innervation sites determines the discharge (firing) pattern of the motor unit and thus the activity of the MU. The number of MUs per muscle in humans may range from about 100 for a small hand muscle to 1000 or more for large limb muscles and the difference in force generated varies greatly from muscle to muscle.

3. Motor Unit Recruitment and Firing Frequency

The greater the number of MUs recruited and their discharge frequency, the greater the force will be. It is well documented that motor unit recruitment and firing

frequency (rate coding) depend primarily on the level of force and the speed of contraction. Thus a direct relationship between the electromyogram (EMG) and exerted force might be expected.

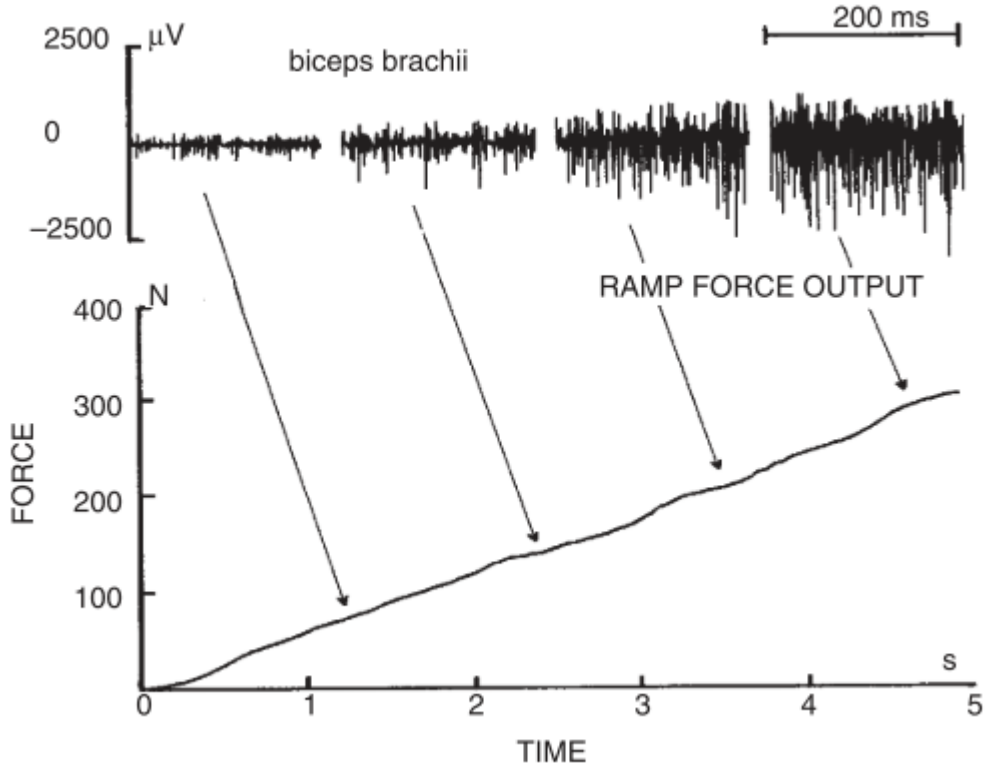


Fig 2. Intramuscular spike recordings obtained from the biceps brachii muscle during linearly force-varying isometric muscle contraction.

4. ESP8266

The ESP8266 is a System on a Chip (SoC), manufactured by the Chinese company Espressif. It consists of a Tensilica L106 32-bit micro controller unit (MCU) and a Wi-Fi transceiver. It has 11 GPIO pins (General Purpose Input/Output pins), and 1 analog input. One can program it like any normal Arduino or other microcontroller.

There are many different modules available, standalone modules like the ESP-series by AI Thinker, or complete development boards like the NodeMCU DevKit or the WeMos D1.

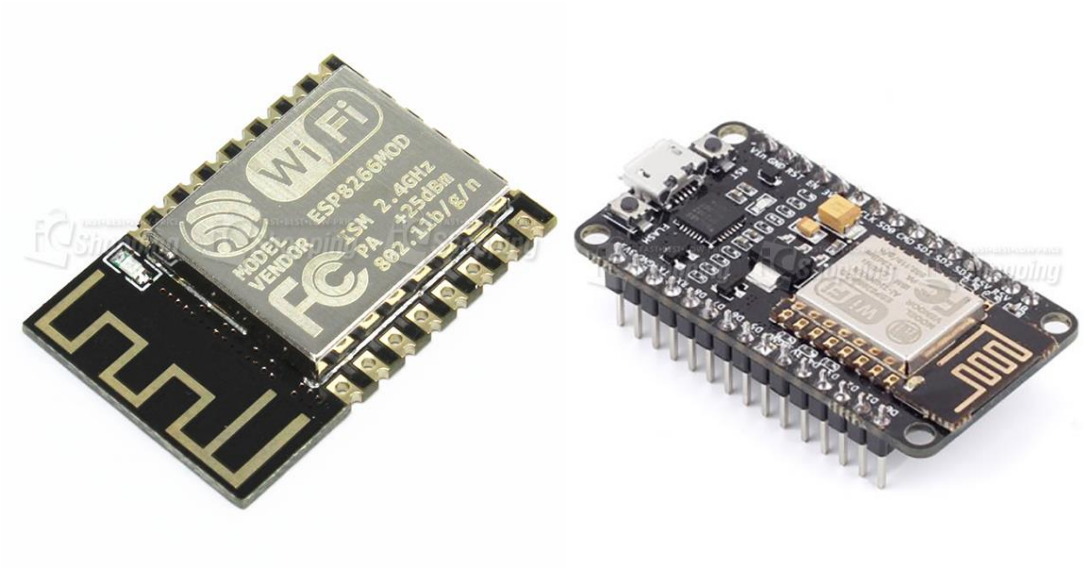


Fig 3. ESP8266 Wi-Fi modules, nodeMCU(right side) is the development board.

5. Network protocols

The TCP/IP stack consists layers of protocals.

Layer	Protocols
Application	HTTP, FTP, mDNS, WebSocket, OSC ...
Transport	TCP, UDP
Internet	IP
Link	Ethernet, Wi-Fi ...

The link layer is the physical link between devices, in the case of the ESP8266, it is a Wi-Fi connection.

The Internet or Network layer uses IP addresses to know where it should send the data. This means that two devices can now send packets of data to each other

The Transport layer makes data transmission more reliable by adding error checking, resending mechanism etc.

The Application layer enable both ends of the internet can understand the data sent through the internet.

Experiments:

1. Using ESP8266 as a digital signal receiver
   The sampling frequency required to avoid signal overlap is anything above 1.2 kHz, since the bandwidth of EMG is between 5 ~ 500 Hz generally speaking. The input amplitude of the ADC is 0-3.3V according to datasheet, sampling frequency is 10 kHz while running a very simple code.

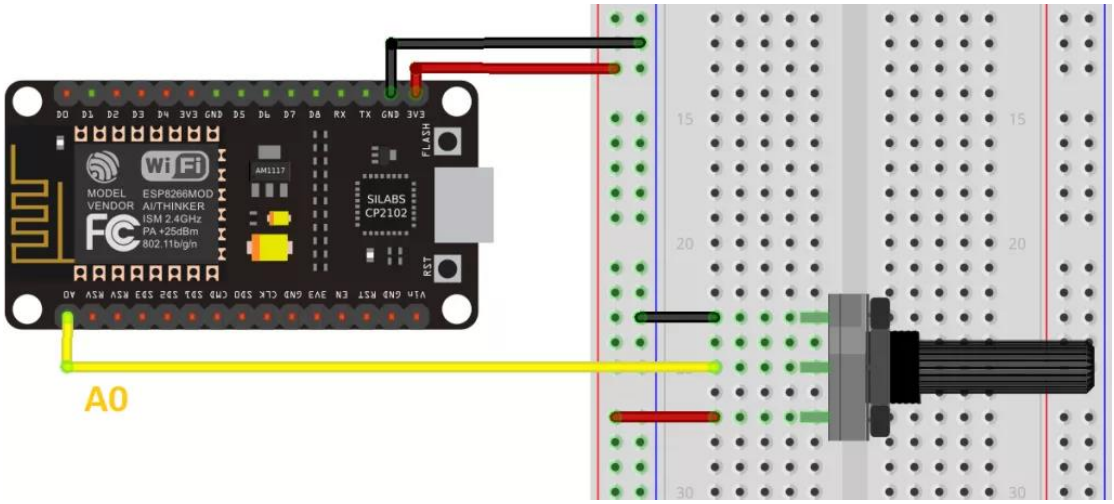


Fig 4. Schematic diagram for ESP8266 as a digital signal receiver.

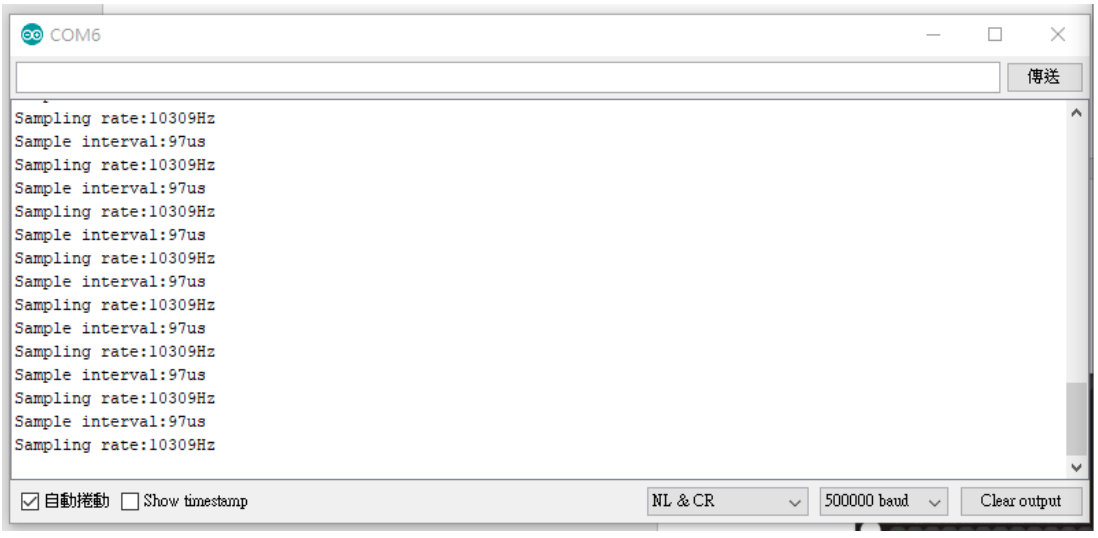


Fig 5. Testing sampling rate. Fs=10309Hz

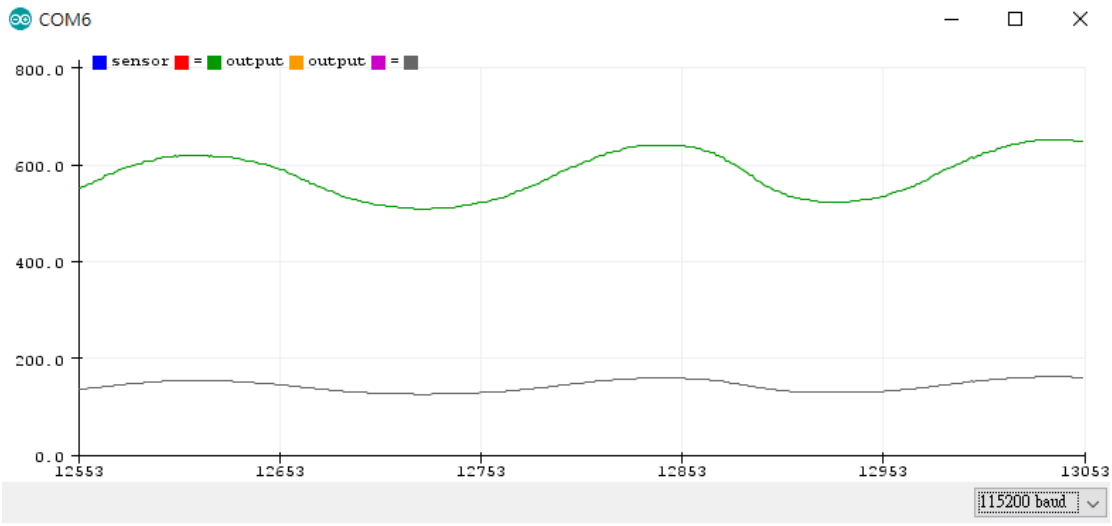


Fig 6. Graphical output from the potentiometer.

