Design and Implementation of Wireless IoT Sensors

Yuguang Fu
Nanyang Technological University

29-July, 14:00 - 17:15 @ TU103 & ZS1107

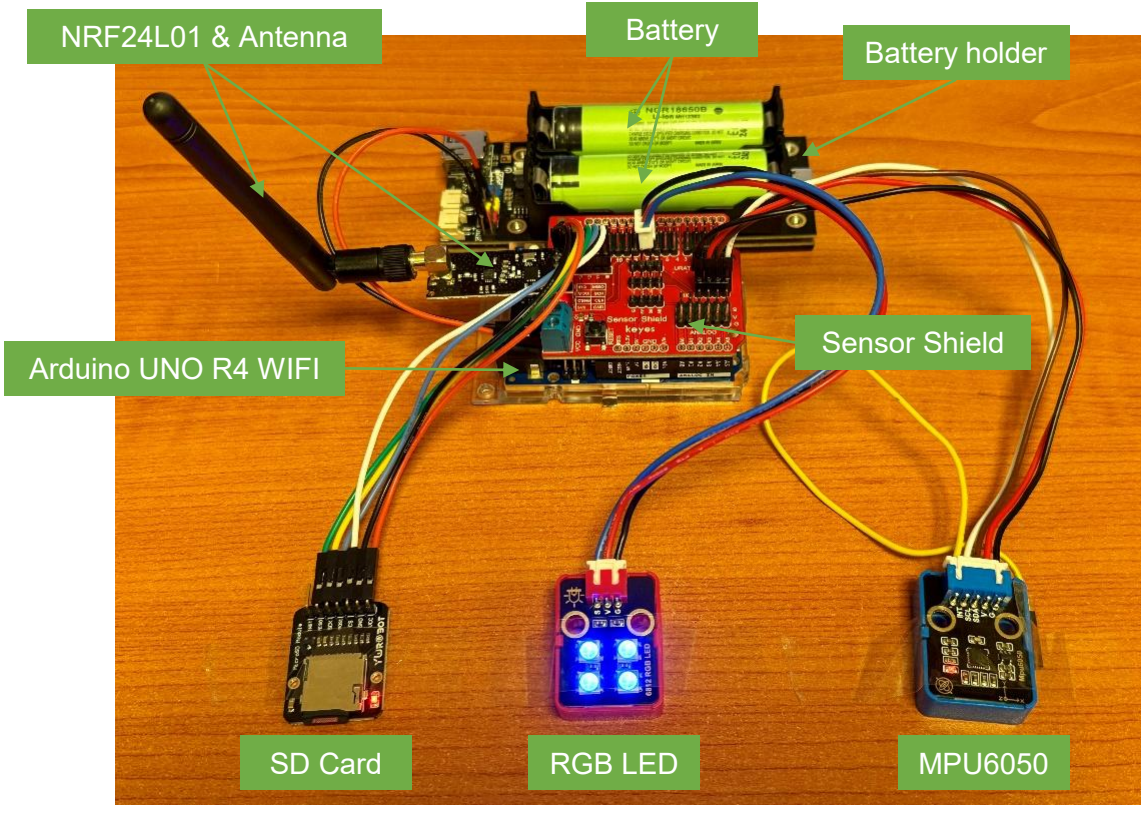
Objective

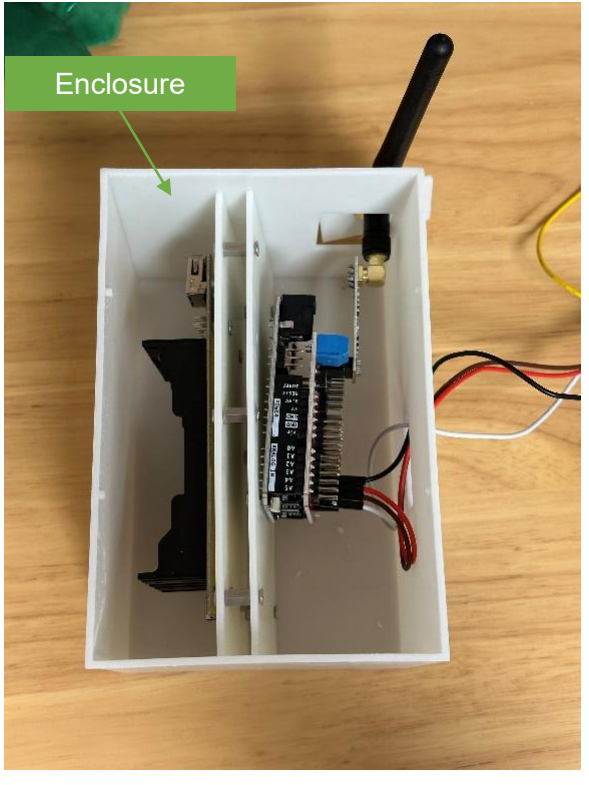
- 🎯 Assemble an IoT wireless sensor
- 🎯 Configure, compile and upload the code
- 🎯 Deploy sensors for vibration testing

Section I - Introduction

Hardware Setup

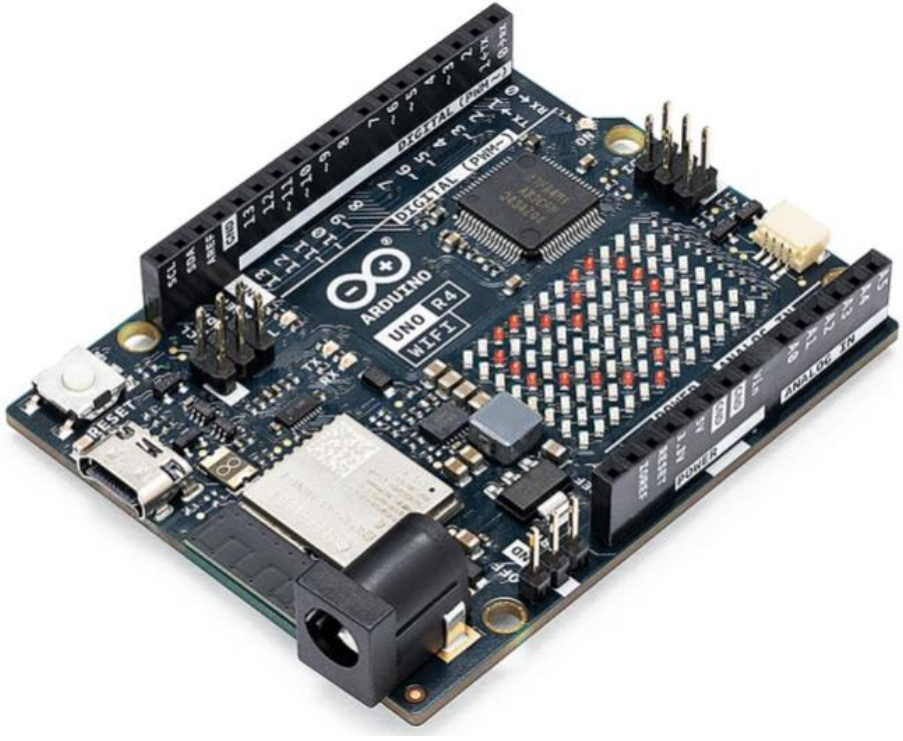
❑ Preview of the Assembled IoT Wireless Sensor





Hardware Setup

❑ Main Controller Board (Arduino Uno R4 WIFI)



The R7FA4M1AB3CFM#AA0 features: **Microcontroller**

- 256 kB flash / 32 kB SRAM / 8 kB data flash (EEPROM)
- Real-time Clock (RTC)
- 4x Direct Memory Access Controller (DMAC)
- 14-bit ADC
- Up to 12-bit DAC
- OPAMP
- CAN bus

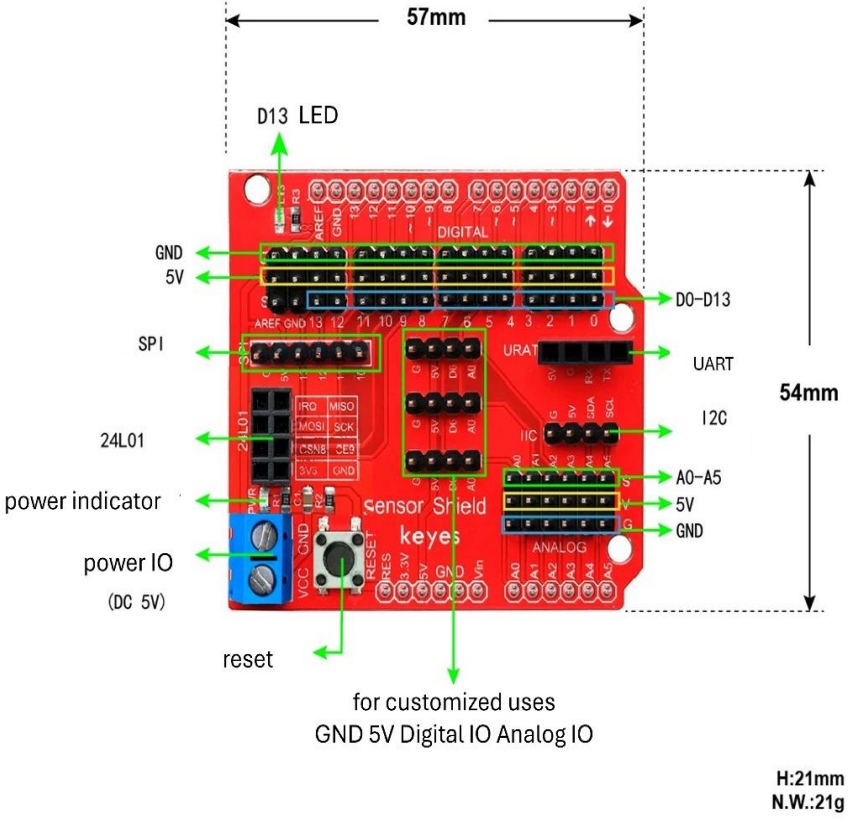
The ESP32-S3-MINI-1-N8 features: **Wi-Fi Module**

- Wi-Fi® 4 - 2.4 GHz band
- Bluetooth® 5 LE support
- 3.3 V operating voltage
- 384 kB ROM
- 512 kB SRAM
- Up to 150 Mbps bit rate

Hardware Setup

❑ Sensor Shield

An expansion board designed for connecting various sensors and other components to an Arduino board, providing a rich set of interfaces to facilitate wiring.



