

Lab 4: The Hardware

ECE 180D: Systems Design Laboratory

Contents

1	Introduction	1
2	Materials and Preparation	2
3	Overview of the Hardware	2
3.1	Microcontroller	2
3.2	IMU	3
4	Setting up Arduino IDE for ESP32 and IMU	3
5	Wireless Connection	4
5.1	Standard Wireless Connection	4
5.2	Eduroam (not currently working)	4
5.3	Multiple WiFi Networks (optional)	4
6	MQTT on the ESP32	5
7	Task 4: IMU Classification	10

1 Introduction

This tutorial will introduce the basics of how to use the ESP32 board from Sparkfun and connecting it to the IMU. As this is the first year we will be using this microcontroller, this tutorial will primarily link you to other online tutorials interspersed with additional points of greater confusion.

2 Materials and Preparation

1. ESP32-S2 Thing Plus.
2. USB 3.1 Cable A to C Cable.
3. SparkFun 9DoF IMU - ICM-20948.
4. Qwiic Cable.
5. A personal computer with USB port.
6. Access to a network with an internet connection.

3 Overview of the Hardware

3.1 Microcontroller

Please find the overview of the Hardware below. Note the number of GPIO pins, processing power, power inputs / data inputs, headers, WIFI/Bluetooth chip etc.

In particular, keep in mind that if you are going to power this board through pins, you cannot use 5V and

SparkFun ESP32-S2 Thing Plus (WRL-17743)

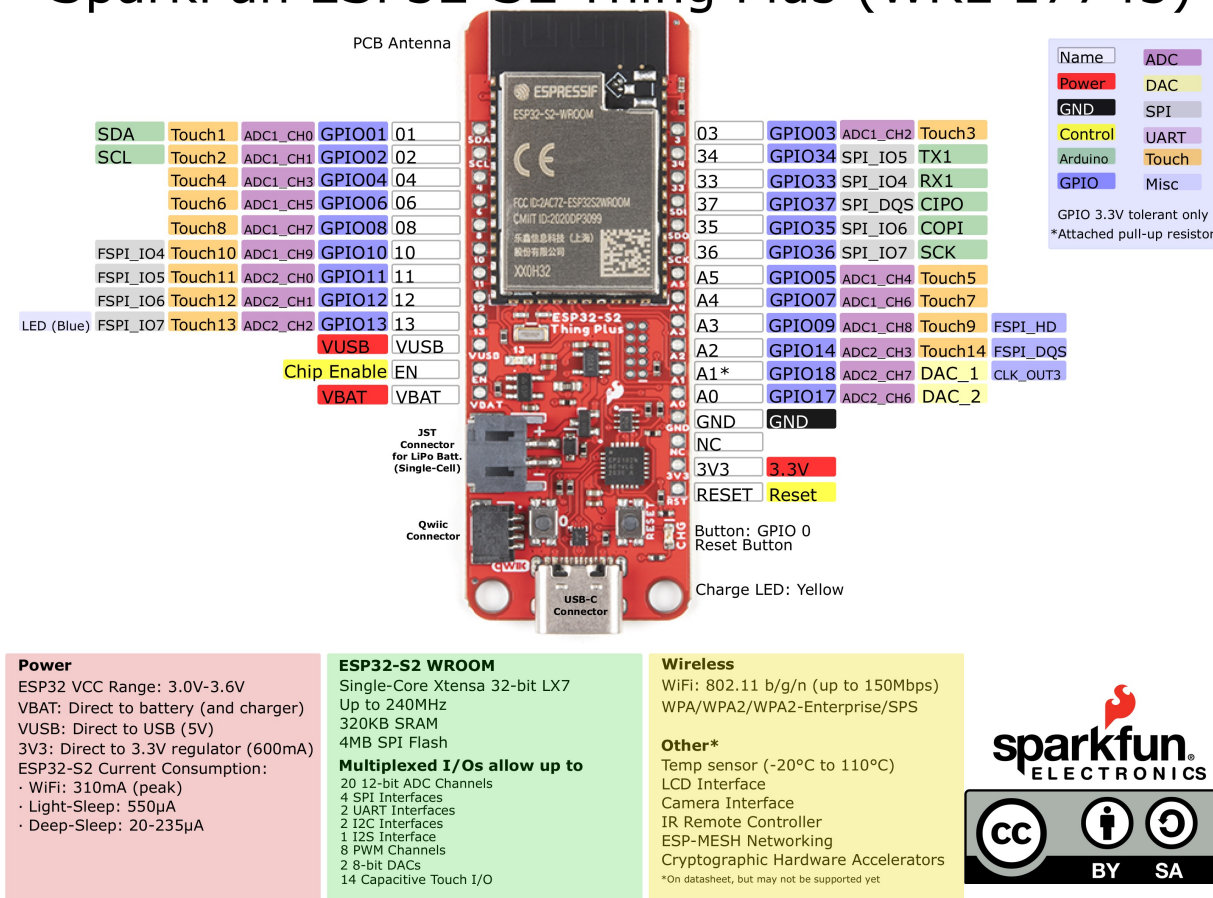


Figure 1: An overview of the Sparkfun ESP32-S2 Thing Plus as obtained from [Sparkfun](#)