2. Open a webserver with esp8266

   Build a simple webserver that can send orders from client side, and send voltage value of ADC pin input from server side to client side in real time.

Fig 7. Controlling LED on and off from client side.

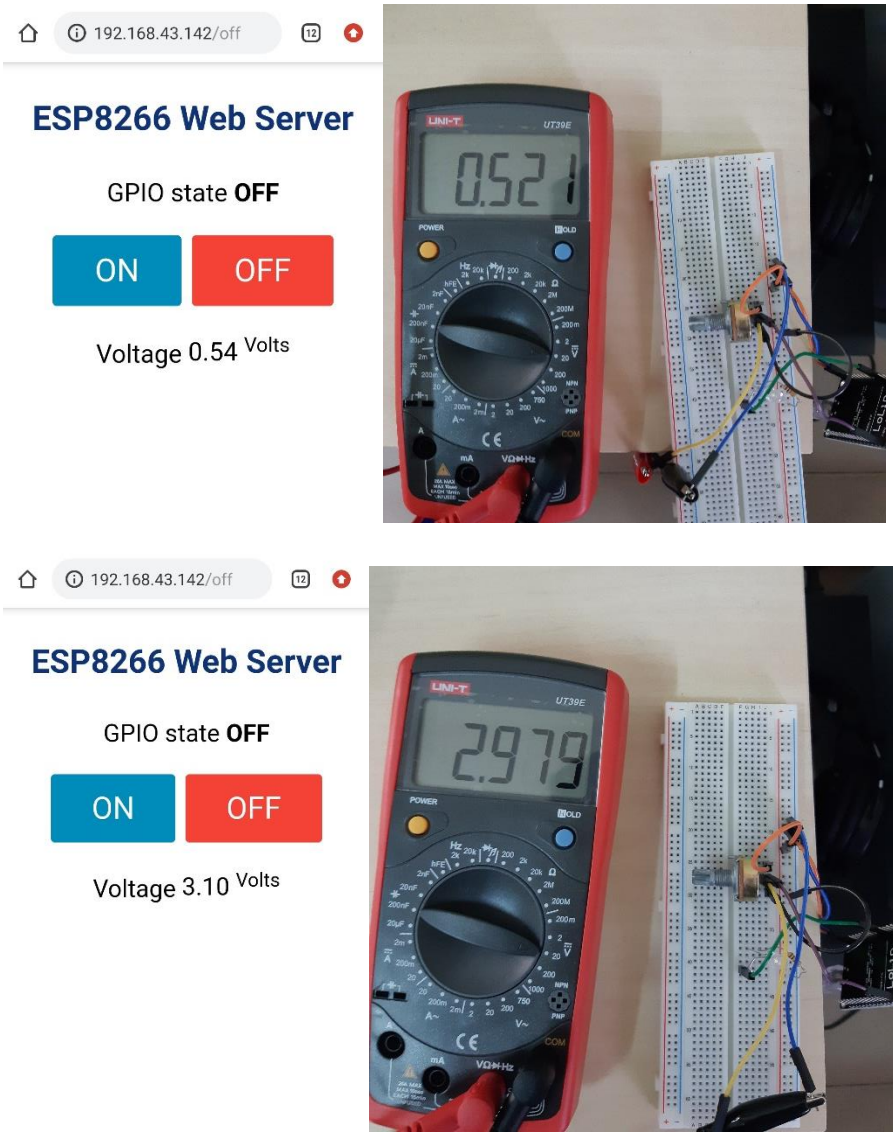


Fig 8. Receiving voltage value of ADC pin from server side, error is significant when voltage is above 3V.

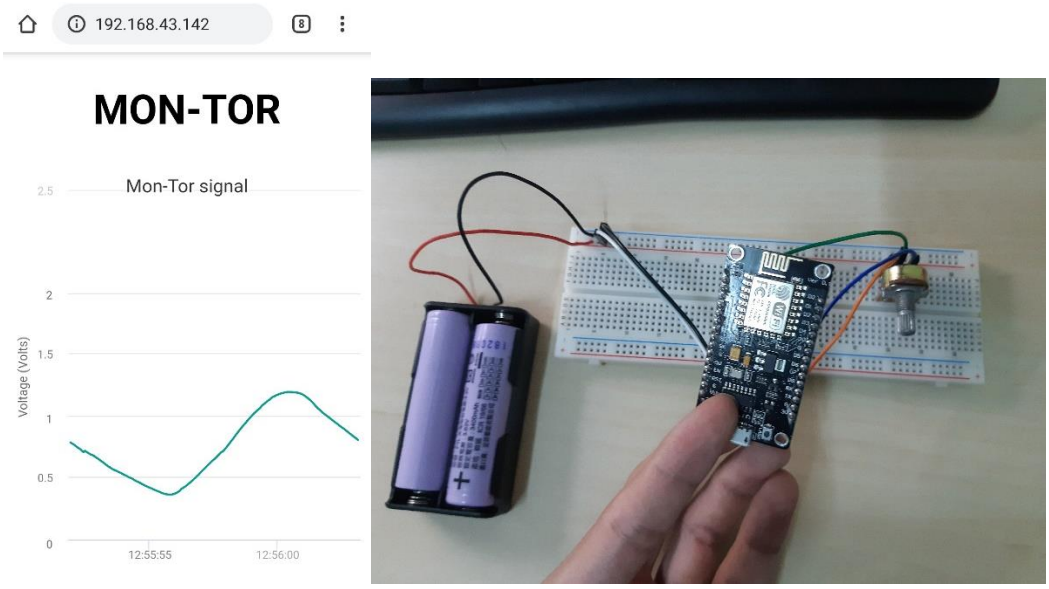


Fig 9. Web server with graphic output, and double cell 18650 lithium ion battery as power source for the microcontroller. The library of this plotting tool is Highchart JavaScript.

3. Design a circuit that should be able to detect EMG on simulation software (TINA-TI).

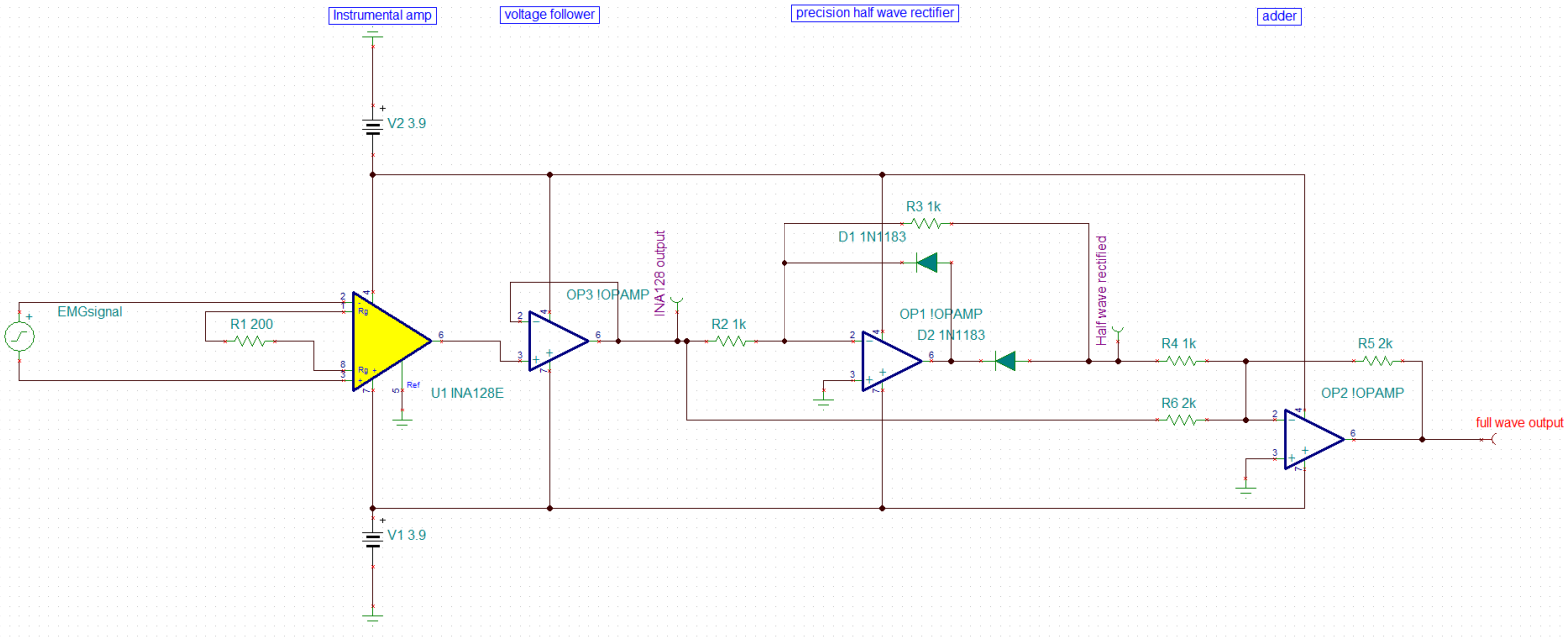


Fig 10. Simulation circuit diagram.

This design uses instrumental amplifier INA128 from Texas Instrument because its relatively low price and low power consumption. Follow by a voltage follower to avoid loading effect. After that is a precision full wave rectifier, which consists of a

half wave precision rectifier and a summing amplifier. Here I'm using an active rectifier instead of a passive rectifier is due to the voltage drop between a diode is significant compared to the signal. This way of configuration ideally allows signal to be amplified to between +3.3V to -3.3V, and rectified to 0 to 3.3V under the rated ADC input of nodeMCU development board.