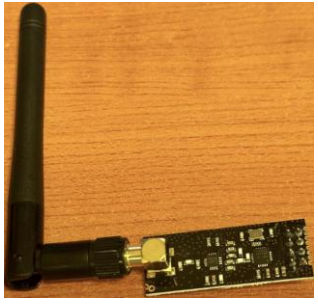
Hardware Setup

❑ Accelerometer & Radio Frequency & RGB LED & SD Card

MPU6050

- an inertial measurement unit (IMU) that integrates a 3-axis gyroscope and a 3-axis accelerometer
- 2g/4g/8g/16g
- 16 bit

Frequency radio module (nRF24L01)



- low-power, low-cost 2.4GHz wireless transceiver
- short-range wireless communication
- 250 kbps/1 Mbps/2 Mbps

RGB LED

- consists of three independent LEDs that emit red, green, and blue light
- various colors can be generated by mixing red, green, and blue

SD Card module

- an external storage device used for data storage
- communicates with Arduino via the SPI interface

Hardware Setup

❑ Power Supply & Enclosure

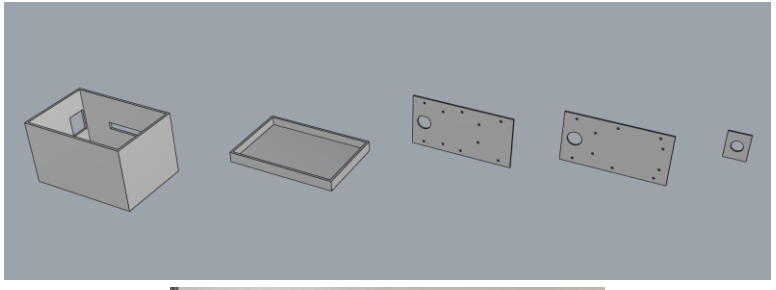
Lithium batteries

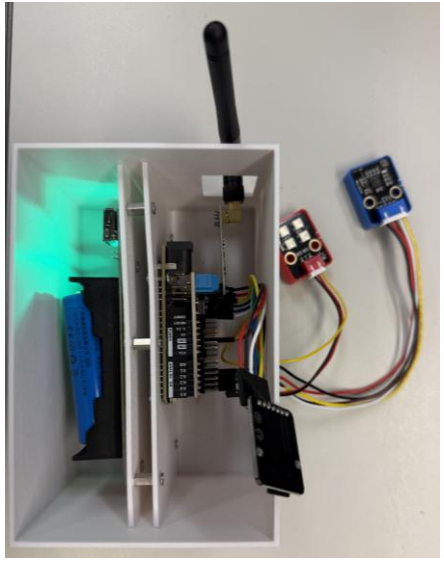
- two 18650 lithium batteries
- rechargeable, high energy density
- output voltage: 3.7V
- battery capacity: 3400mAh

Battery holder

- accommodate two 18650 lithium batteries
- provide necessary connection interface

Enclosure



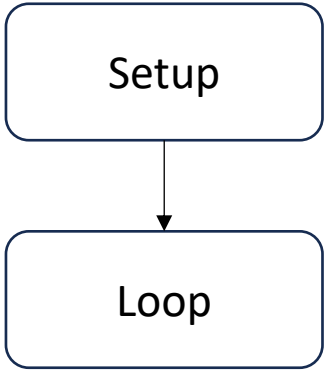


Embedded Programming

❑ Development Framework

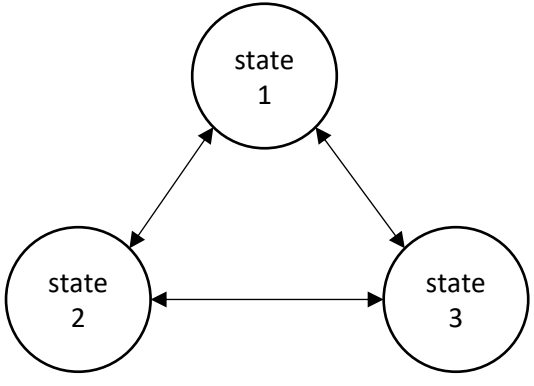
I Setup + Loop

The **'Setup + Loop'** structure forms the foundation of Arduino development. It embodies the widely adopted **"initialization + infinite loop"** paradigm used across most embedded systems.



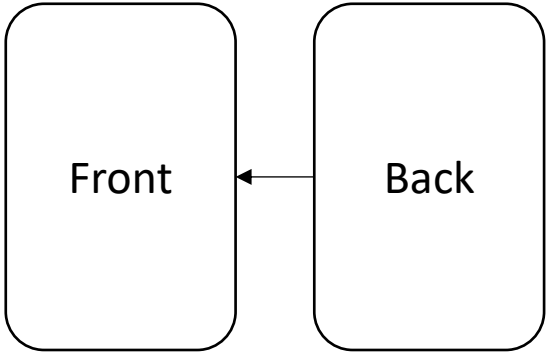
II State Machine

By using a state machine, users can effectively separate **the internal logic of each state** from **the transitions between states**.



III Front End & Back End

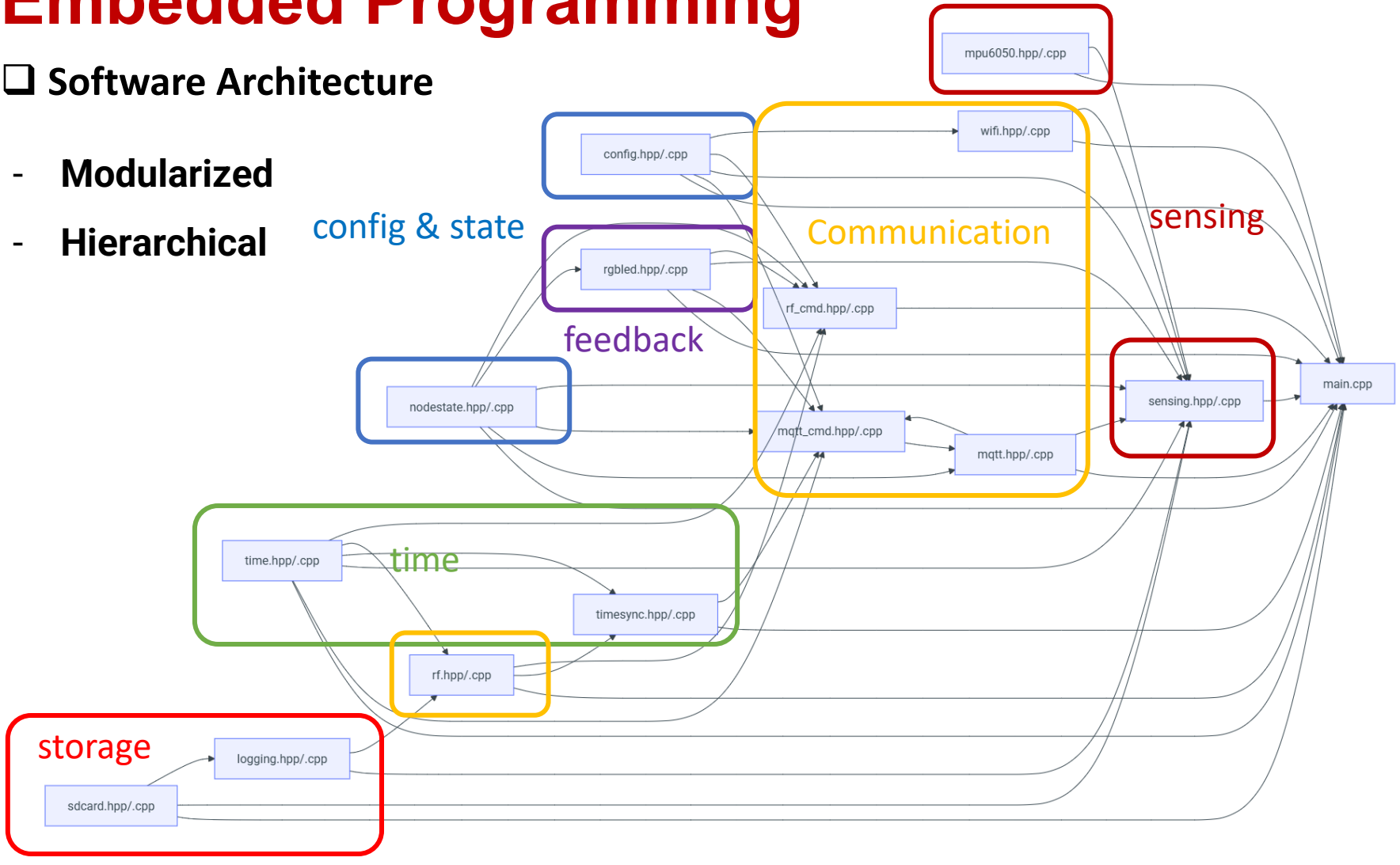
With a front-end and back-end architecture, the embedded system can support **event-driven programming**, which is highly beneficial for many applications.