must regulate the voltage down to 3.3V. If you don't do this, the expectation is a burnt board and we don't want that. If you are just going to connect via the USB-C cable to a power bank (recommended), this won't be a problem.

Other specifications can be found on the [Sparkfun tutorial](#).

3.2 IMU

An IMU is an inertial measurement unit. What that means is that it is able to measure a full-range of sensing. In particular, it can be segmented into three parts:

1. Gyroscope (angular velocity measurements)
2. Accelerometer (acceleration measurements)
3. Magnetometer (magnetic sensor)

While a magnetometer is typically not great indoors (especially in Engineering IV - may be better in a normal house / apartment, but this will have to be tested), the gyroscope and accelerometer have the possibility of doing some level of tracking of gestures and motions. Possible project ideas include:

- Head tracking for a VR project (gyroscope)
- Fall detection (accelerometer)
- athletic performance tracker (both accelerometer and gyroscope)
- indoor localization (both accelerometer and gyroscope)

While an IMU has many uses, there are several caveats to using an IMU. There is a huge amount of drift involved when using an IMU, so long-duration fine-tune tracking is a lot less plausible for use. Consider using the IMU for tracking quick gestures, such as a nod, a flick of the wrist, or the tapping of your foot.

4 Setting up Arduino IDE for ESP32 and IMU

Please refer to [the Sparkfun ESP32-S2 tutorial](#), of which you will also need to complete the [board definition installation for Arduino IDE](#). In particular, you will need to add the link https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json into your *Additional boards manager URLs* field in **File > Preferences**. Please do the entire tutorial, double check that blink works and attempt to connect to WiFi, but we will describe this a bit more in the next section.

Once completed, please also set up Arduino for the IMU. Just as with the microcontroller, please follow the [IMU guide](#). Should you have the Qwiic cable, you should be able to skip directly to the software setup step. For this tutorial get to the point where you can open up your Serial Monitor and see the IMU printouts.

Task 1:

1. Make sure your microcontroller is working. You can have a picture of a MCU with a lit LED as proof.
2. Please screenshot your Serial Monitor showing IMU printouts. This will show that your microcontroller and IMU is working.
3. You CAN use any of the BASELINE scripts linked above. CITE the references AND provide any improvements to this baseline that you may have accomplished (IN a header comment in the .py script). Please push your contributions to your ECE180D-Warmup repository.

5 Wireless Connection

Since we are working with IoT devices, eventually what we want is to access our device wirelessly. As such, we will need to do a couple other small things in order to allow our devices to work wirelessly.

5.1 Standard Wireless Connection

Unfortunately, you are not all at home this year, so this part may be somewhat less useful. This part should be easy to do on a normal home network as per the WiFi example in the [Arduino examples of the tutorial](#), reproduced below. This step should be easier to do than over a school WiFi network because there isn't as much security protections over the network as **eduroam**. You can find this file on Bruinlearn called `wifi_connect.ino`.

However, we can get around this fact using hotspots. Android phones are able to hotspot WiFi, which can completely circumvent the need for eduroam. For now, try having someone set up a hotspot on their phone and replace the strings in lines 4 and 5 with the hotspot name and password. If you are working at home, you can also feel free to just use your WiFi.

5.2 Eduroam (not currently working)

Unfortunately, we will want to get a network connection working in the lab, so it would be nice if we could connect to Eduroam. Here, we will mostly follow [this guide](#). In particular, we will be trying to reference the code provided by [his Github repo](#) or the example provided within the [esp32 repo](#).

For this part, we will need to still do a wired connection, so keep your MicroUSB cord around. Just as a reminder, eduroam can be accessed by all students with the following credentials:

Username: <your MyUCLA username>@ucla.edu
Password: <your MyUCLA password>

For the purposes of these wireless connections, remember that in order to actually connect to things, you will need to have both your laptop (the connecting device) and your Raspberry Pi to be on the same network.