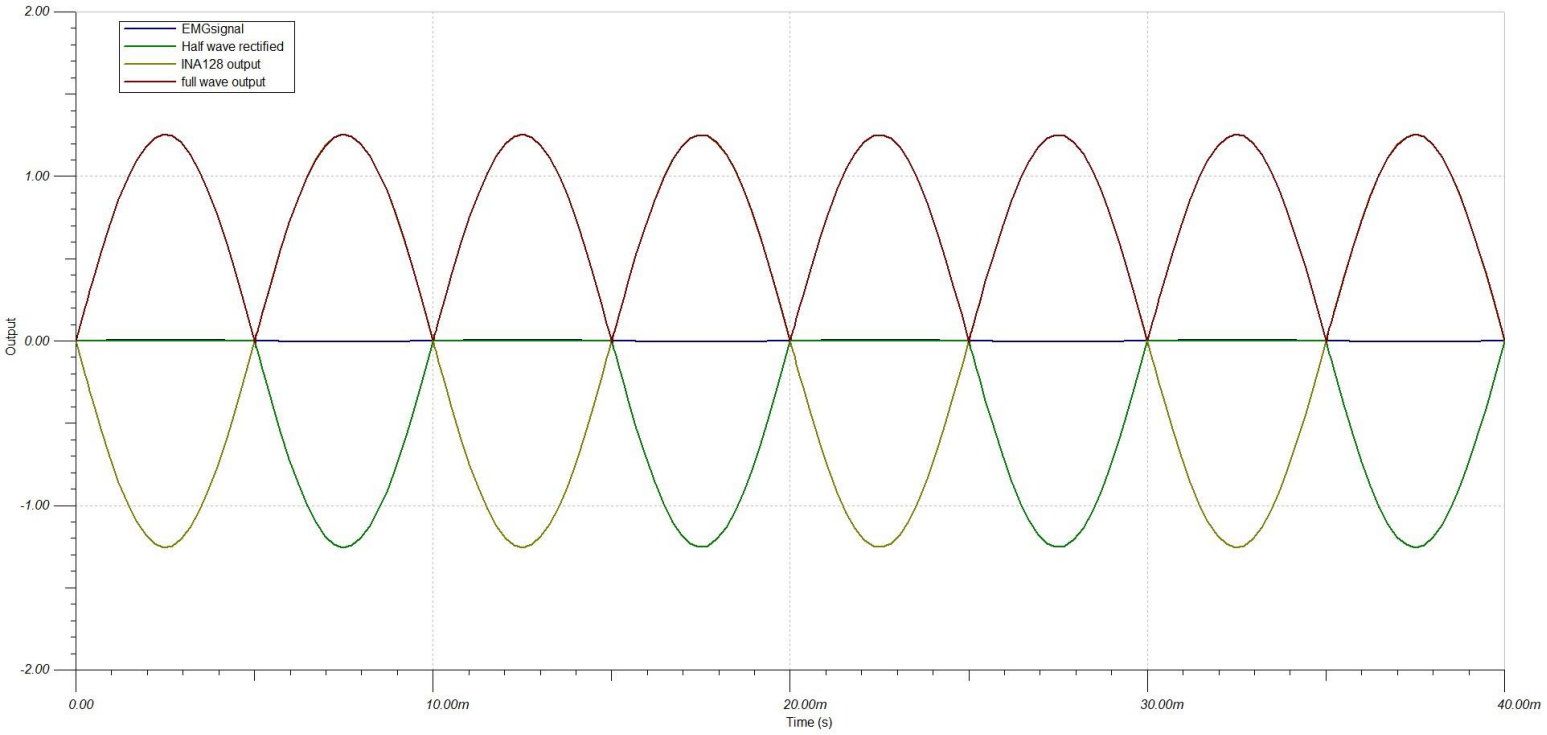
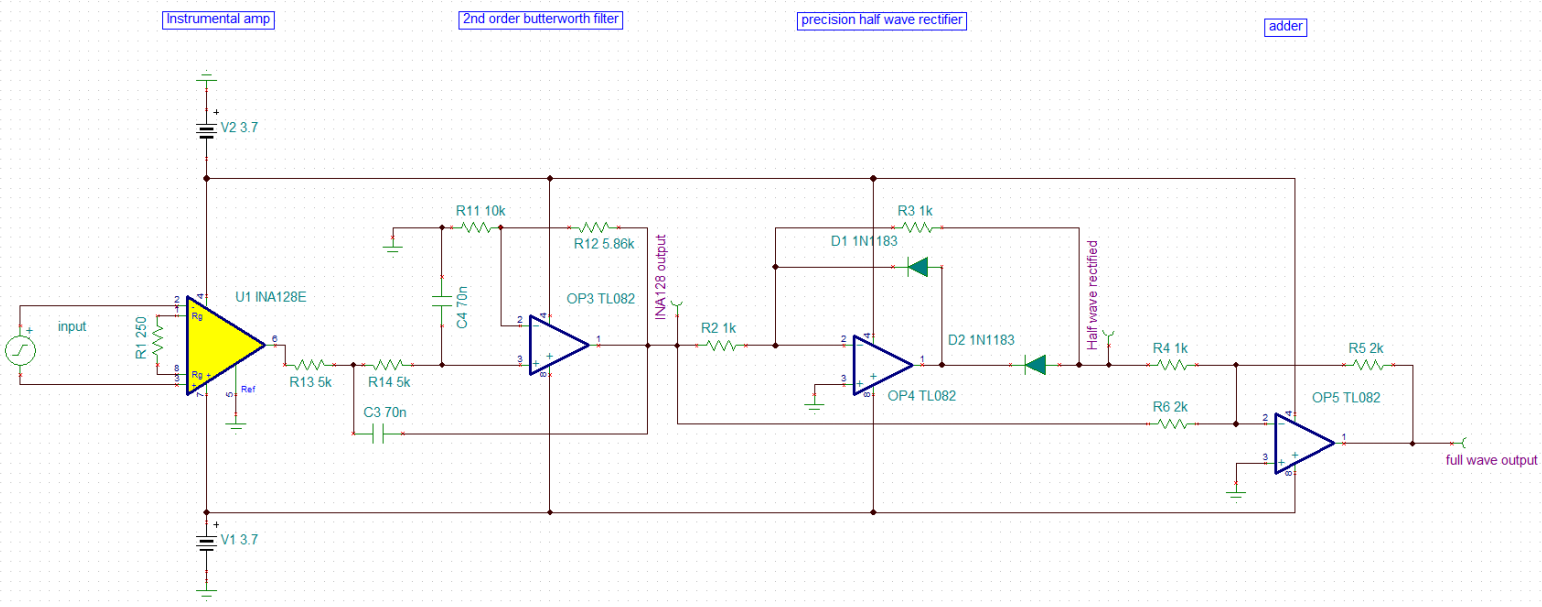


Fig 11. Simulated waveform with input 5mV, 100Hz sine wave. Output rectified sine wave with 1.25V amplitude, gain is 250, 47.958dB.

Second iteration, I soon realized that without an analog filter, the bandwidth of



my signal would be too wild consider the relatively low sampling frequency. The signals would overlap according to sampling theory. So, I replace the voltage follower with a 2nd order Butterworth filter with cutoff frequency of ~450Hz, as shown below.

Fig 12. Added LP filter into the circuit.

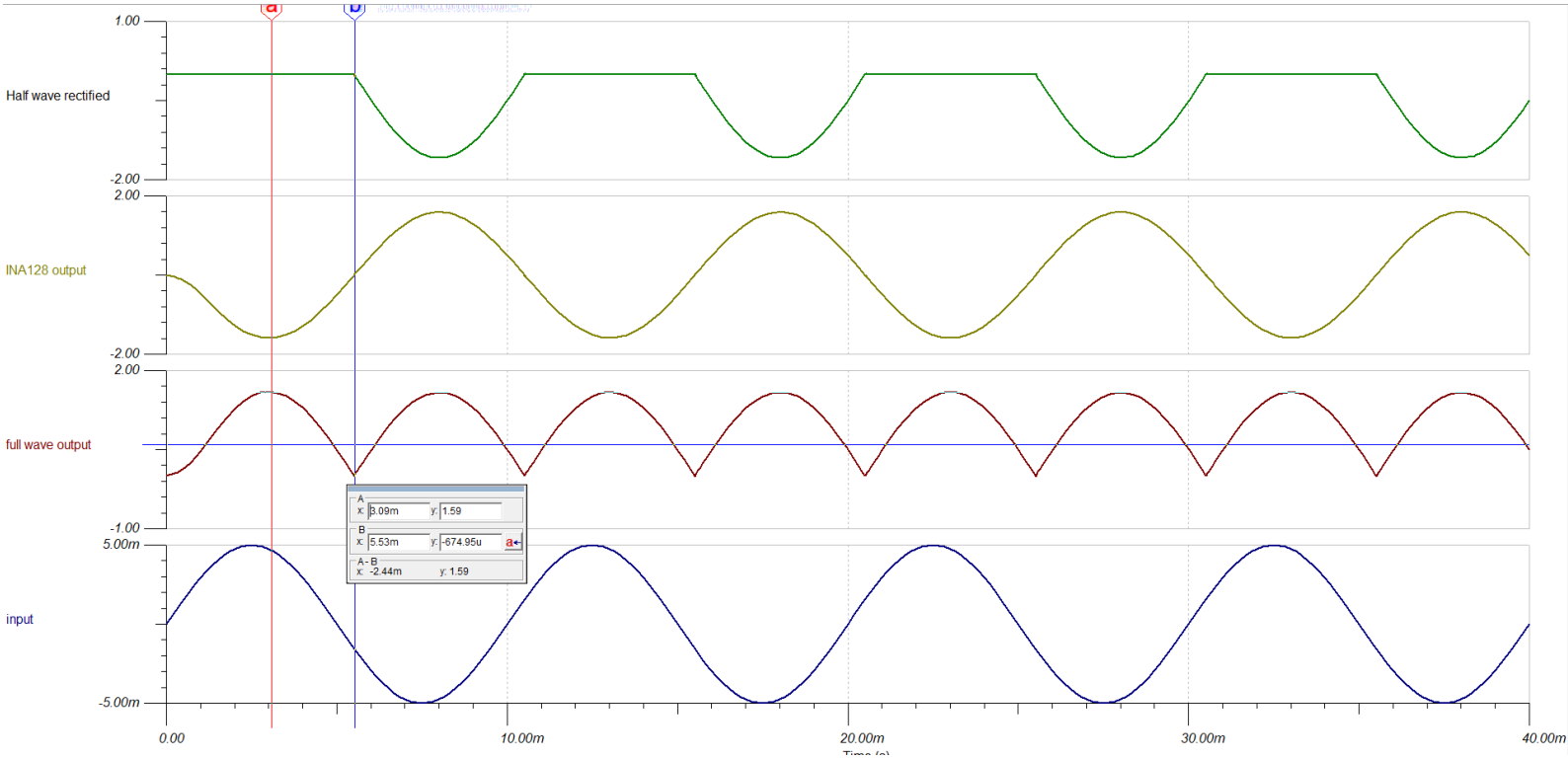


Fig 13. Simulated waveform with input 5mV, 100Hz sine wave. Output rectified sine wave with 1.59V amplitude, gain is 318, 50.04dB.

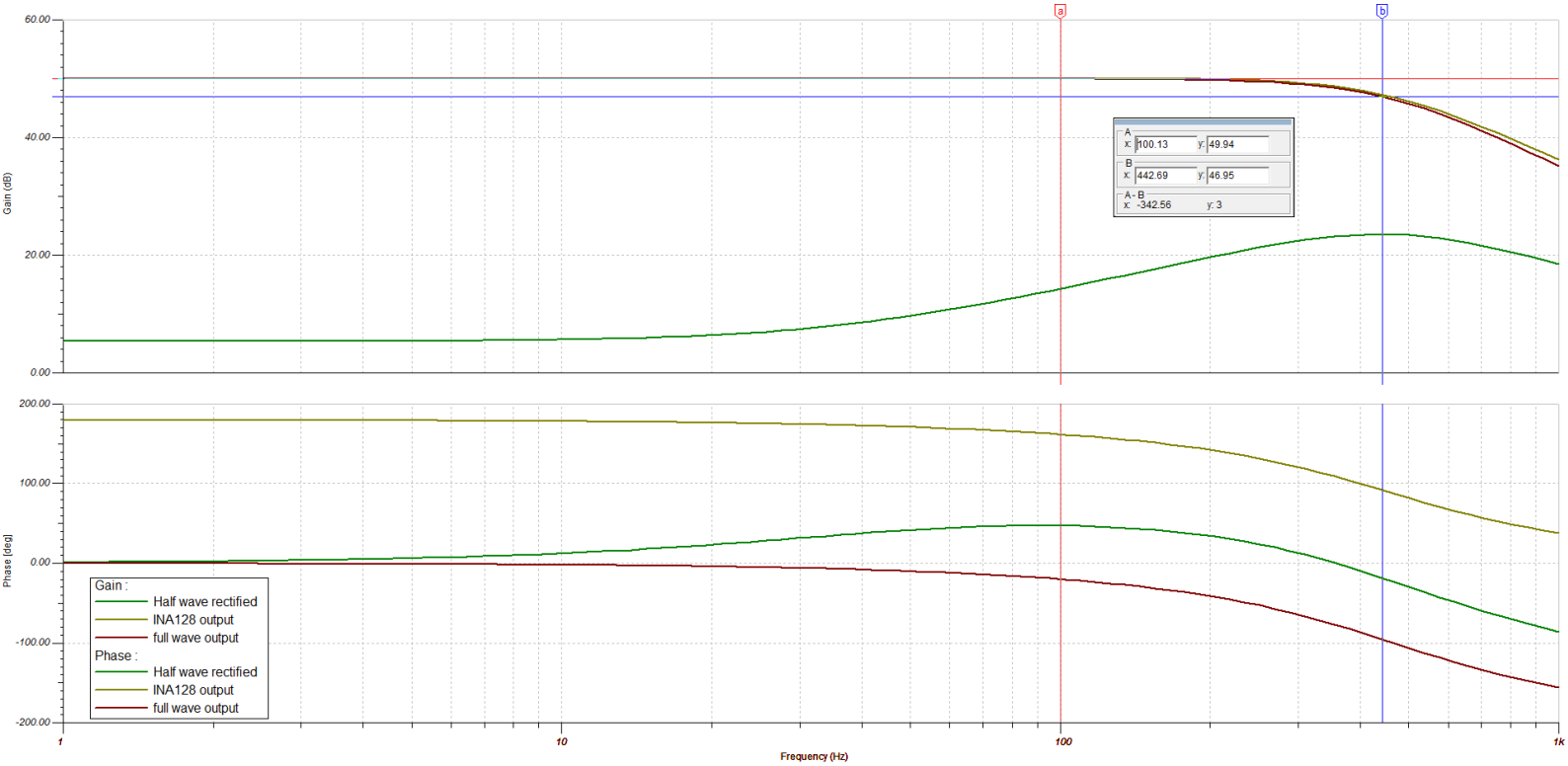


Fig 14. Bode plot. Simulated gain is 49.94dB, cutoff frequency is 442Hz. (ideal $w_c = 2*\pi*f_c = 1/(5k*70n)$)