Embedded Programming

❑ Software Architecture

- **Modularized**
- **Hierarchical**



Embedded Programming

❑ Configuration

To facilitate the configuration of the software, a configuration file is provided for modification as needed.

GATEWAY NODE EXAMPLE

```
#pragma once
#include <Arduino.h>

/* Node Information */
#define GATEWAY          // for main node
// #define LEAFNODE        // for sensor node

#define NODE_ID 100      // GATEWAY should be 100
// #define NODE_ID 1 // for LEAFNODE: 1, 2, 3, 4
// #define NODE_ID 2
// #define NODE_ID 3
// #define NODE_ID 4

#define NUM_NODES 4 // Total number of nodes in the network

/* WiFi Credentials */

#define WIFI_SSID "Shaun's Iphone"
#define WIFI_PASSWORD "cshw0918"

/* MQTT Configurations */
#define MQTT_CLIENT_ID      "GATEWAY"
// #define MQTT_CLIENT_ID      "LEAFNODE1"
// #define MQTT_CLIENT_ID      "LEAFNODE2"
// #define MQTT_CLIENT_ID      "LEAFNODE3"
// #define MQTT_CLIENT_ID      "LEAFNODE4"
```

LEAFNODE EXAMPLE

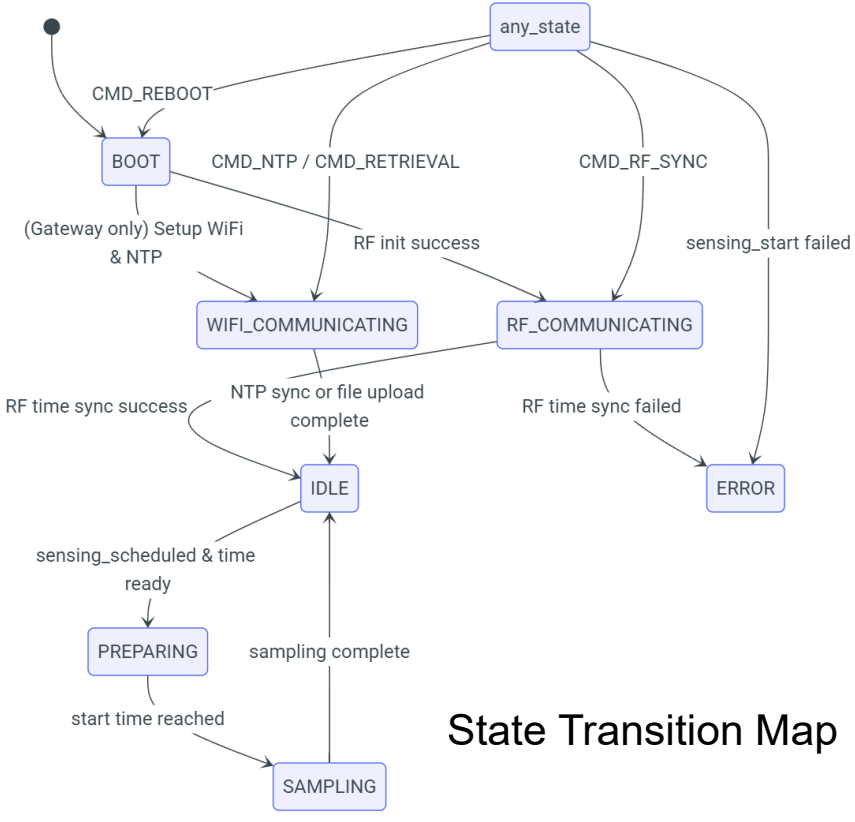
```
#pragma once
#include <Arduino.h>

/* Node Information */
// #define GATEWAY          // for main node
#define LEAFNODE        // for sensor node

// #define NODE_ID 100      // GATEWAY should be 100
#define NODE_ID 1 // for LEAFNODE: 1, 2, 3, 4
// #define NODE_ID 2
// #define NODE_ID 3
// #define NODE_ID 4

#define NUM_NODES 4 // Total number of nodes in the network

/* WiFi Credentials */

#define WIFI_SSID "Shaun's Iphone"
#define WIFI_PASSWORD "cshw0918"

/* MQTT Configurations */
// #define MQTT_CLIENT_ID      "GATEWAY"
#define MQTT_CLIENT_ID      "LEAFNODE1"
// #define MQTT_CLIENT_ID      "LEAFNODE2"
// #define MQTT_CLIENT_ID      "LEAFNODE3"
// #define MQTT_CLIENT_ID      "LEAFNODE4"
```

Handson – TASK 1

Embedded Programming

❑ State Machine and Indicators

Without the introduction of a real-time operating system, a state machine is a common method for implementing complex logic. It manages system behavior by defining states and transitions between states.



State Transition Map

nodestate.hpp

```cpp
#pragma once

// === Mutually exclusive node states ===
enum class NodeState
{
    BOOT,
    IDLE, // routine operation & monitoring
    PREPARING, // no routine operation, preparing for sensing
    SAMPLING, // actively sampling data
    RF_COMMUNICATING, // communicating with other nodes via RF
    WIFI_COMMUNICATING, // communicating with server via WiFi
    ERROR // error state
};
```

Embedded Programming

❑ Communication

Internet Comm. - WIFI

In this project, we use the onboard WIFI module to connect to the Internet, which is the foundation for NTP time synchronization and MQTT remote command control.

Internet Comm. - MQTT

Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol designed for low-bandwidth, high-latency, or unreliable networks. It is commonly used for communication between Internet of Things (IoT) devices.

Local Comm. – Radio Frequency

RF communication refers to the technology of transmitting information through radio waves. It is widely used in radio, television, mobile phones, satellite communications and other fields. The basic principle of RF communication is to modulate information onto RF signals and transmit and receive them through antennas. This project uses nRF24L01 wireless module for RF communication.

Embedded Programming

❑ Time

Natural Timekeeping – for human

Natural timekeeping refers to using calendar and clock formats familiar to humans to represent time. This usually includes fields such as year, month, day, hour, minute, and second.

Unix Time – for machine

Unix time refers to the number of seconds that have elapsed since January 1, 1970, 00:00:00 UTC. It is a standard time representation widely used in computer systems. The advantage of Unix time is that it is easy to compute and compare, although it is not easily readable by humans.

Runtime – for local timetracking

Runtime refers to the duration from system startup to the current moment, usually measured in milliseconds. It is particularly important in real-time systems and embedded devices, as it helps monitor system status and performance. Most hardware platforms provide a timer to track runtime.

❑ Time Synchronization

Global Time Sync - NTP

NTP (Network Time Protocol) is used to synchronize local device time with an accurate time source on the internet.

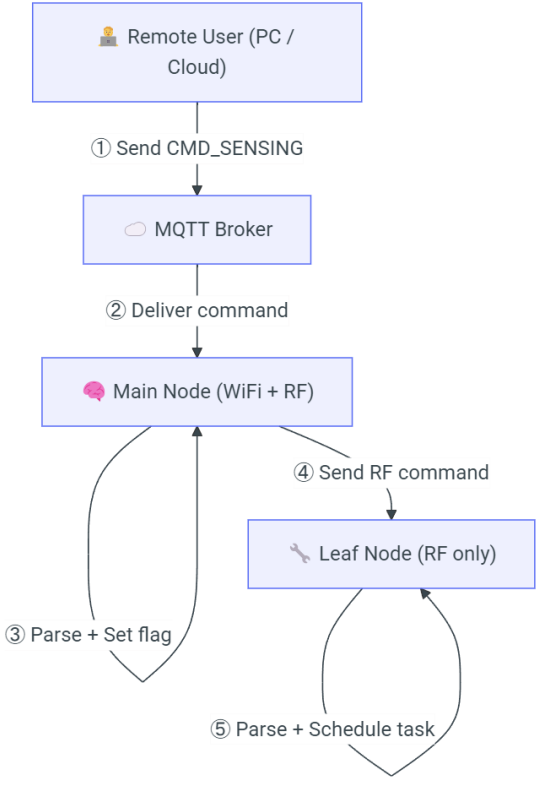
Local Time Sync - FTSP

This project implements a flooding-based time synchronization protocol (FTSP), where the gateway node periodically broadcasts its current time to all leaf nodes. Each leaf node receives the messages and calculates timing offset and clock drift to align its local time accordingly.

Embedded Programming

❑ Command

In this project, we achieve node control over the internet based on the MQTT callback mechanism.