Just as a reason why **eduroam** requires special attention: The **eduroam** network utilizes the **Dynamic Host Configuration Protocol** (DHCP) to assign IP addresses; as it must service users connecting and disconnecting at unknown time intervals. DHCP allows devices to request an IP address on connection initialization. Devices may be assigned a static IP address, or have one automatically assigned to them by DHCP. As such, devices that are not assigned a static IP address may not rely upon having the same IP address on initialization of a connection with the Wi-Fi network utilizing DHCP.

The ultimate meaning behind this is that **eduroam** creates a new IP address for every device each time it joins the network and that it has a bunch of hidden security that protect the IP addresses from being seen. As such, we cannot just access the Raspberry Pi using the phrase `raspberrypi.local` because **eduroam** hides that information.

5.3 Multiple WiFi Networks (optional)

While re-flashing sketches onto the ESP32 is fairly simple, you may just want a single piece of code that can find all the WiFi connections that you may use for you, like a computer. If so, feel free to try [this guide](#) that uses an example sketch from the Arduino packages.

Task 2:

1. Please screenshot your Serial Monitor showing WiFi connections, using whichever method that you have that allows it to work.

2. Again push your contributions to your ECE180D-Warmup repository. However, keep in mind that you may want to remove the usernames and passwords to your WiFi networks before you push! You can separate this out by defining an `arduino_secrets.h`, which contains

```
1 #define SECRET_SSID "YOUR_NETWORK_NAME"
2 #define SECRET_PASS "YOUR_NETWORK_PASS"
```

6 MQTT on the ESP32

Now, we finally want to set up MQTT on the microcontroller so that we can communicate between the ESP32 (and the IMU) and your computer (or other devices). This will mostly follow [this MQTT tutorial](#).

Once you have installed all the dependencies,

1. Try running `basic_imu_mqtt.ino` in BruinLearn. This code is reproduced below. Try and reference to see what the differences between this code is and the individual IMU and MQTT code is and understand how we are combining the two to send messages. In particular, note where we are publishing messages.

```
1 #include <WiFi.h>
2 #include <PubSubClient.h>
3 #include "arduino_secrets.h"
4 #include "ICM_20948.h" // Click here to get the library:
   http://librarymanager/All#SparkFun_ICM_20948_IMU
5
6 // #define USE_SPI      // Uncomment this to use SPI
7
8 #define SERIAL_PORT Serial
9
10 #define SPI_PORT SPI // Your desired SPI port. Used only when "USE_SPI" is defined
11 #define CS_PIN 2     // Which pin you connect CS to. Used only when "USE_SPI" is defined
12
13 #define WIRE_PORT Wire // Your desired Wire port. Used when "USE_SPI" is not defined
14 // The value of the last bit of the I2C address.
15 // On the SparkFun 9DoF IMU breakout the default is 1, and when the ADR jumper is closed the
   value becomes 0
16 #define ADO_VAL 1
17
18 #ifndef USE_SPI
19 ICM_20948_SPI myICM; // If using SPI create an ICM_20948_SPI object
20 #else
21 ICM_20948_I2C myICM; // Otherwise create an ICM_20948_I2C object
22 #endif
23
24 // WiFi
25 const char *ssid = SECRET_SSID; // Enter your WiFi name
26 const char *password = SECRET_PASS; // Enter WiFi password
27
28 // MQTT Broker
29 // const char *mqtt_broker = "broker.emqx.io";
30 // const char *topic = "ece180d/test";
31 // const char *mqtt_username = "emqx";
32 // const char *mqtt_password = "public";
33 // const int mqtt_port = 1883;
34
35 // MQTT Broker
36 const char *mqtt_broker = "mqtt.eclipseprojects.io";
```

```

37 const char *topic = "ece180d/test";
38 // const char *mqtt_username = "emqx";
39 // const char *mqtt_password = "public";
40 const int mqtt_port = 1883;
41
42 WiFiClient espClient;
43 PubSubClient client(espClient);
44
45 void setup() {
46     // Set software serial baud to 115200;
47     Serial.begin(115200);
48     // connecting to a WiFi network
49     WiFi.begin(ssid, password);
50     while (WiFi.status() != WL_CONNECTED) {
51         delay(500);
52         Serial.println("Connecting to WiFi..");
53     }
54     Serial.println("Connected to the WiFi network");
55     //connecting to a mqtt broker
56     client.setServer(mqtt_broker, mqtt_port);
57     client.setCallback(callback);
58     while (!client.connected()) {
59         String client_id = "esp32-client-";
60         client_id += String(WiFi.macAddress());
61         Serial.printf("The client %s connects to the public mqtt broker\n", client_id.c_str());
62         if (client.connect(client_id.c_str()) { //, mqtt_username, mqtt_password)) {
63             Serial.println("mqtt broker connected");
64         } else {
65             Serial.print("failed with state ");
66             Serial.print(client.state());
67             delay(2000);
68         }
69     }
70     // publish and subscribe
71     client.publish(topic, "Hi I'm ESP32 ^^");
72     client.subscribe(topic);
73
74     #ifdef USE_SPI
75         SPI_PORT.begin();
76     #else
77         WIRE_PORT.begin();
78         WIRE_PORT.setClock(400000);
79     #endif
80
81     bool initialized = false;
82     while (!initialized) {
83         #ifdef USE_SPI
84             myICM.begin(CS_PIN, SPI_PORT);
85         #else
86             myICM.begin(WIRE_PORT, ADO_VAL);
87         #endif
88
89         SERIAL_PORT.print(F("Initialization of the sensor returned: "));
90         SERIAL_PORT.println(myICM.statusString());
91         if (myICM.status != ICM_20948_Stat_Ok) {
92             SERIAL_PORT.println("Trying again...");
93             delay(500);
94         } else {
95             initialized = true;