4. Build the circuit on the bread board and test the ability of the circuit

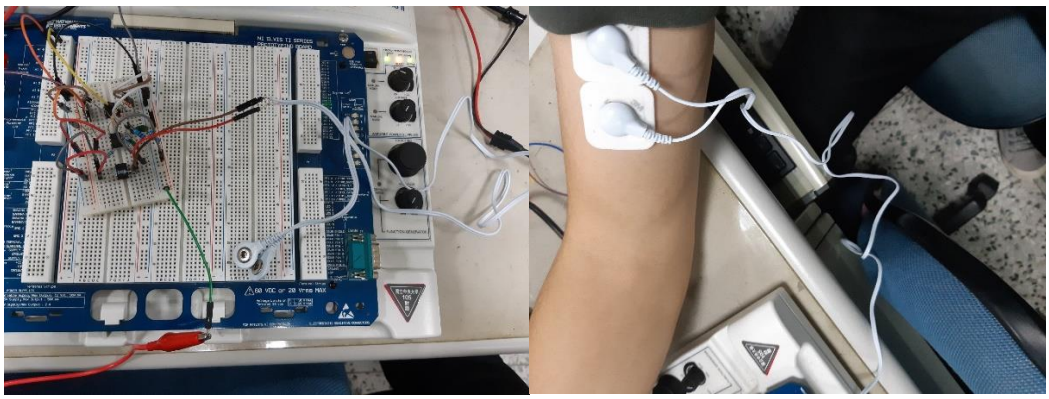


Fig 15. The experiment setup on bread board.

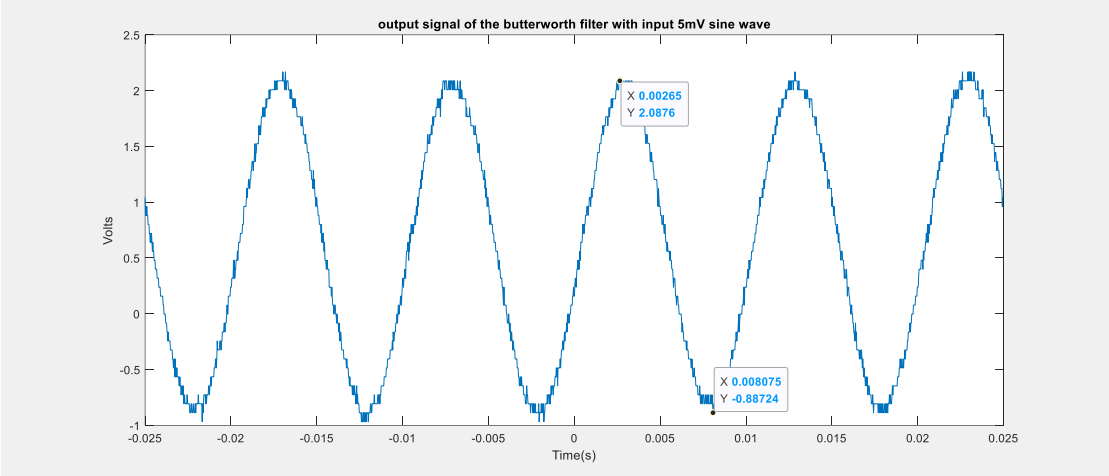


Fig 16. Output signal after the Butterworth filter with input 5mV sine wave, unrectified. Gain after filter is 2.974/0.005 = 594.9, 55.48dB.

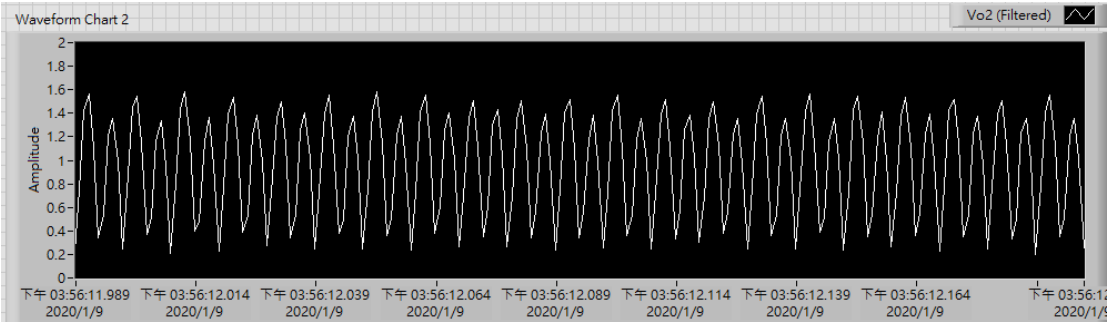


Fig 17. Output signal of the rectified signal with input 5mV sine wave, the gain is approximately 280, 48.9dB.

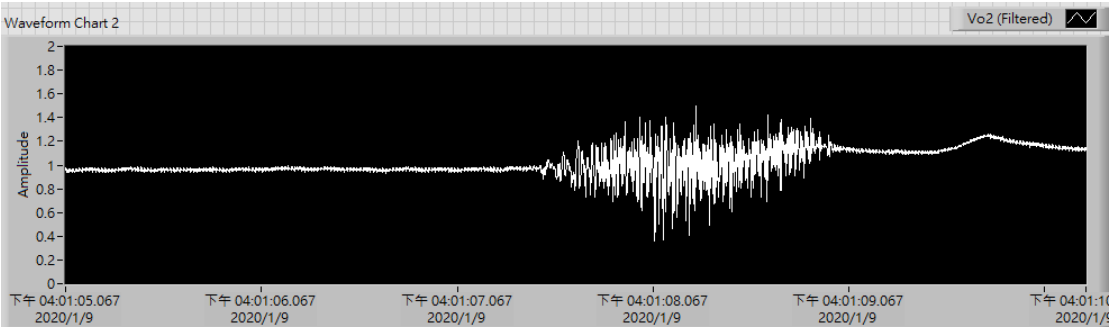


Fig 18. Measurement of bicep zero load contraction EMG. Peak to peak is approximately 1V. Note that the signal is unrectified due to the absent of the ability to adjust offset voltage in the circuit design.

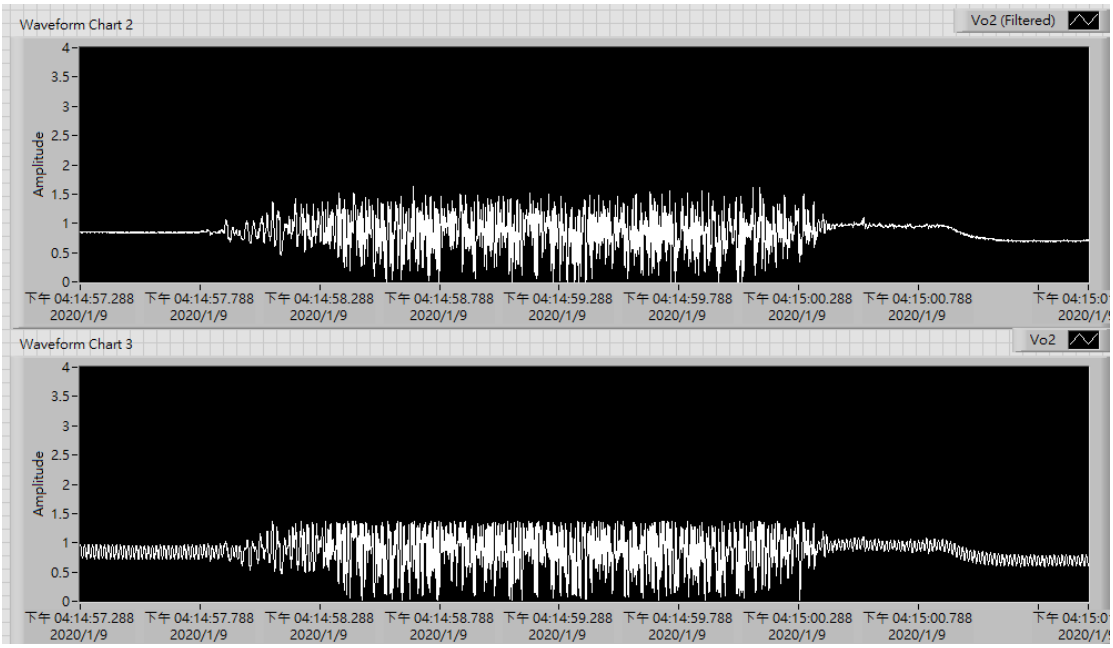


Fig 19. Bicep EMG when the VCC is set to Lithium Ion battery cell voltage 4V with lab bench power supply. Bottom figure is without digital band rejecting filter at 60 Hz, the small perturbation present in the signal is 60Hz noise, which in theory be reduced when using only battery DC supply.

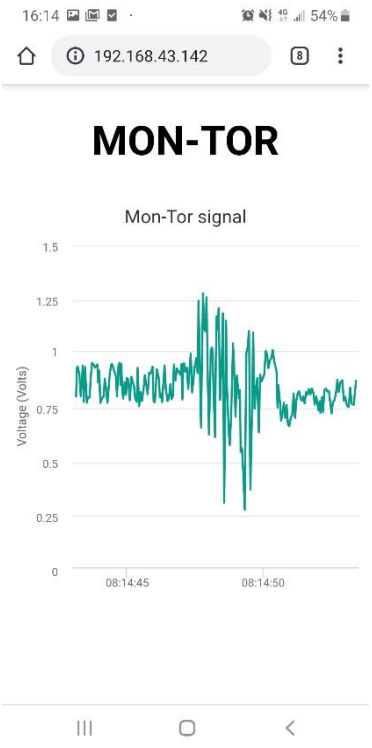


Fig 20. Bicep EMG in Fig 19. is sent to the web server in real time. The web file still needs to be tweak, as the y-axis is auto adjusting.