1. Reboot
2. NTP synchronization
3. RF synchronization
4. Sensing command
5. Data retrieval command

❑ Feedback - RGBLED

Associate with the state machine, different states are indicated by different colors of the RGBLED.

rgbled.hpp

```
#pragma once

#include <Arduino.h>
#include <FastLED.h>
#include "nodestate.hpp"

#define NUM_LEDS 4
#define LED_PIN 7

extern CRGB leds[NUM_LEDS];

void rgbled_init();
void rgbled_set_all(CRGB color);
void rgbled_clear();
void rgbled_set_by_state(NodeState state);
```

Embedded Programming

❑ Acceleration Sensing, Storage, and Upload

Since Arduino performance is very limited, this project uses a method of **sampling and storing at the same time** to achieve data collection.
Since there is no real-time operating system, the storage process will have a certain impact on sampling, so a high sampling frequency cannot be achieved. The highest sampling rate in this project is 250Hz.

sensing.hpp

```
#pragma once

#include <stdint.h>

#define SENSING_PREPARING_DUR_MS 5000  // Duration for preparing sensing in millise

typedef struct {
    uint16_t elapsed_ms;  // Elapsed time since sensing started (ms)
    int16_t ax;
    int16_t ay;
    int16_t az;
} SamplePoint;

bool sensing_prepare();                 // Called once at the beginning of PREP
void sensing_sample_once();             // Called repeatedly during SAMPLING st
void sensing_stop();                    // Called once at the end of SAMPLING s

void sensing_retrieve_file();           // Retrieve file from SD card
```

Handson – TASK 2

Section II – Hands On

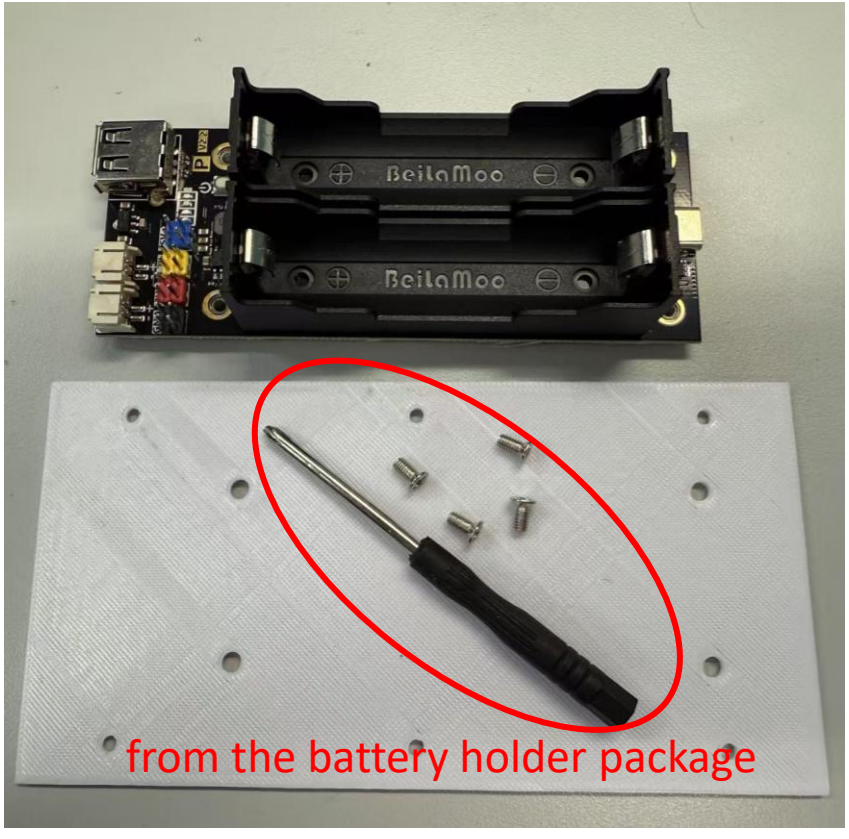
TIP: This is a team effort — please work together with your teammates to speed up the process.