```

```

96     }
97 }
98 }
99
100 void callback(char *topic, byte *payload, unsigned int length) {
101     Serial.print("Message arrived in topic: ");
102     Serial.println(topic);
103     Serial.print("Message:");
104     for (int i = 0; i < length; i++) {
105         Serial.print((char) payload[i]);
106     }
107     Serial.println();
108     Serial.println("-----");
109 }
110
111 void loop()
112 {
113
114     if (myICM.dataReady())
115     {
116         myICM.getAGMT();           // The values are only updated when you call 'getAGMT'
117                                   //   printRawAGMT( myICM.agmt ); // Uncomment this to see the raw
                                   //   values, taken directly from the agmt structure
118         printScaledAGMT(&myICM); // This function takes into account the scale settings from when
                                   //   the measurement was made to calculate the values with units
119         // client.publish(topic, myICM->acc.axes.x);
120         client.loop();
121         delay(30);
122     }
123     else
124     {
125         SERIAL_PORT.println("Waiting for data");
126         client.loop();
127         delay(500);
128     }
129 }
130
131 // Below here are some helper functions to print the data nicely!
132
133 void printPaddedInt16b(int16_t val)
134 {
135     if (val > 0)
136     {
137         SERIAL_PORT.print(" ");
138         if (val < 10000)
139         {
140             SERIAL_PORT.print("0");
141         }
142         if (val < 1000)
143         {
144             SERIAL_PORT.print("0");
145         }
146         if (val < 100)
147         {
148             SERIAL_PORT.print("0");
149         }
150         if (val < 10)
151         {
152             SERIAL_PORT.print("0");

```

```

153     }
154 }
155 else
156 {
157     SERIAL_PORT.print("-");
158     if (abs(val) < 10000)
159     {
160         SERIAL_PORT.print("0");
161     }
162     if (abs(val) < 1000)
163     {
164         SERIAL_PORT.print("0");
165     }
166     if (abs(val) < 100)
167     {
168         SERIAL_PORT.print("0");
169     }
170     if (abs(val) < 10)
171     {
172         SERIAL_PORT.print("0");
173     }
174 }
175 SERIAL_PORT.print(abs(val));
176 }
177
178 void printRawAGMT(ICM_20948_AGMT_t agmt)
179 {
180     SERIAL_PORT.print("RAW. Acc [ ");
181     printPaddedInt16b(agmt.acc.axes.x);
182     SERIAL_PORT.print(", ");
183     printPaddedInt16b(agmt.acc.axes.y);
184     SERIAL_PORT.print(", ");
185     printPaddedInt16b(agmt.acc.axes.z);
186     SERIAL_PORT.print(" ], Gyr [ ");
187     printPaddedInt16b(agmt.gyr.axes.x);
188     SERIAL_PORT.print(", ");
189     printPaddedInt16b(agmt.gyr.axes.y);
190     SERIAL_PORT.print(", ");
191     printPaddedInt16b(agmt.gyr.axes.z);
192     SERIAL_PORT.print(" ], Mag [ ");
193     printPaddedInt16b(agmt.mag.axes.x);
194     SERIAL_PORT.print(", ");
195     printPaddedInt16b(agmt.mag.axes.y);
196     SERIAL_PORT.print(", ");
197     printPaddedInt16b(agmt.mag.axes.z);
198     SERIAL_PORT.print(" ], Tmp [ ");
199     printPaddedInt16b(agmt.tmp.val);
200     SERIAL_PORT.print(" ]");
201     SERIAL_PORT.println();
202 }
203
204 void printFormattedFloat(float val, uint8_t leading, uint8_t decimals)
205 {
206     float aval = abs(val);
207     if (val < 0)
208     {
209         SERIAL_PORT.print("-");
210     }
211     else