Prototypes:

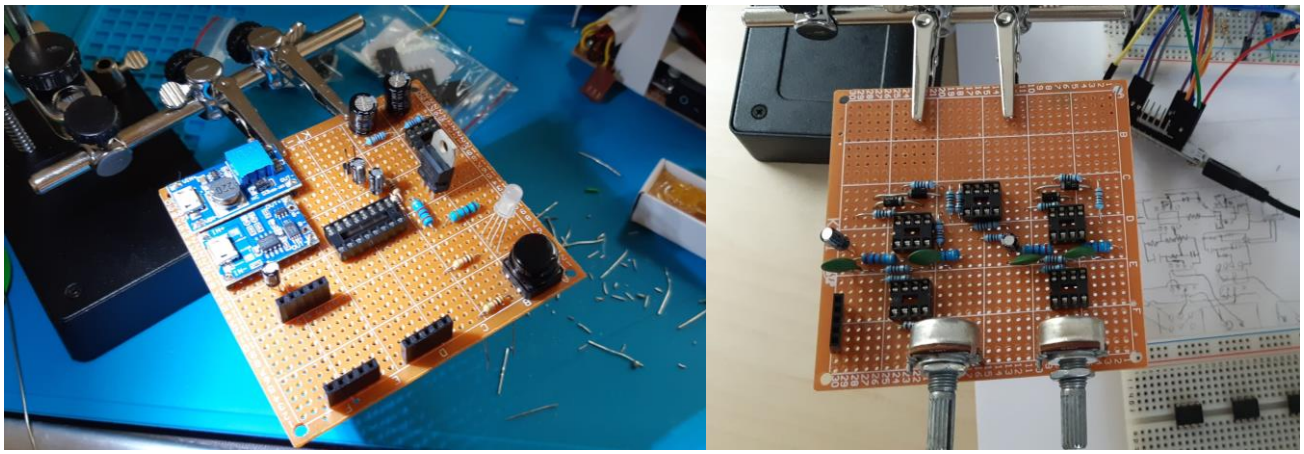


Fig 21. Building the prototype on perfboard.

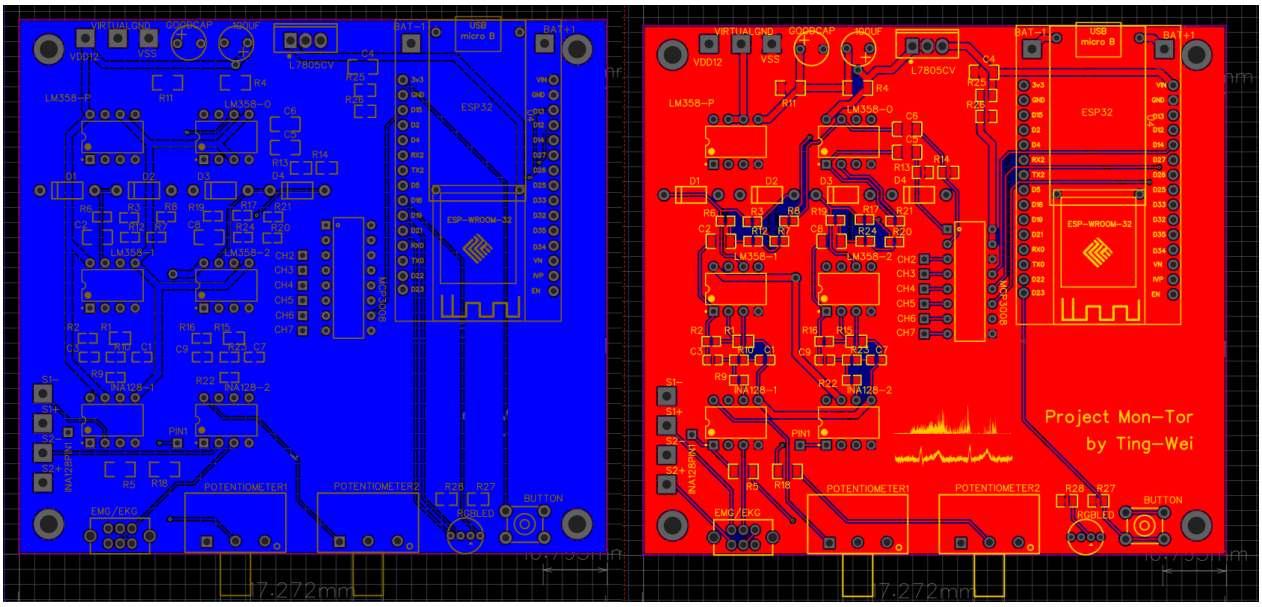


Figure 22. 2 layers PCB layout. Bottom layer on the right with copper mark as blue, top layer on the left with copper mark as red and silk screen as yellow.

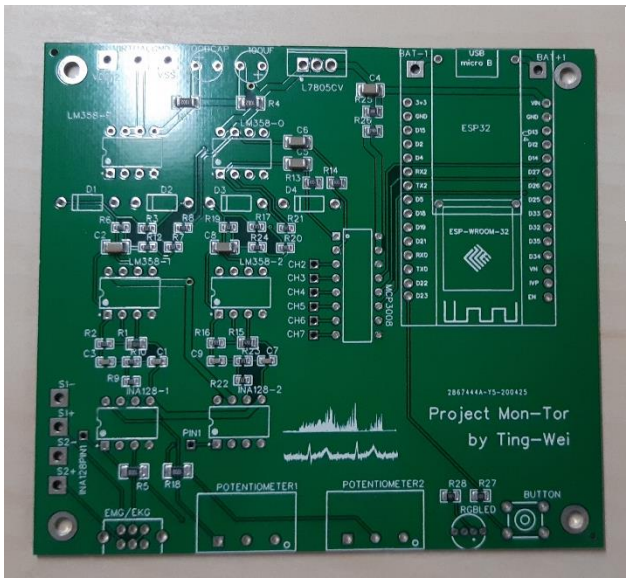
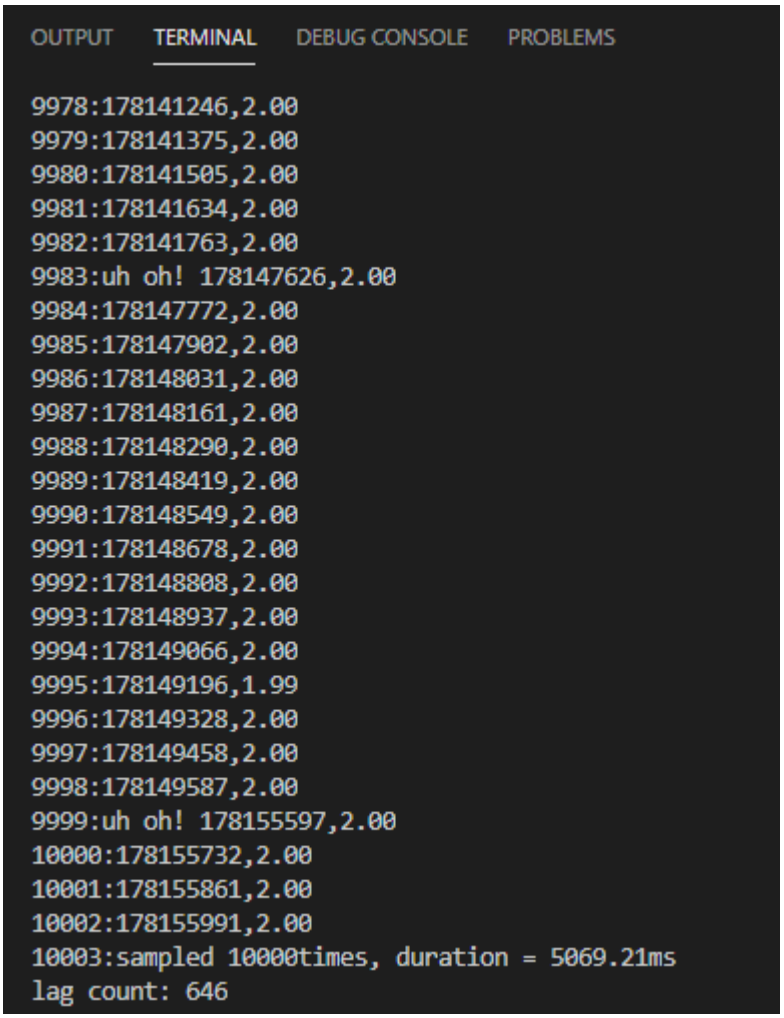


Fig 23. Manufactured PCB with SMT passive component, 5 pieces for 550 NT$ shipment included.

Difficulties encountered:

1. Esp8266 Serial port interface flash file system (SPIFFS) upload unsuccessful due to software issue. SPIFFS is used to upload the Html file from the computer to the microcontroller. After searching through the online forum, I install an older version of the tool, which got rid of the problem.
2. The computer simulation software Pspice, that I was using, does not support the number of nodes in my circuit as I'm using the student version. Luckily, I found TINA-TI, a spice simulation software by Texas Instrument that can run the circuit simulation.
3. The offset of the EMG signal needs to be adjustable so that the signal can be rectified. Another reason being the offset voltage will change when measuring different muscles.
   This will be added in the next iteration of design.
4. Using 18650 lithium ion battery cell as VCC to the amplifier is not a good solution. The +- 4.2 to 3.9V as +-VCC provided by two cells in series is limiting the output of the amplifier as seen in the Fig 19., top half of the EMG is cut off.
   The better solution in my opinion would be using single cell with a boost converter. The 3400mAh battery cell has enough capacity consider the current drawn from the esp8266 and the circuit is around 75mA on average. For a boost converter that can raise the voltage to 12V should be enough. VCC is 6V, and esp8266 can also run on that voltage.
   This will be added in the next iteration of design.

5. When Esp8266 is communicating with external ADC and storing the value to its flash memory, there exist delays between consecutive operations, the amount of delay is around 6 ms.



6. Assume 1ksps, for a 10 second operation, with time and a voltage value store in a 4 byte for each sample, that's 80000byte ~= 79KB required SRAM size which is too small for ESP-8266. I change the microcontroller to ESP-32. It has a 520KB SRAM with 4MB external flash and two powerful cores.

Codes:

1. Analog read speed testing

esp8266_analogread_speedtest§

```
const int analogInPin = A0;  // ESP8266 Analog Pin ADC0 = A0

int sensorValue = 0;  // value read from the pot
int outputValue = 0;  // value to output to a PWM pin

void setup() {
  // initialize serial communication at 115200
  Serial.begin(500000);
}

void loop() {

  int count=0;
  count=micros();
  for(int i=0;i<10000;i++ ){
    sensorValue = analogRead(analogInPin);
    outputValue = map(sensorValue, 0, 1024, 0, 255);

  }
  int interval=(micros()-count)/10000;
  Serial.print("Sample interval:");
  Serial.print(interval);
  Serial.print("us \n");
  Serial.print("Sampling rate:");
  Serial.print(1000000/interval);
  Serial.println("Hz");



  delay(1000);
}
```

2. Webserver test

server1

```
*/

// Import required libraries
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <FS.h>
#include <Wire.h>

// Replace with your network credentials
const char* ssid = "HehexD";
const char* password = "00000008";

// Set LED GPIO
const int ledPin = 2;
// Stores LED state
String ledState;
// ESP8266 Analog Pin ADC0 = A0
const int analogInPin = A0;
// value read from the input voltage
float sensorValue = 0;
float voltValue = 0;


// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

String getvoltage() {
  // Read voltage from ADC0
  sensorValue = analogRead(analogInPin);
  voltValue = sensorValue*3.3/1024;
  Serial.println(voltValue);
  return String(voltValue);
}


// Replaces placeholder with LED state value
String processor(const String& var){
  Serial.println(var);
  if(var == "STATE"){
    if(digitalRead(ledPin)){
      ledState = "ON";
    }
    else{
      ledState = "OFF";
```

server1

```
// Replaces placeholder with LED state value
String processor(const String& var){
  Serial.println(var);
  if(var == "STATE"){
    if(digitalRead(ledPin)){
      ledState = "ON";
    }
    else{
      ledState = "OFF";
    }
    Serial.print(ledState);
    return ledState;
  }
  else if (var == "VOLTAGE"){
    return getvoltage();
  }
}

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);

  // Initialize SPIFFS
  if(!SPIFFS.begin()){
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  // Print ESP32 Local IP Address
  Serial.println(WiFi.localIP());

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", String(), false, processor);
  });
```

server1

```
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  // Print ESP32 Local IP Address
  Serial.println(WiFi.localIP());

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", String(), false, processor);
  });

  // Route to load style.css file
  server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/style.css", "text/css");
  });

  // Route to set GPIO to HIGH
  server.on("/on", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin, HIGH);
    request->send(SPIFFS, "/index.html", String(), false, processor);
  });

  // Route to set GPIO to LOW
  server.on("/off", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin, LOW);
    request->send(SPIFFS, "/index.html", String(), false, processor);
  });

  server.on("/voltage", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", getvoltage().c_str());
  });

  // Start server
  server.begin();
}

void loop(){

}
```