Hardware Setup

- **What is in the bag?** Contact the helper if any component is missing

Hardware Setup

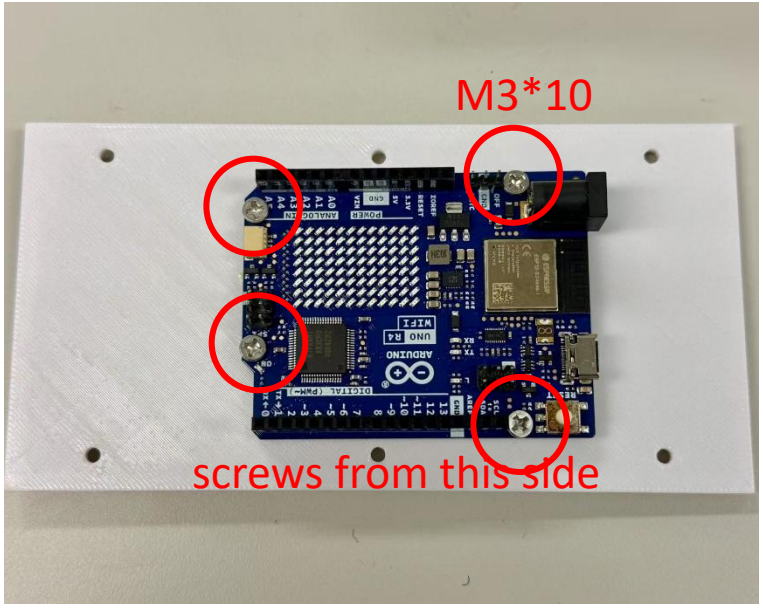
❑ 1 Battery Box Installation

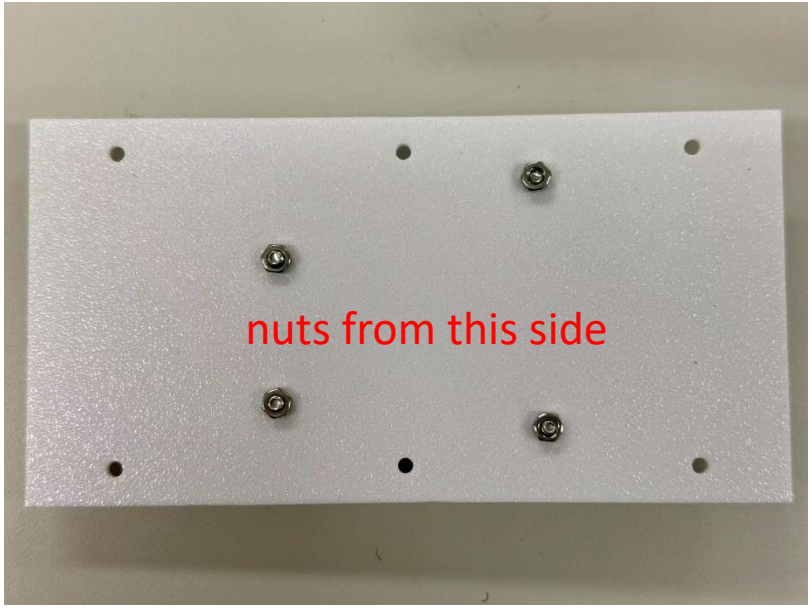


Please ensure the orientation is correct

Hardware Setup

❑ 2 Main Controller Board Installation

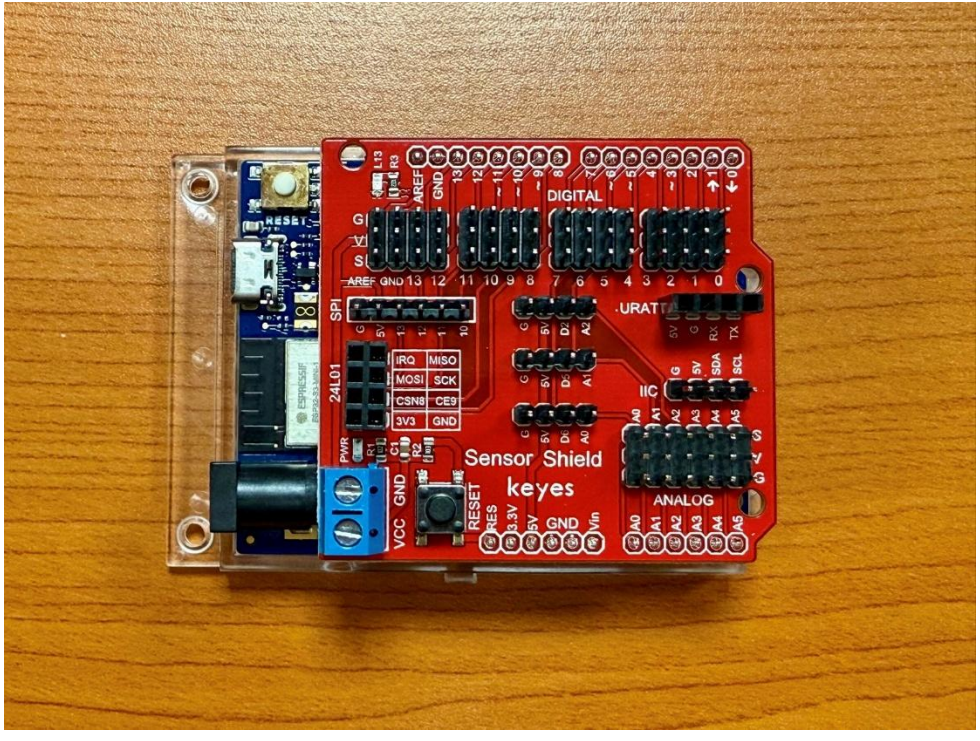




Please ensure the orientation is correct

Hardware Setup

❑ 3 Extension Board (Sensor Shield) Installation

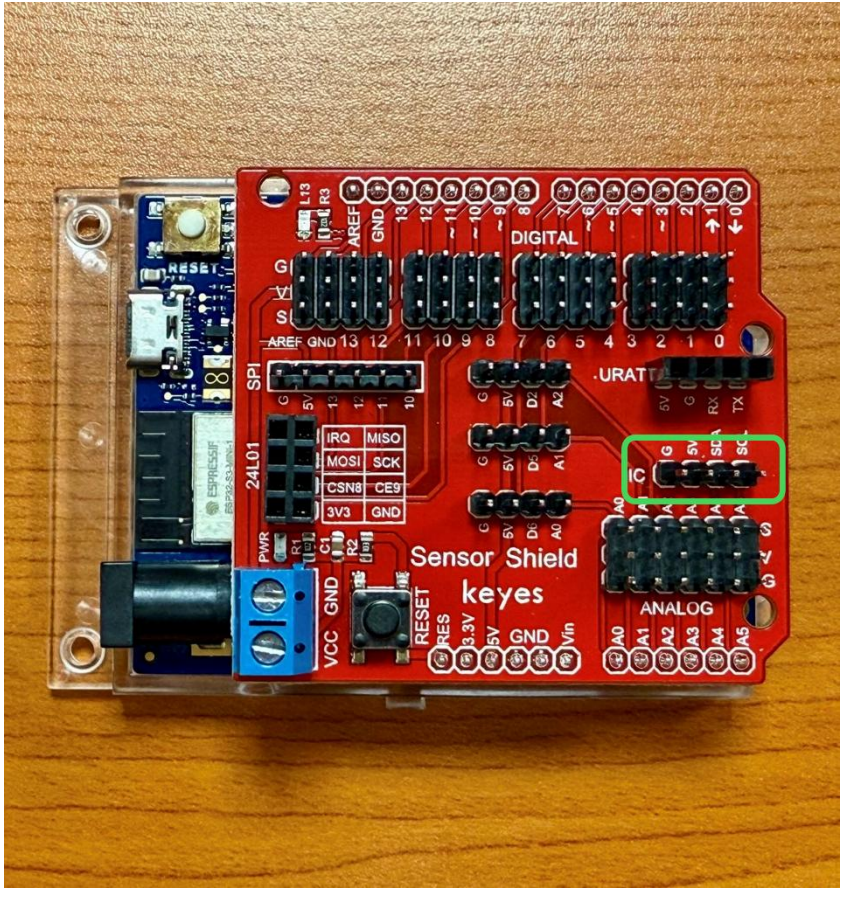


Please make sure the pins are correctly connected.

Hardware Setup

❑ 4 MPU6050 Installation

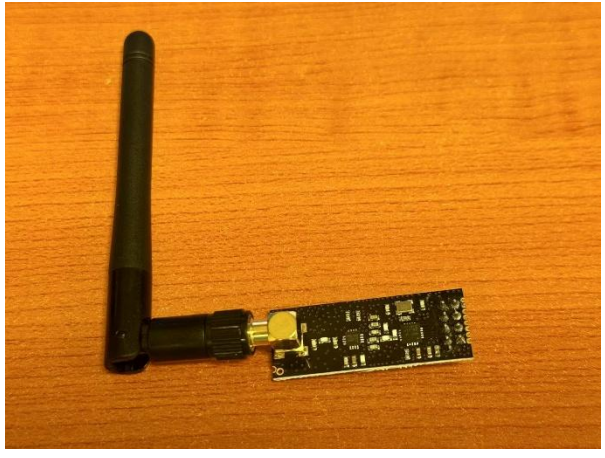
Sensor Shield Pin		MPU6050 Pin
G		G
5V		V
SDA		SDA
SCL		SCL
-		INT (not used)

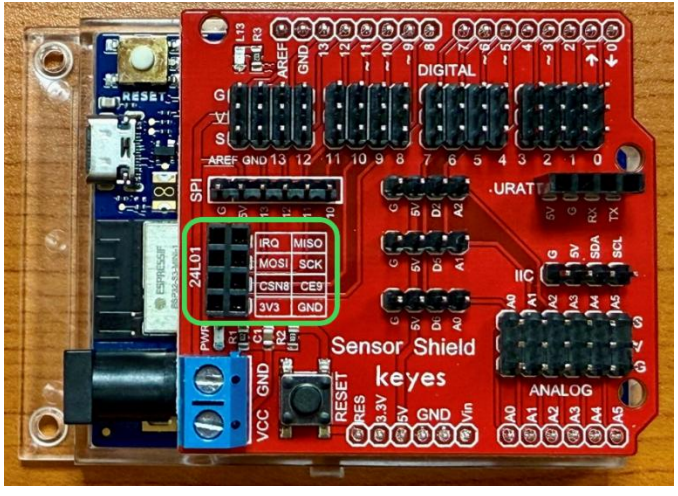


Please make sure the pins are correctly connected. INT can be left unconnected.

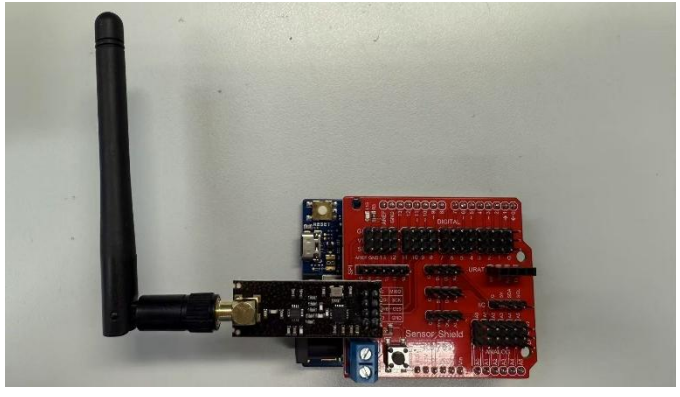
Hardware Setup

❑ 5 Radio Frequency Module Installation





Sensor Shield Pin	nRF24L01 Pin
GND	GND
3V3	VCC
CE9	CE
CSN8	CSN
SCK	SCK
MOSI	MOSI
MISO	MISO
IRQ	IRQ (optional)



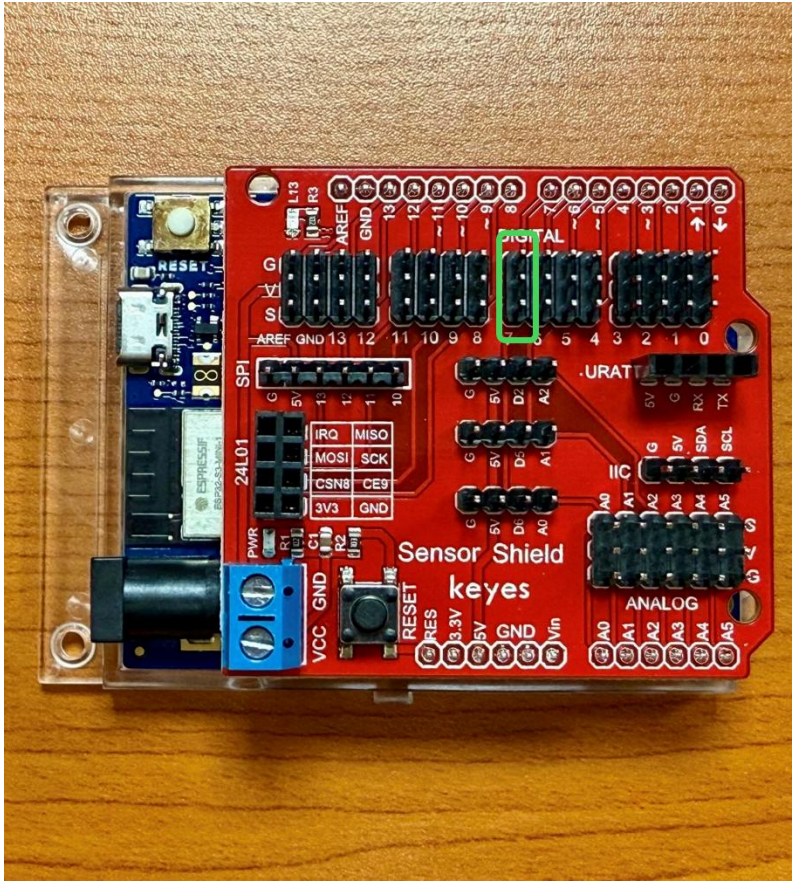
You can just attach the RF module onto the sensor board like this. ⬆

Hardware Setup

❑ 6 RGBLED Installation

Sensor Shield Pin		RGB LED Pin
G		G
V		V
S		S

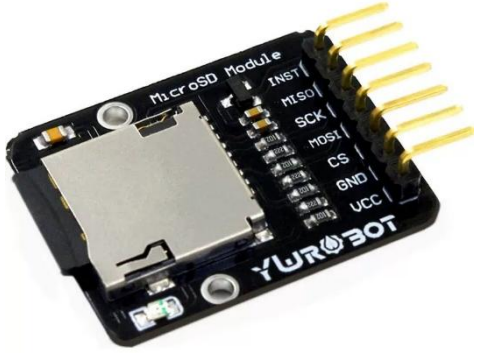
Use the associated wires in RGBLED package.