```



```

212 {
213     SERIAL_PORT.print(" ");
214 }
215 for (uint8_t indi = 0; indi < leading; indi++)
216 {
217     uint32_t tenpow = 0;
218     if (indi < (leading - 1))
219     {
220         tenpow = 1;
221     }
222     for (uint8_t c = 0; c < (leading - 1 - indi); c++)
223     {
224         tenpow *= 10;
225     }
226     if (aval < tenpow)
227     {
228         SERIAL_PORT.print("0");
229     }
230     else
231     {
232         break;
233     }
234 }
235 if (val < 0)
236 {
237     SERIAL_PORT.print(-val, decimals);
238 }
239 else
240 {
241     SERIAL_PORT.print(val, decimals);
242 }
243 }
244
245 #ifndef USE_SPI
246 void printScaledAGMT(ICM_20948_SPI *sensor)
247 {
248     #else
249     void printScaledAGMT(ICM_20948_I2C *sensor)
250     {
251     #endif
252     SERIAL_PORT.print("Scaled. Acc (mg) [ ");
253     printFormattedFloat(sensor->accX(), 5, 2);
254     char buf[10];
255     snprintf(buf, 10, "%f", sensor->accX());
256     client.publish(topic, buf);
257     SERIAL_PORT.print(", ");
258     printFormattedFloat(sensor->accY(), 5, 2);
259     SERIAL_PORT.print(", ");
260     printFormattedFloat(sensor->accZ(), 5, 2);
261     SERIAL_PORT.print(" ], Gyr (DPS) [ ");
262     printFormattedFloat(sensor->gyrX(), 5, 2);
263     SERIAL_PORT.print(", ");
264     printFormattedFloat(sensor->gyrY(), 5, 2);
265     SERIAL_PORT.print(", ");
266     printFormattedFloat(sensor->gyrZ(), 5, 2);
267     SERIAL_PORT.print(" ], Mag (uT) [ ");
268     printFormattedFloat(sensor->magX(), 5, 2);
269     SERIAL_PORT.print(", ");
270     printFormattedFloat(sensor->magY(), 5, 2);

```

```

271 SERIAL_PORT.print(", ");
272 printFormattedFloat(sensor->magZ(), 5, 2);
273 SERIAL_PORT.print(" ], Tmp (C) [ ");
274 printFormattedFloat(sensor->temp(), 5, 2);
275 SERIAL_PORT.print(" ]");
276 SERIAL_PORT.println();
277 }

```

2. Now, run your `mqtt_sub.py` that we used in past weeks. Edit the topic in both scripts so that you are on a unique topic identifier and you will start seeing IMU data send to your computer.

Task 3:

1. Please screenshot you receiving IMU messages on your computer's command-line (through Python code).
2. Observe how much lag is present. Are there ways to reduce the lag (e.g. reducing the frequency at which messages are sent)? What if you could do processing on the IMU itself so that only a single message is sent when something is recognized? What are other ways of getting around the lag, if you have to have lag?
3. Again push your contributions to your ECE180D-Warmup repository. Again, make sure to remove any usernames and passwords from whatever is pushed publicly!

7 Task 4: IMU Classification

1. Visually explore the IMU data from the constant stream of input upon running the IMU code. Roughly determine the $+x$, $+y$, $+z$ directions and confirm the roll, pitch, yaw (rotations about the x , y , z axes, respectively) values. Do you see the gravity acceleration when idle?
2. Roughly characterize what the idle IMU position looks like from the provided features. Rotate the IMU in each direction to idle in a different orientation. Do the values drift even when idle? What is a good feature to classify idle vs. non-idle? Make this classification and record the accuracy (make a confusion matrix - how many idle experiments are classified as idle/non-idle and the same for non-idle experiments?).
3. Now build a simple classifier to differentiate between two trivial actions (forward push, upward lift) and no action - and compare your model with your teammates. What kind of features did you use? Did you do it structurally (e.g. with a decision tree)? Edit either the sketch for your Arduino or your `mqtt_sub.py` to do these characterizations.
4. Attempt to build a classifier for 3 actions (+idle): the last 2 and a circular rotation motion (a non-trivial motion). Can you use the same features as before to do this separation? Are you able to track a circular rotation motion easily using an IMU? If you are, what features did you use? What kind of action might be easier, if not?
5. Please push your code onto your 180D-WarmUp repository.