3. HTML in flash memory of esp8266

index - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
<!DOCTYPE html>
<!--
  Rui Santos
  Complete project details at https://RandomNerdTutorials.com
-->
<html>
<head>
  <title>ESP8266 Web Server</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <h1>ESP8266 Web Server</h1>
  <p>GPIO state<strong> %STATE%</strong></p>
  <p>
    <a href="/on"><button class="button">ON</button></a>
    <a href="/off"><button class="button button2">OFF</button></a>
  </p>
  <p>
    <span class="sensor-labels">Voltage</span>
    <span id="voltage">%VOLTAGE%</span>
    <sup class="units">Volts</sup>
  </p>
</body>
<script>
  setInterval(function ( ) {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("voltage").innerHTML = this.responseText;
      }
    };
    xhttp.open("GET", "/voltage", true);
    xhttp.send();
  }, 100 ) ;

</script>
</html>
```

4. Web chart code

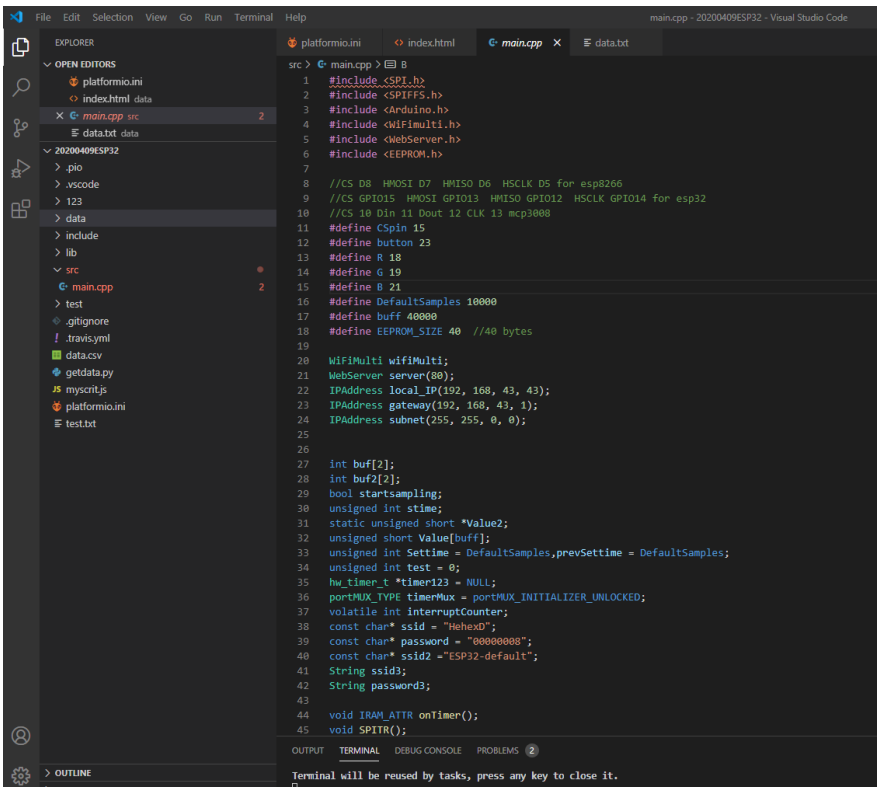
ESP_Chart_Web_Server

```
//#include <SPIFFS.h>
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <Hash.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <FS.h>
#include <Wire.h>

// Replace with your network credentials
const char* ssid = "HehexD";
const char* password = "00000008";
// ESP8266 Analog Pin ADC0 = A0
const int analogInPin = A0;
// value read from the input voltage
float sensorValue = 0;
float voltValue = 0;


// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

String getvoltage() {
  // Read voltage from ADC0
  sensorValue = analogRead(analogInPin);
  voltValue = sensorValue*3.3/1024;
  Serial.println(voltValue);
  return String(voltValue);
}

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);

  bool status;
  // default settings


  // Initialize SPIFFS
  if(!SPIFFS.begin()){
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }
```

```
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  // Print ESP32 Local IP Address
  Serial.println(WiFi.localIP());

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html");
  });
  server.on("/voltage", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", getvoltage().c_str());
  });

  // Start server
  server.begin();
}

void loop(){

}
```

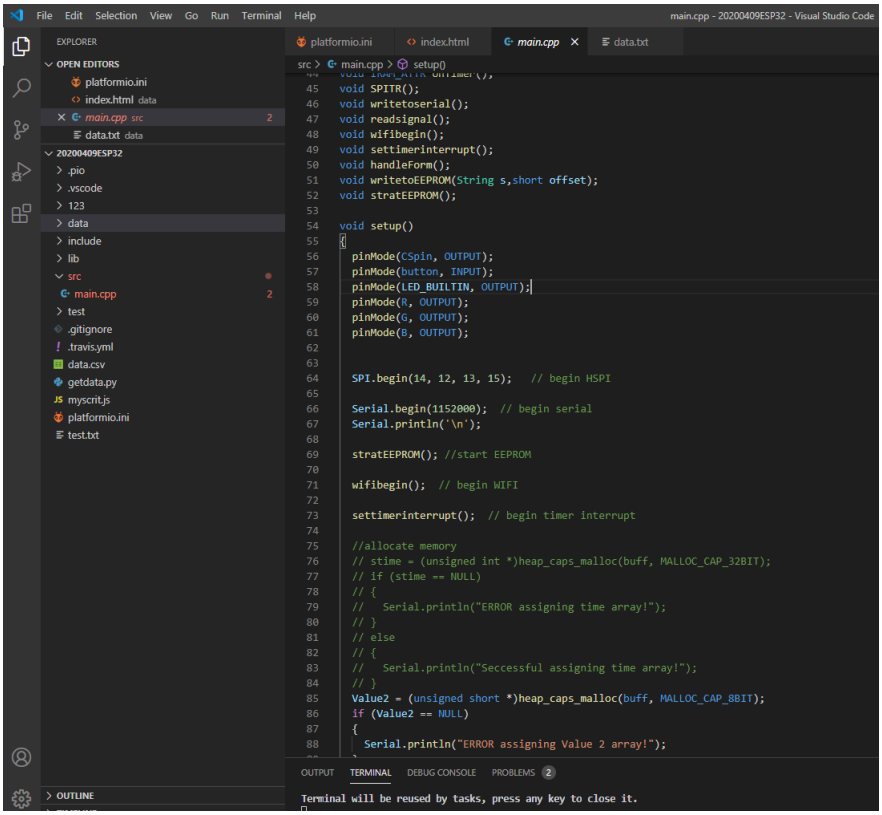
```
<!DOCTYPE HTML><html>
<!-- Rui Santos - Complete project details at https://RandomNerdTutorials.com

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files.
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software. -->
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script src="https://code.highcharts.com/highcharts.js"></script>
  <style>
    body {
      min-width: 310px;
        max-width: 800px;
        height: 400px;
      margin: 0 auto;
    }
    h2 {
      font-family: Arial;
      font-size: 2.5rem;
      text-align: center;
    }
  </style>
</head>
<body>
  <h2>MON-TOR</h2>
  <div id="chart-voltage" class="container"></div>
</body>
<script>
var chartT = new Highcharts.Chart({
  chart:{ renderTo : 'chart-voltage' },
  title: { text: 'Mon-Tor signal' },
  series: [{
    showInLegend: false,
    data: []
  }],
  plotOptions: {
    line: { animation: false,
      dataLabels: { enabled: false }
    },
    series: { color: '#059e8a' }
  },
  xAxis: { type: 'datetime',
    dateTimeLabelFormats: { second: '%H:%M:%S' }
  },
  yAxis: {
    title: { text: 'Voltage (Volts)' }
    //title: { text: 'Temperature (Fahrenheit)' }
  },
  credits: { enabled: false }
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var x = (new Date()).getTime(),
          y = parseFloat(this.responseText);
      //console.log(this.responseText);
      if(chartT.series[0].data.length > 200) {
        chartT.series[0].addPoint([x, y], true, true, true);
      } else {
        chartT.series[0].addPoint([x, y], true, false, true);
      }
    }
  };
  xhttp.open("GET", "/voltage", true);
  xhttp.send();
}, 50 ) ;

</script>
</html>
```

5. The final code



```
#include <SPI.h>
#include <SPIFFS.h>
#include <Arduino.h>
#include <WiFimulti.h>
#include <WebServer.h>
#include <EEPROM.h>

//CS D8  HMOSI D7  HMISO D6  HSCLK D5 for esp8266
//CS GPIO15  HMOSI GPIO13  HMISO GPIO12  HSCLK GPIO14 for esp32
//CS 10 Din 11 Dout 12 CLK 13 mcp3008
#define CSpin 15
#define button 23
#define R 18
#define G 19
#define B 21
#define DefaultSamples 10000
#define buff 40000
#define EEPROM_SIZE 40  //40 bytes

WiFiMulti wifiMulti;
WebServer server(80);
IPAddress local_IP(192, 168, 43, 43);
IPAddress gateway(192, 168, 43, 1);
IPAddress subnet(255, 255, 0, 0);


int buf[2];
int buf2[2];
bool startsampling;
unsigned int stime;
static unsigned short *Value2;
unsigned short Value[buff];
unsigned int Settime = DefaultSamples,prevSettime = DefaultSamples;
unsigned int test = 0;
hw_timer_t *timer123 = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
volatile int interruptCounter;
const char* ssid = "HehexD";
const char* password = "00000008";
const char* ssid2 ="ESP32-default";
String ssid3;
String password3;

void IRAM_ATTR onTimer();
void SPITR();
```

```
Terminal will be reused by tasks, press any key to close it.
```



```
void SPITR();
void writetoserial();
void readsignal();
void wifibegin();
void settimerinterrupt();
void handleForm();
void writetoEEPROM(String s,short offset);
void stratEEPROM();

void setup()
{
  pinMode(CSpin, OUTPUT);
  pinMode(button, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(R, OUTPUT);
  pinMode(G, OUTPUT);
  pinMode(B, OUTPUT);


  SPI.begin(14, 12, 13, 15);   // begin HSPI

  Serial.begin(1152000);  // begin serial
  Serial.println('\n');

  stratEEPROM(); //start EEPROM

  wifibegin();  // begin WIFI

  settimerinterrupt();  // begin timer interrupt

  //allocate memory
  // stime = (unsigned int *)heap_caps_malloc(buff, MALLOC_CAP_32BIT);
  // if (stime == NULL)
  // {
  //   Serial.println("ERROR assigning time array!");
  // }
  // else
  // {
  //   Serial.println("Seccessful assigning time array!");
  // }
  Value2 = (unsigned short *)heap_caps_malloc(buff, MALLOC_CAP_8BIT);
  if (Value2 == NULL)
  {
    Serial.println("ERROR assigning Value 2 array!");
```