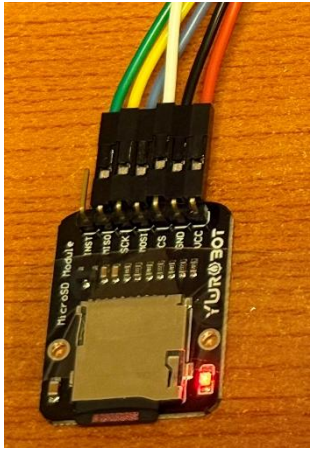


Please connect to the pins corresponding to digital 7.

Hardware Setup

❑ 7 SD Card Module Installation

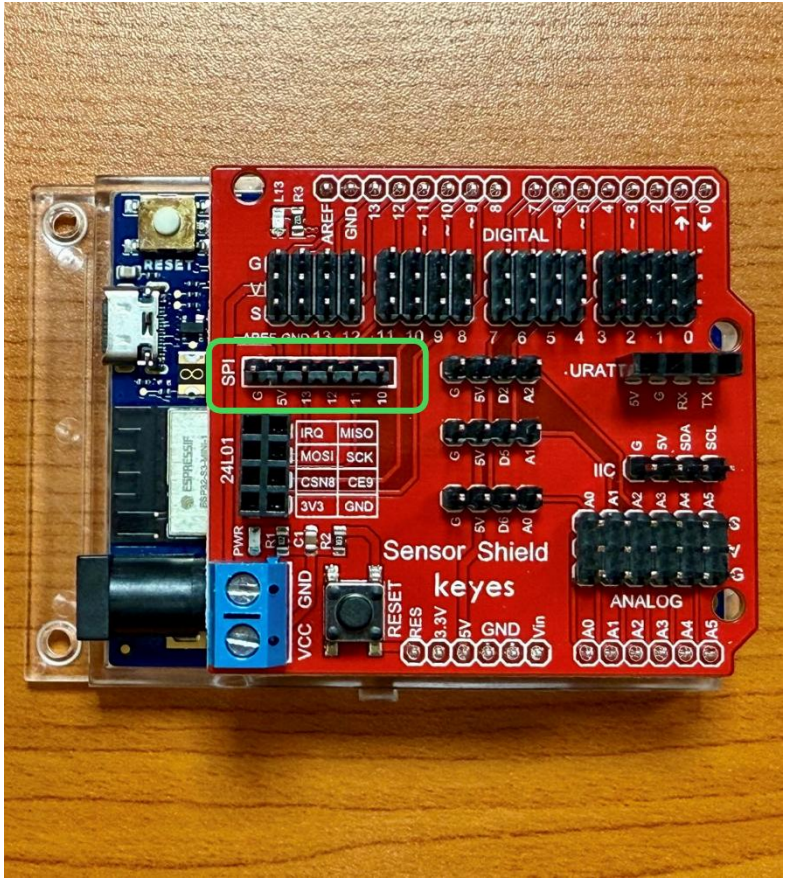




Sensor Shield Pin		SD card module Pin
5V		VCC
GND		GND
D10		CS
D11		MOSI
D12		MISO
D13		SCK

Recommended Dupont Wire Color

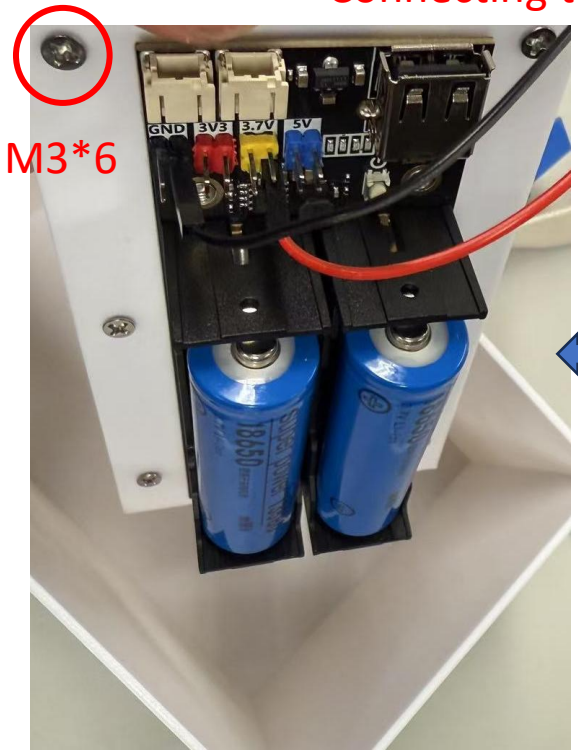
Please make sure the pins are correctly connected. Don't forget to insert a SD card.

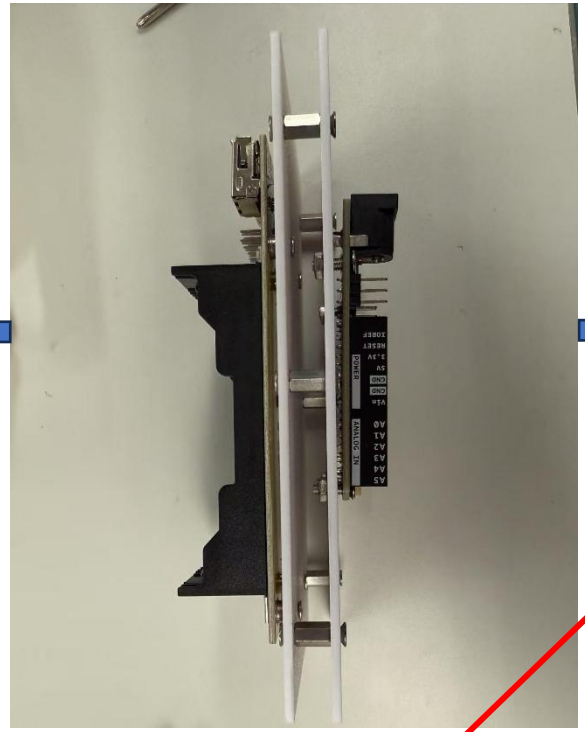


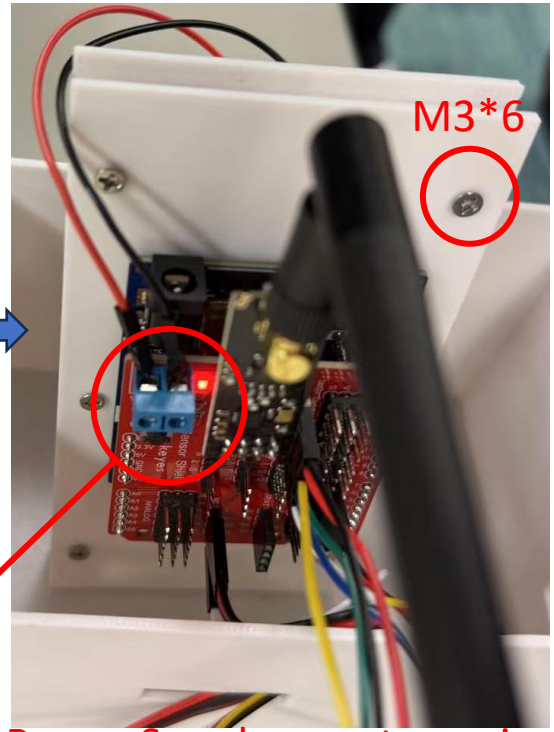
Hardware Setup

❑8 Enclosure Assembly – Better do this after programming

Connecting the two mounting boards by standoff and screws.







Power Supply over two wires.

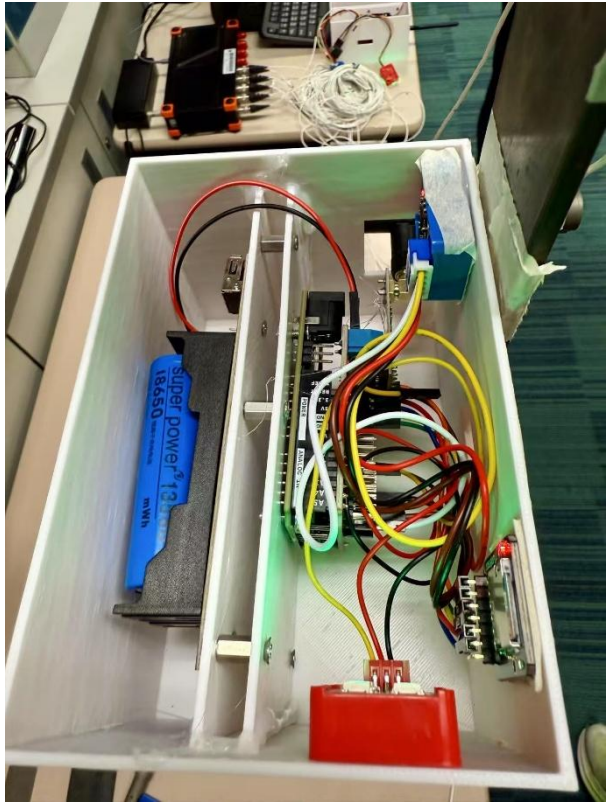
Batteries can not be fully inside.

Wires: GND and 3.7V

Carefully check all connections, then approach to the helpers for this step before powering up the board.

Hardware Setup

❑ 8 Enclosure Assembly



- You don’t need to use glue.
- Fixing component to better serve its purpose in different use cases.
- Before you power up your devices using batteries, better let our helpers to check for you incase it burns.