```
Terminal will be reused by tasks, press any key to close it.
```


Help
main.cpp - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
src > main.cpp > setup()
// else
// {
//   Serial.println("Seccessful assigning time array!");
// }
Value2 = (unsigned short *)heap_caps_malloc(buff, MALLOC_CAP_8BIT);
if (Value2 == NULL)
{
Serial.println("ERROR assigning Value 2 array!");
}
else
{
Serial.println("Seccessful assigning Value 2 array!");
}
Serial.println(esp_get_free_heap_size());
//start SPIFFS
if (!SPIFFS.begin())
{ /* begin SPIFFS */
Serial.println("SPIFFS begins failed.");
}
else
{
Serial.println("SPIFFS begins.");
}
//Server reply settings
server.on("/myscript.js",[](){
File file = SPIFFS.open("/myscript.js", "r");
// Open it
server.streamFile(file, "application/javascript"); // And send it to the client
file.close();
});
server.on("/",[](){
File file = SPIFFS.open("/index.html", "r");
// Open it
server.streamFile(file, "text/html"); // And send it to the client
file.close();
delay(100);
});
server.on("/data.txt",[](){
File file = SPIFFS.open("/data.txt", "r");
// Open it
server.streamFile(file, "text/plain"); // And send it to the client
file.close();
});
server.on("/set10",[](){
Settime = 10000;
timerAlarmWrite(timer123, 1000, true);
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
main.cpp - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
src > main.cpp > setup()
server.on("/data.txt",[](){
File file = SPIFFS.open("/data.txt", "r");
// Open it
server.streamFile(file, "text/plain"); // And send it to the client
file.close();
});
server.on("/set10",[](){
Settime = 10000;
timerAlarmWrite(timer123, 1000, true);
server.send(200, "text/plain", "MCU response: Set sampling time 10s.");
});
server.on("/set20",[](){
Settime = 20000;
timerAlarmWrite(timer123, 1000, true);
server.send(200, "text/plain", "MCU response: Set sampling time 20s.");
});
server.on("/set30",[](){
Settime = 30000;
timerAlarmWrite(timer123, 1000, true);
server.send(200, "text/plain", "MCU response: Set sampling time 30s.");
});
server.on("/set102K",[](){
Settime = 20000;
timerAlarmWrite(timer123, 500, true);
server.send(200, "text/plain", "MCU response: Set sampling time 10s.");
});
server.on("/set202K",[](){
Settime = 40000;
timerAlarmWrite(timer123, 500, true);
server.send(200, "text/plain", "MCU response: Set sampling time 20s.");
});
server.on("/action_page",[](){
handleForm();
});
server.on("/startsampling",[](){
startsampling = true;
server.send(200, "text/plain", "Start sampling.");
});
server.begin();
// Actually start the server
Serial.println("HTTP server started");
}
void loop(void)
{
digitalWrite(G, HIGH);
digitalWrite(LED_BUILTIN, LOW);
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
main.cpp - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
src > main.cpp > setup()
server.begin(); // Actually start the server
Serial.println("HTTP server started");
}
void loop(void)
{
digitalWrite(G, HIGH);
digitalWrite(LED_BUILTIN, LOW);
if(prevSettime != Settime) {
Serial.print(Settime);
Serial.println(" samples let's go!");
prevSettime = Settime ;
}
if(digitalRead(button)==HIGH){
startsampling = true;
}
if (startsampling == true){
digitalWrite(G, LOW);
readsignal();
startsampling = false;
}
server.handleClient();
}
void handleForm() {
String ssidarg = server.arg("ssid");
String PWarg = server.arg("password");
server.sendHeader("Location", "/");
server.send(302, "text/plain", "Updated- Press Back Button");
Serial.print("New ssid:");
Serial.println(ssidarg);
writetoEEPROM(ssidarg,0);
Serial.print("New password:");
Serial.println(PWarg);
writetoEEPROM(PWarg,EEPROM_SIZE/2);
EEPROM.commit();
}
void settimerinterrupt(){
timer123 = timerBegin(1, 80, true);
timerAttachInterrupt(timer123, &onTimer, true);
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
main.cpp - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
src > main.cpp > setup()
void settimerinterrupt(){
timer123 = timerBegin(1, 80, true);
timerAttachInterrupt(timer123, &onTimer, true);
timerAlarmWrite(timer123, 1000, true);
timerAlarmEnable(timer123);
}
void wifibegin(){
volatile unsigned short wifidelay = 0;
wifiMulti.addAP(ssid, password);
wifiMulti.addAP(ssid2);
wifiMulti.addAP(ssid3.c_str(), password3.c_str());
// Configures static IP address
if (!WiFi.config(local_IP, gateway, subnet)) {
Serial.println("STA Failed to configure");
}
Serial.println("Connecting ...");
while (wifiMulti.run() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
digitalWrite(R, !digitalRead(R));
Serial.print('.');
wifidelay+=1;
if(wifidelay > 4){
Serial.println("Unable to connect Wi-Fi");
break;
}
}
digitalWrite(R,LOW);
Serial.println('\n');
Serial.print("Connected to ");
Serial.println(WiFi.SSID()); // Tell us what network we're connected to
Serial.print("IP address:\t");
Serial.println(WiFi.localIP()); // Send the IP address of the ESP8266 to the computer
}
void SPITR()
{
// enable Slave Select
digitalWrite(CSpin, LOW);
SPI.transfer(1); // initiate transmission
for (int pos = 0; pos < 2; pos++)
{
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.

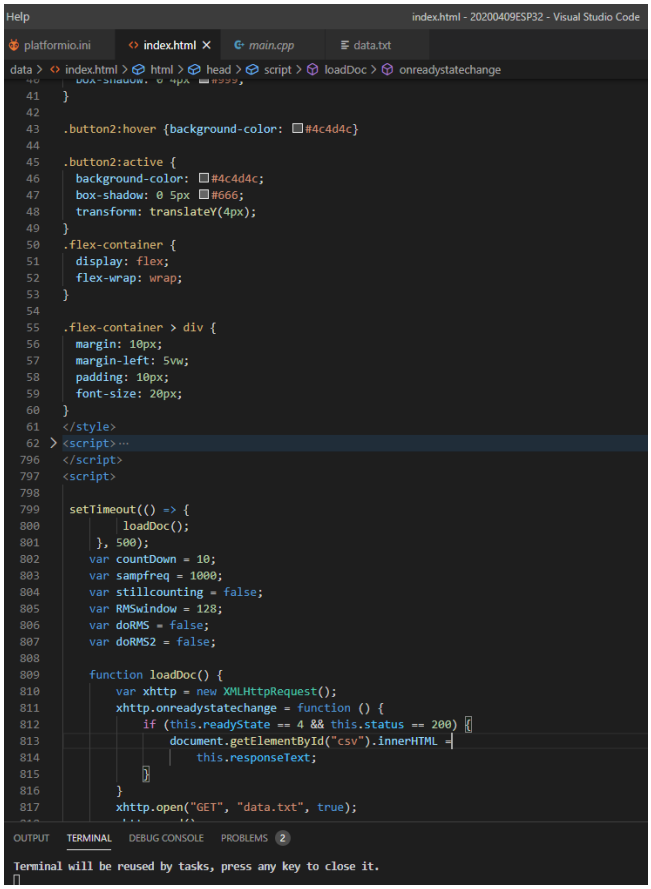

Help
main.cpp - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
src > main.cpp > setup()
Serial.println(WiFi.localIP());
// Send the IP address of the ESP8266 to the c
}
void SPITR()
{
// enable Slave Select
digitalWrite(CSpin, LOW);
SPI.transfer(1); // initiate transmission
for (int pos = 0; pos < 2; pos++)
{
buf[pos] = SPI.transfer(144); //128 for single mode, CH0
}
// disable Slave Select
digitalWrite(CSpin, HIGH);
digitalWrite(CSpin, LOW);
SPI.transfer(1); // initiate transmission
for (int pos = 0; pos < 2; pos++)
{
buf2[pos]= SPI.transfer(128); //144 for single mode, CH1
}
// disable Slave Select
digitalWrite(CSpin, HIGH);
}
void readsignal(void)
{
digitalWrite(LED_BUILTIN, HIGH);
digitalWrite(R, HIGH);
Serial.println("Reading signal from ADC.");
for (int i = 0; i < Settime; i++)
{
if (interruptCounter > 0)
{
portENTER_CRITICAL(&timerMux);
interruptCounter = 0;
portEXIT_CRITICAL(&timerMux);
if(micros()-stime >600) Serial.println("");
stime = micros();
SPITR();
buf[0] = (buf[0] & B00000011) << 8;
buf2[0] = (buf2[0] & B00000011) << 8;
Value[i] = (buf[0] + buf[1]);
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
main.cpp - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
src > main.cpp > SPITR()
}
void readsignal(void)
{
digitalWrite(LED_BUILTIN, HIGH);
digitalWrite(R, HIGH);
Serial.println("Reading signal from ADC.");
for (int i = 0; i < Settime; i++)
{
if (interruptCounter > 0)
{
portENTER_CRITICAL(&timerMux);
interruptCounter = 0;
portEXIT_CRITICAL(&timerMux);
if(micros()-stime >600) Serial.println("");
stime = micros();
SPITR();
buf[0] = (buf[0] & B00000011) << 8;
buf2[0] = (buf2[0] & B00000011) << 8;
Value[i] = (buf[0] + buf[1]);
Value2[i]= (buf2[0] + buf2[1]);
}
else
{
i--;
}
}
digitalWrite(R, LOW);
//writing to data.txt
digitalWrite(B, HIGH);
File f = SPIFFS.open("/data.txt", "w");
if (!f) {
Serial.println("file open failed");
}
Serial.println("writing to data.txt");
f.print ("Time(ms),Channel 1,Channel 2\n");
for (int i = 0; i < Settime; i++)
{
// test = stime[i]-stime[0];
f.print(i);
f.print(",");
f.print((float)(Value[i]) / 1024 * 3.3);
f.print(",");
f.print((float)(Value2[i]) / 1024 * 3.3);
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
main.cpp - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
src > main.cpp > SPITR()
for (int i = 0; i < Settime; i++)
{
// test = stime[i]-stime[0];
f.print(i);
f.print(",");
f.print((float)(Value[i]) / 1024 * 3.3);
f.print(",");
f.print((float)(Value2[i]) / 1024 * 3.3);
f.print("\n");
}
f.close();
Serial.println("End of printing.");
Serial.print("sampled ");
Serial.print(Settime);
Serial.print("times, ");
// Serial.print("duration = ");
// Serial.print(float(stime[Settime-1]-stime[0])/1000);
// Serial.println("ms");
digitalWrite(8, LOW);
digitalWrite(LED_BUILTIN, LOW);
}
void IRAM_ATTR onTimer()
{
portENTER_CRITICAL_ISR(&timerMux);
interruptCounter++;
portEXIT_CRITICAL_ISR(&timerMux);
}
void writetoEEPROM(String s,short offset){
for(int i = 0+offset; i <= offset + s.length(); i++){
EEPROM.write(i,byte(s.charAt(i-offset)));
}
}
void stratEEPROM(){
char r;
ssid3 = ""; password3 = "";
if (!EEPROM.begin(EEPROM_SIZE))
{
Serial.println("failed to initialise EEPROM");
}
Serial.println(" bytes read from Flash . Values are:");
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
main.cpp - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
src > main.cpp > onTimer()
void IRAM_ATTR onTimer()
{
portENTER_CRITICAL_ISR(&timerMux);
interruptCounter++;
portEXIT_CRITICAL_ISR(&timerMux);
}
void writetoEEPROM(String s,short offset){
for(int i = 0+offset; i <= offset + s.length(); i++){
EEPROM.write(i,byte(s.charAt(i-offset)));
}
}
void stratEEPROM(){
char r;
ssid3 = ""; password3 = "";
if (!EEPROM.begin(EEPROM_SIZE))
{
Serial.println("failed to initialise EEPROM");
}
Serial.println(" bytes read from Flash . Values are:");
for (int i = 0; i < EEPROM_SIZE/2; i++)
{
r = EEPROM.read(i);
Serial.print(byte(r)); Serial.print(" ");
if(byte(r)==0){
break;
}
ssid3 += r;
}
for (int i = 0+EEPROM_SIZE/2; i < EEPROM_SIZE; i++)
{
r = EEPROM.read(i);
Serial.print(byte(r)); Serial.print(" ");
if(byte(r)==0){
break;
}
password3 += r;
}
Serial.println("");
Serial.print(ssid3);Serial.print(" ; ");Serial.println(password3);
}
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.

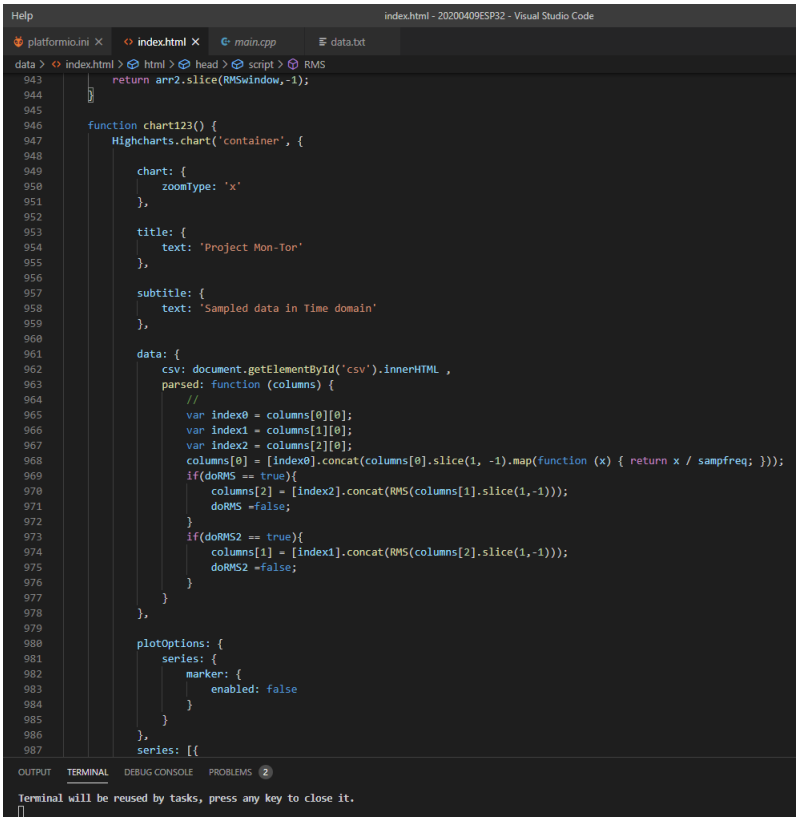
6. HTML in flash memory of esp32




Help
index.html - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
data > index.html > html > head > script > loadDoc > onreadystatechange
<script> ···
</script>
<script>
setTimeout(() => {
loadDoc();
}, 500);
var countDown = 10;
var sampfreq = 1000;
var stillcounting = false;
var RMSwindow = 128;
var doRMS = false;
var doRMS2 = false;
function loadDoc() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("csv").innerHTML =
this.responseText;
}
}
xhttp.open("GET", "data.txt", true);
xhttp.send();
setTimeout(() => {
chart123();
}, 500);
}
function set10s() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("esp respond").innerHTML =
this.responseText;
}
}
xhttp.open("GET", "set10", true);
xhttp.send();
countDown = 10;
sampfreq = 1000;
}
function set20s() { ···
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
index.html - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
data > index.html > html > head > script > loadDoc > onreadystatechange
function set10s2K() { ···
function set20s2K() {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("esp respond").innerHTML =
this.responseText;
}
}
xhttp.open("GET", "set202K", true);
xhttp.send();
countDown = 20;
sampfreq = 2000;
}
function Countdown() {
if(stillcounting!=true){
stillcounting = true;
console.log("yo");
var x = setInterval(function () {
document.getElementById("countdown").innerHTML = (countDown - 0) + "s ";
if ((countDown - 0) == 0) {
clearInterval(x);
document.getElementById("countdown").innerHTML = "DONE! Please refresh this page.";
}
countDown -= 1;
}, 1000);
stillcounting = false;
countDown = 10;
}
}
function startsampling() {
Countdown();
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("SS").innerHTML =
"Sampling finished in: ";
}
}
xhttp.open("GET", "startsampling", true);
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.

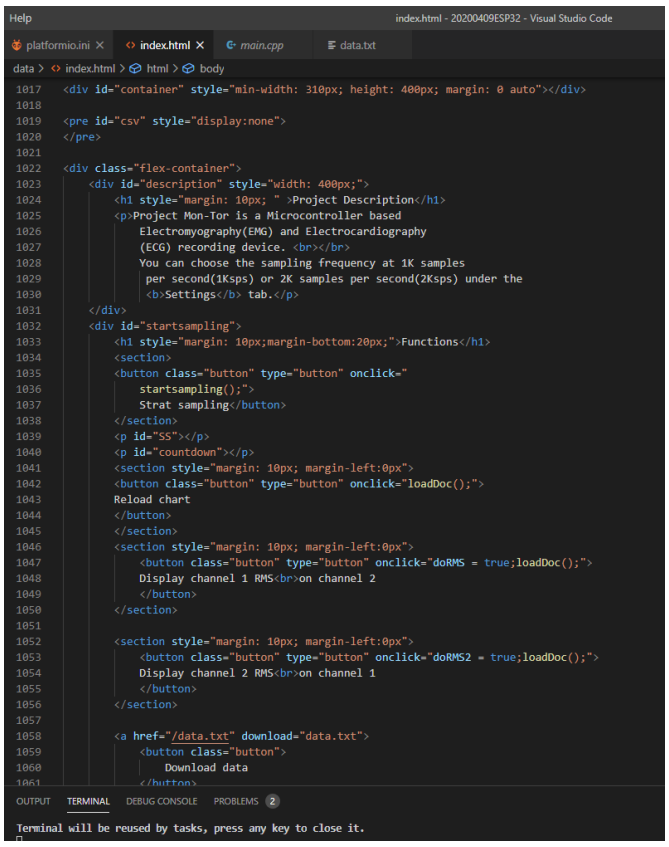

Help
index.html - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
data > index.html > html > head > script > WTF
912
913 function startsampling() {
914 Countdown();
915 var xhttp = new XMLHttpRequest();
916 xhttp.onreadystatechange = function () {
917 if (this.readyState == 4 && this.status == 200) {
918 document.getElementById("SS").innerHTML =
919 "Sampling finished in: ";
920 }
921 }
922 xhttp.open("GET", "startsampling", true);
923 xhttp.send();
924 }
925
926 function WTF() {...
929
930 function RMS(arr) {
931 var zeropad=[];
932 var arr2=[];
933 arr2.length=arr.length;
934 zeropad.length = RMSwindow;
935 zeropad.fill(0);
936 arr = zeropad.concat(arr);
937 console.log(arr);
938 for(var i=RMSwindow;i<arr.length;i++) {
939 arr2[i] = Math.sqrt(arr.slice(i-RMSwindow,i).reduce((a, b) => a +(b*b))/RMSwindow);
940 }
941 console.log(arr);
942 doRMS = false;
943 return arr2.slice(RMSwindow,-1);
944 }
945
946 function chart123() {
947 Highcharts.chart('container', {
948
949 chart: {
950 zoomType: 'x'
951 },
952
953 title: {
954 text: 'Project Mon-Tor'
955 },
956
957 subtitle: {
958 text: 'Sampled data in Time domain'
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
index.html - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
data > index.html > html > head > script > RMS
943 return arr2.slice(RMSwindow,-1);
944 }
945
946 function chart123() {
947 Highcharts.chart('container', {
948
949 chart: {
950 zoomType: 'x'
951 },
952
953 title: {
954 text: 'Project Mon-Tor'
955 },
956
957 subtitle: {
958 text: 'Sampled data in Time domain'
959 },
960
961 data: {
962 csv: document.getElementById('csv').innerHTML ,
963 parsed: function (columns) {
964 //
965 var index0 = columns[0][0];
966 var index1 = columns[1][0];
967 var index2 = columns[2][0];
968 columns[0] = [index0].concat(columns[0].slice(1, -1).map(function (x) { return x / sampfreq; }));
969 if(doRMS == true){
970 columns[2] = [index2].concat(RMS(columns[1].slice(1,-1)));
971 doRMS =false;
972 }
973 if(doRMS2 == true){
974 columns[1] = [index1].concat(RMS(columns[2].slice(1,-1)));
975 doRMS2 =false;
976 }
977 }
978 },
979
980 plotOptions: {
981 series: {
982 marker: {
983 enabled: false
984 }
985 }
986 },
987 series: [{
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
index.html - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
data > index.html > html > head > script > chart123 > data > parsed
plotOptions: {
series: {
marker: {
enabled: false
}
}
},
series: [{
lineWidth: 0.5,
name: 'channel 1'
}],
xAxis: [{
title: {
text: 'Time(s)'
}
}],
yAxis: [{
title: {
text: 'Voltage(V)'
},
min: 0,
max: 3.3
}]
});
}
</script>
</head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<body>

<pre id="csv" style="display:none">
</pre>

<h1 style="margin: 10px; " >Project Description</h1>
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
index.html - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
data > index.html > html > body

<pre id="csv" style="display:none">
</pre>

<h1 style="margin: 10px; " >Project Description</h1>
Project Mon-Tor is a Microcontroller based
Electromyography(EMG) and Electrocardiography
(ECG) recording device.
</br>
You can choose the sampling frequency at 1K samples
per second(1Ksps) or 2K samples per second(2Ksps) under the
Settings tab.

<h1 style="margin: 10px;margin-bottom:20px;">Functions</h1>
<section>
<button class="button" type="button" onclick="
startsampling();">
Strat sampling</button>
</section>

<section style="margin: 10px; margin-left:0px">
<button class="button" type="button" onclick="loadDoc();">
Reload chart
</button>
</section>
<section style="margin: 10px; margin-left:0px">
<button class="button" type="button" onclick="doRMS = true;loadDoc();">
Display channel 1 RMS
on channel 2
</button>
</section>
<section style="margin: 10px; margin-left:0px">
<button class="button" type="button" onclick="doRMS2 = true;loadDoc();">
Display channel 2 RMS
on channel 1
</button>
</section>

<button class="button">
Download data
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
index.html - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
data > index.html > html > body
</section>

<button class="button">
Download data
</button>

<h1 style="margin: 10px;">
Settings
</h1>
<h2 style="font-size:120%;">1K samples per second (1Ksps)</h2>
<section id="set 10s" style="margin: 8px">
<button class="button" type="button" onclick="set10s();
document.getElementById('set 10s').innerHTML =
' setting period to 10 seconds ';">
for 10 seconds(default)</button>
</section>
<section id="set 20s" style="margin: 8px">
<button class="button" type="button" onclick="set20s();
document.getElementById('set 20s').innerHTML =
' setting period to 20 seconds ';">
for 20 seconds</button>
</section>
<section id="set 30s" style="margin: 8px">
<button class="button" type="button" onclick="set30s();
document.getElementById('set 30s').innerHTML =
' setting period to 30 seconds ';">
for 30 seconds</button>
</section>
<h2 style="font-size:120%;">2K samples per second (2Ksps)</h2>
<section id="2K set 10s" style="margin: 8px">
<button class="button" type="button" onclick=" set10s2K();
document.getElementById('2K set 10s').innerHTML =
' setting period to 10 seconds ';">
for 10 seconds</button>
</section>
<section id="2K set 20s" style="margin: 8px">
<button class="button" type="button" onclick="set20s2K();
document.getElementById('2K set 20s').innerHTML =
' setting period to 20 seconds ';">
for 20 seconds</button>
</section>
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.


Help
index.html - 20200409ESP32 - Visual Studio Code
platformio.ini
index.html
main.cpp
data.txt
data > index.html > html > body
' setting period to 10 seconds ';">
for 10 seconds(default)</button>
</section>
<section id="set 20s" style="margin: 8px">
<button class="button" type="button" onclick="set20s();
document.getElementById('set 20s').innerHTML =
' setting period to 20 seconds ';">
for 20 seconds</button>
</section>
<section id="set 30s" style="margin: 8px">
<button class="button" type="button" onclick="set30s();
document.getElementById('set 30s').innerHTML =
' setting period to 30 seconds ';">
for 30 seconds</button>
</section>
<h2 style="font-size:120%;">2K samples per second (2Ksps)</h2>
<section id="2K set 10s" style="margin: 8px">
<button class="button" type="button" onclick=" set10s2K();
document.getElementById('2K set 10s').innerHTML =
' setting period to 10 seconds ';">
for 10 seconds</button>
</section>
<section id="2K set 20s" style="margin: 8px">
<button class="button" type="button" onclick="set20s2K();
document.getElementById('2K set 20s').innerHTML =
' setting period to 20 seconds ';">
for 20 seconds</button>
</section>

<button class="button2" type="button" onclick="WTF();">
Show Advanced Setting</button>

</body>
</html>
OUTPUT
TERMINAL
DEBUG CONSOLE
PROBLEMS 2
Terminal will be reused by tasks, press any key to close it.