Embedded Programming

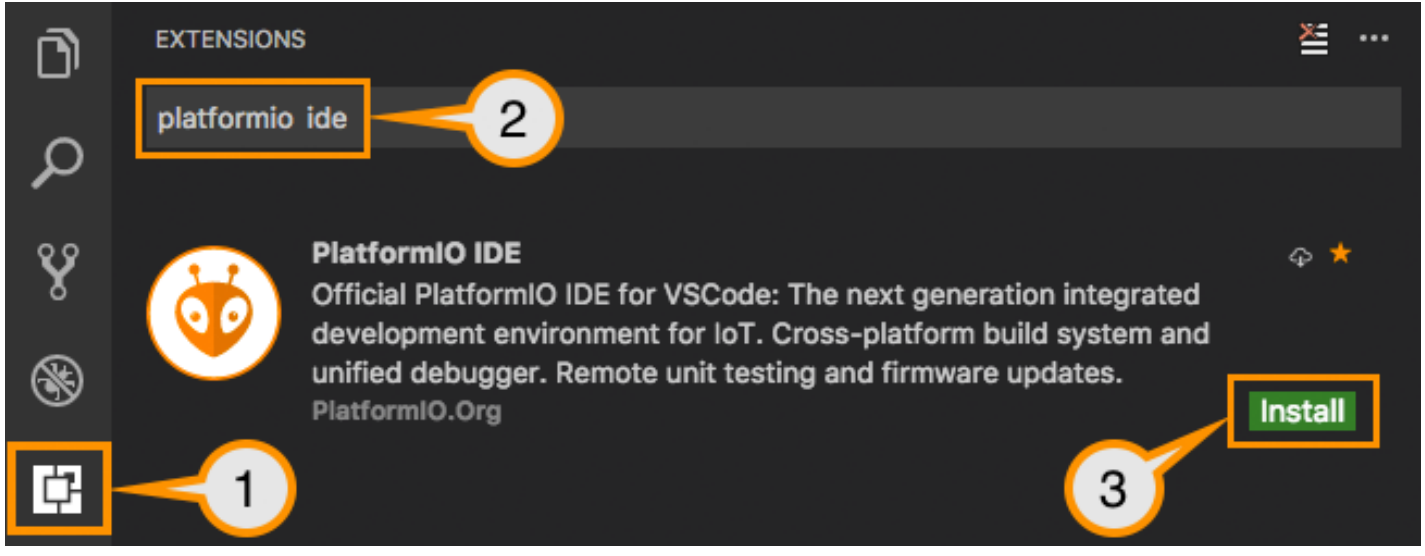
❑ 1 Preparation – 1 IDE

- Install VS Code https://code.visualstudio.com/download

Embedded Programming

❑ 1 Preparation – 2 Platformio

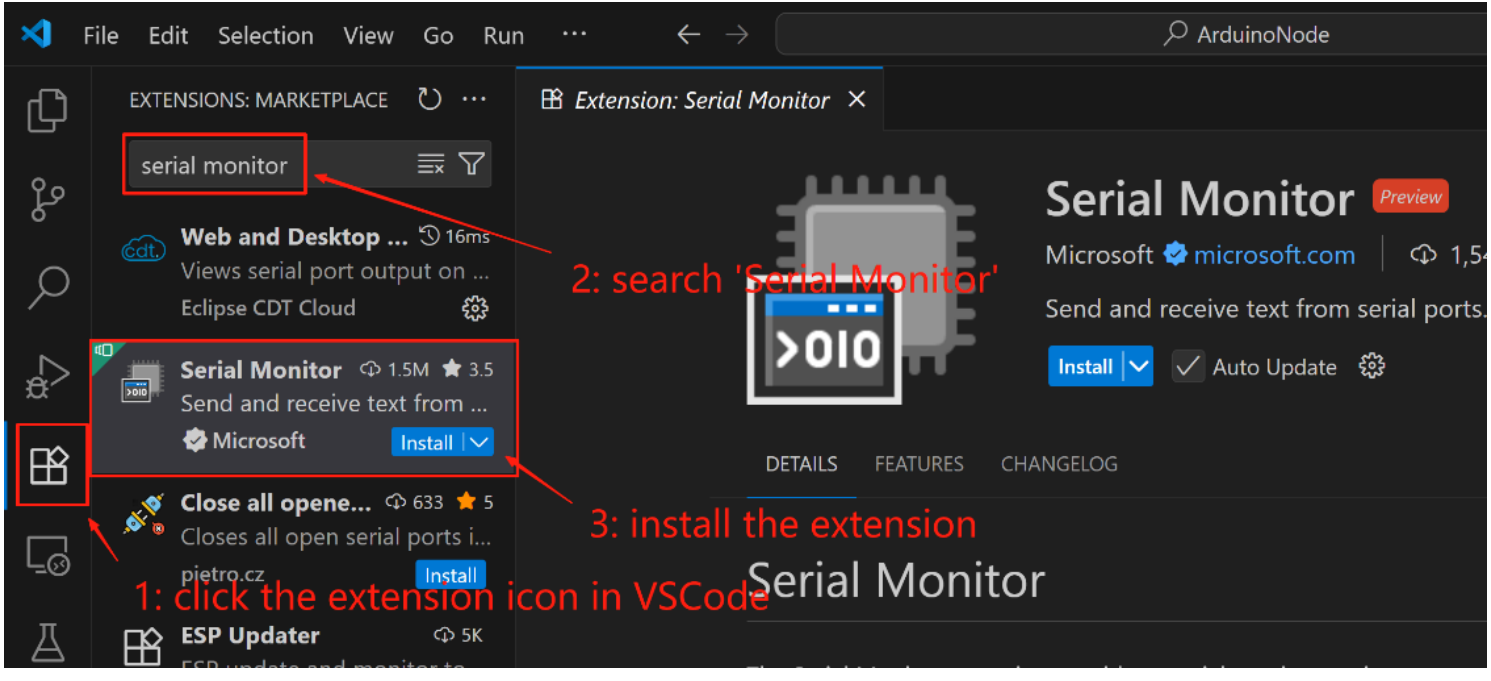
- Install PlatformIO for programming



Embedded Programming

❑ 1 Preparation – 3 Serial Monitor

- Install Serial Monitor for serial monitoring



📌Embedded Programming

❑ 1 Preparation – 4 Download Source Code

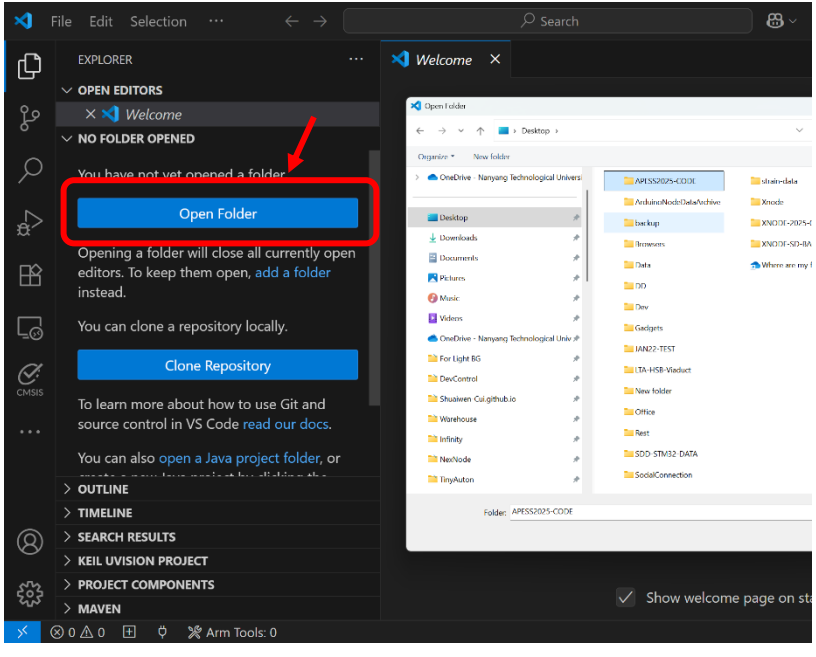
- Google Drive Link

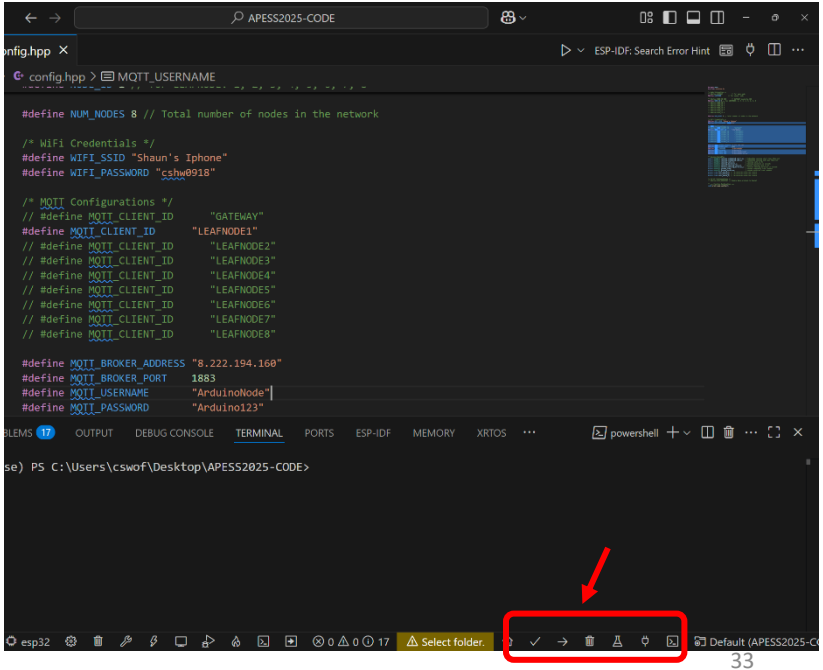
https://drive.google.com/file/d/1a3NcdvuK-aEz1CZiBiajPYCWUiWINM_Y/view?usp=sharing

Embedded Programming

❑ 2 Programming –Open the downloaded project

- ➢ Step 1: Launch VSCode
- ➢ Step 2: Open the folder where you extract the project code.
- ➢ Step 3: Ensure the Platformio environment is activated. Simply double click a code file.





Embedded Programming

❑ 2 Programming – TASK 1 Modify the Configuration File

- Step 1: Navigate to the ‘src’ folder, double click the ‘config.hpp’.
- Step 2: Modify the ‘NODE_ID’ and ‘MQTT_CLIENT_ID’ to the one corresponding to your group number. For example, group 1 Please use NODE_ID 1 and MQTT_CLIENT_ID “LEAFNODE1”, and ensure the others are commented.

```
// #define NODE_ID 10
#define NODE_ID 1 //
// #define NODE_ID 2
// #define NODE_ID 3
// #define NODE_ID 4
// #define NODE_ID 5
// #define NODE_ID 6
// #define NODE_ID 7
// #define NODE_ID 8
```

```
/* MQTT Configurations */
// #define MQTT_CLIENT_ID      "GATEWAY"
#define MQTT_CLIENT_ID      "LEAFNODE1"
// #define MQTT_CLIENT_ID      "LEAFNODE2"
// #define MQTT_CLIENT_ID      "LEAFNODE3"
// #define MQTT_CLIENT_ID      "LEAFNODE4"
// #define MQTT_CLIENT_ID      "LEAFNODE5"
// #define MQTT_CLIENT_ID      "LEAFNODE6"
// #define MQTT_CLIENT_ID      "LEAFNODE7"
// #define MQTT_CLIENT_ID      "LEAFNODE8"
```

Embedded Programming

❑ 2 Programming – TASK 2 Complete the Sensing Function

- Step 1: Navigate to the '**src**' folder, double click the '**sensing.cpp**', go the the '**sensing_sample_once**'.
- Step 2: Complete the part marked in red
  - Use the function from **mpu6050.hpp/cpp** to get **ax ay az** (**raw** data)
  - Convert the data with unit of '**g**' & calibrate the data by '**cali_scale_x/ cali_scale_y/ cali_scale_z**'. Refer to **config.hpp/cpp**. Hint: the sensor range is 2g, and the resolution is 16 bit.

```
 90,0.054443,-0.003418,0.974854
 95,0.051758,-0.003418,0.981445
100,0.054199,-0.003906,0.978760
105,0.053955,-0.007080,0.977539
110,0.053955,-0.002441,0.972900
115,0.051270,-0.002930,0.979492
120,0.054443,-0.005615,0.973877
```

Embedded Programming

❑ 2 Programming – TASK 2 Complete the Sensing Function

```
void sensing_sample_once()
{
    // Check current time
    uint32_t now_ms = Time.get_time();
    // Check if we should sample
    if (now_ms - last_sample_time >= (1000 / sensing_rate_hz))
    {
        // if yes, update the last sample time
        last_sample_time += (1000 / sensing_rate_hz);
        // Prepare the variables for reading IMU data
        int16_t ax, ay, az;
        // <Read acceleration data from the IMU, to be completed by students, refering to mpu6050.hpp and mpu6050.cpp>

        // Calculate the elapsed time since the start of sensing
        uint32_t elapsed = now_ms - t_start_ms;

        // Converting raw acceleration data to g's using the scaling factor and calibration factors
        // float ax_g
        // float ay_g
        // float az_g

        // print the data to the SD card file
        char line[64];
        snprintf(line, sizeof(line), "%8lu,%8.6f,%8.6f,%8.6f", elapsed, ax_g, ay_g, az_g);

        data_file.println(line); // print to the SD card file

        // Update the number of samples taken
        sample_count++;
    }
}
```

Embedded Programming

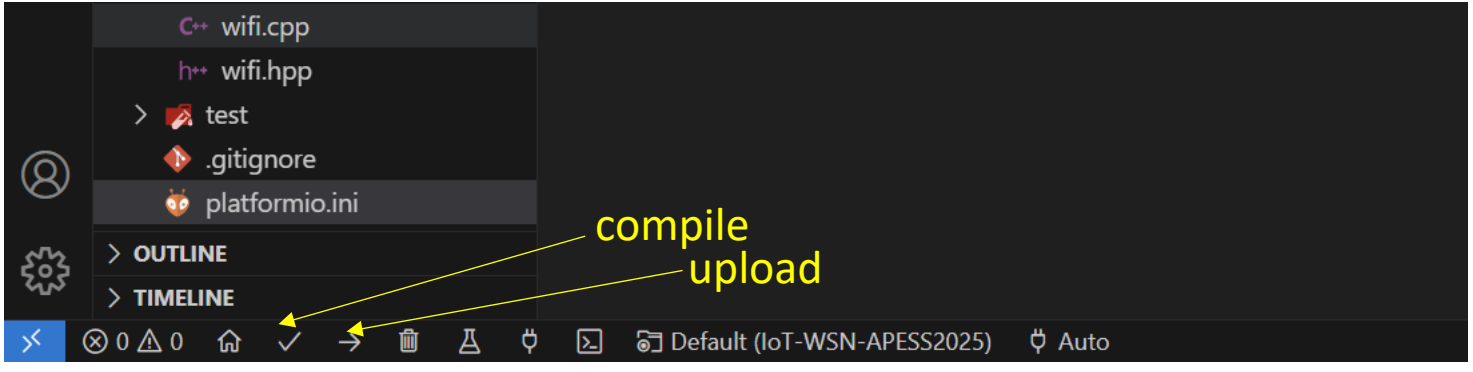
❑ 2 Programming – TASK 2 Hints

- The ‘sensing_sample_once’: Variables ax, ay, az are already created. We need to find the function to read the data from the mpu6050. Please check mpu6050.hpp and mpu6050.cpp to find one useful function to do this.
- The mpu6050 range is by default 2g, also for this project. Variables, ax, ay, az, stand for raw data. But we need to convert the data from raw data to data with unit of g. Also, we want to calibrate the sensing. Hint: the range is by dault 2g, and we need to convert the data into values with unit of g. ax_g, ay_g, az_g = scalefactors * (raw value)/(2^(16)/4) = scalefactors * (raw value)/2^(14)

Embedded Programming

❑ 2 Programming – Compile and Upload Code

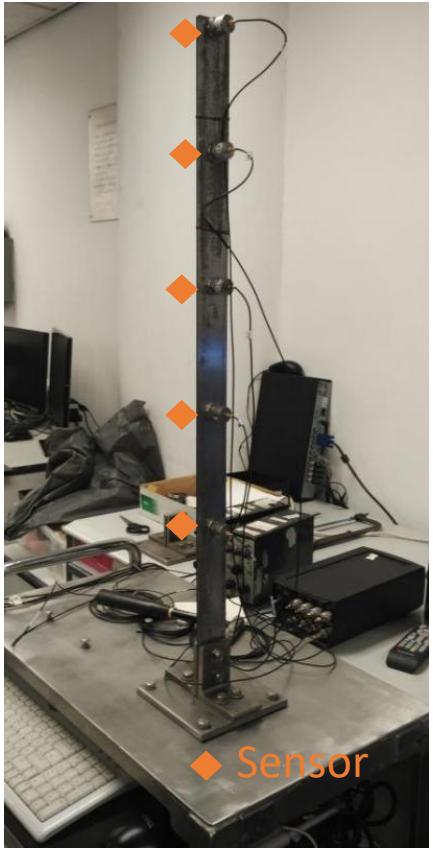
1. Build and upload code for the main node
   - Ensure you finished the tasks in previous slides
   - Click the build button, wait for the build to succeed
   - Connect one sensor node to your computer via USB, click the upload button, and wait for the upload to finish



2. Testing
   - Launch the Serial Monitor (baud reate 115200) to check the output. Press reset for rebooting after building up the serial connection.
   - Check the RGBLED for debugging:

     white for a long time – booting error; red – error; light blue – rf communication; blue – wifi communication; green – idle/ready; yellow – preparing for sensing; purple - sensing.

Indoor Vibration Test (ZS1107)



Cantilever Beam

Task:

1) Deploy wireless sensors on the beam and obtain the vibration accelerations 100Hz 30s

2) Plot the time history data and PSD. Calibrate the scaling factor for each axis.

- Excited by the hamper at the end (Tap gently to avoid exceeding the range 2g for Arduino node)
- 2~3 sets

Outdoor Vibration Test (TBD)

Footbridge

Task:

1) Deploy wireless sensors on the footbridge and obtain the vibration accelerations

2) Identify modal parameters leveraging the obtained vibration signal

- Excited by 1) jumping near the midspan and 2) the environment
- Each for 3 sets

Thank you for your attention
Q&A

Yuguang Fu
Nanyang Technological University

29-July, 14:00 - 17:15 @ TU103 & ZS1107