(Sensor Suite) Operation Guide

Team 20

Team members: Yousef Alkhelaifi, Calvin Xaybanha, Yikun Wang, Yudi Bao, Brandon Garcia

11 June 2021

Version 1.0

Outline

1. Introduction

1.1 Purpose

The purpose of our product is to create a low power wireless data logger that measures the environmental data within a network(SmartMesh IP), and displays this data to the user in a meaningful way.

1.2 Product Background

In 2020, there were more than 2 million farms in the United States, with a total area of 897,400,000 acres. However, the number of farmers is very small. It means that on average each farmer needs to manage hundreds or even thousands of acres of farmland. If American agriculture does not implement production mechanization and intelligence, farmers cannot have enough energy and resources to fertilize, water, harvest crops. This is because factors such as weather, soil, pests reduce crop yields. Once farmers cannot recognize and stop these disasters in time, they are likely to suffer huge economic losses.

` In order to provide more convenient management of farmland, and a more rapid response to all kinds of natural or assumed disasters, farmers need a sensor detection system to ensure the normal operation of the farmland. For this system, there are several chips having different sensors and are connected to a network at the same time. By placing them in different areas, the temperature, humidity, oxygen/carbon dioxide, wind direction and other indicators of each area can be automatically detected in real time. Therefore, the system can help farmers to ensure the environment for each crop and keep those crops under the best possible growing conditions.Finally farmers can obtain the highest profit with the lowest cost.

1.3 Reference

Smartmesh API code, Smartmesh GUI code, PCB schematic and layout, sensors code, and I2C code

(https://github.com/ECE-412-Capstone-Sensor-Suite/Team-20-Sensor-Suite)

2. Hardware

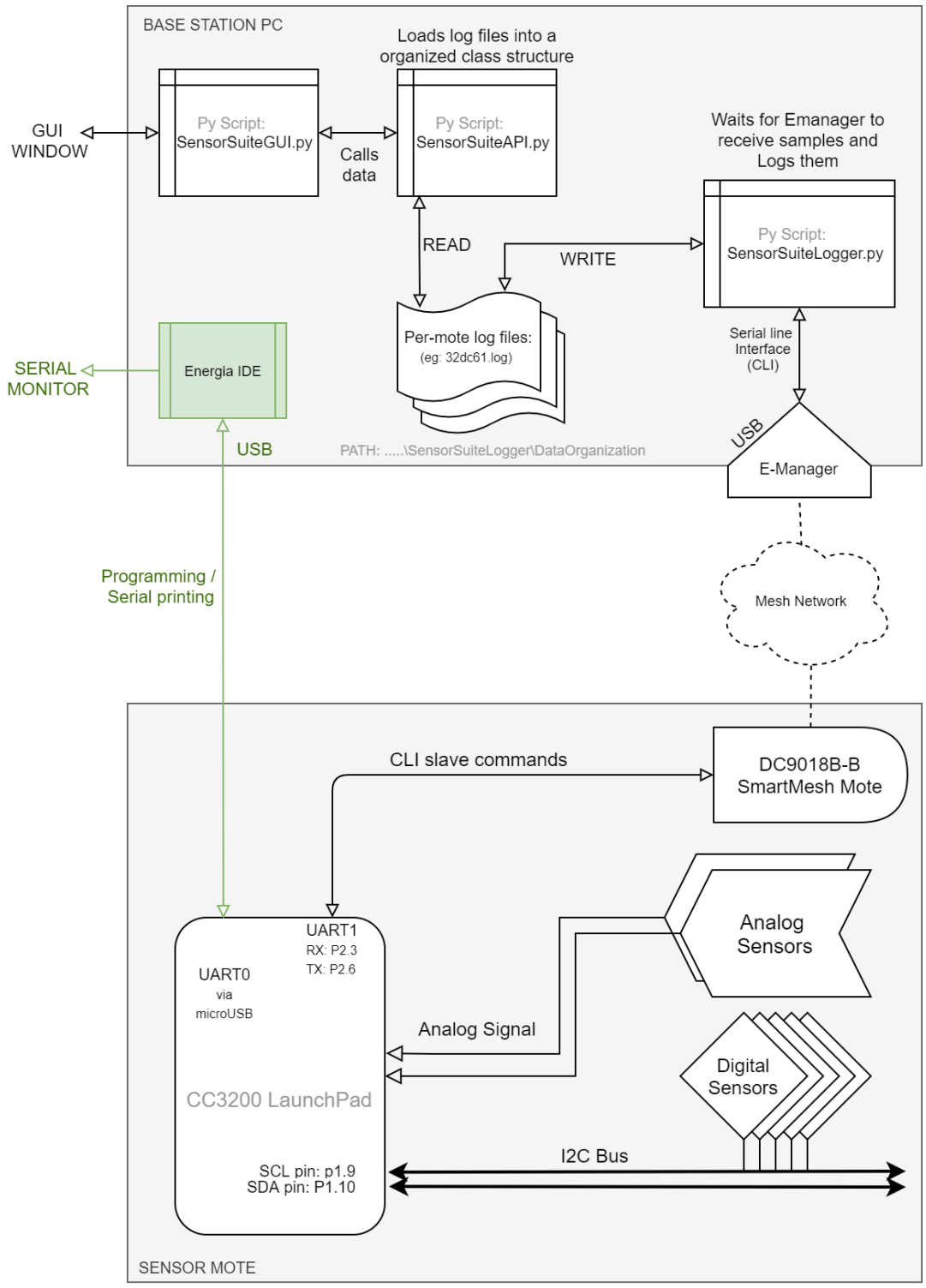
2.1 List of Hardware

- Microcontroller
    - Texas instruments - CC3200: Sensors communication
- RF wireless Hardware
    - DC2274A-A: E-manager access point
    - Evaluation Mote DC9018B-B
        - Contains: LTP 5902 SmartMesh IP - Wireless network module
    - DC9006A: Eterna interface card
    - DC9010B: Eterna Serial programmer
- Digital Sensors
    - DFRobot SEN0322: Oxygen Sensor
    - Noyito OPT3001: Ambient light sensor
    - ADXL345: Accelerometer
    - Honeywell HIH8120-021-001: Temp/Humidity Sensor
- Analog Sensors
    - Wind Sensor Rev. C: Wind Sensor
    - Water Contact Sensor: Grove-Water Sensor V1.1
    - DFRobot SEN0219: CO2 Sensor

2.2 Function & Performance

Hardware	Serial Communication	Performance
SmartMesh IP - LTP 5902	Analog	Extremely reliable wireless connection using mesh network technology 0-100 motes with single Emanager +1000 notes with multiple access points
Texas instruments - CC3200	I2C	
HIH8120	I2C	±2.0 %RH (humidity) ±0.5 °C (temp) -40C - 125C
Noyito OPT3001	I2C	wavelength 550 nm Low operating current: 1.8 μA Operating temperature range: –40°C to +85°C 5.5 V tolerant I/O
DFRobot SEN0219	Analog	Measuring Range: 0 ~ 5000ppm. Accuracy: ± (50ppm + 3% reading)
DFRobot SEN0322	I2C	Measurement Range: 0~25%Vol, Stability: <2%, Response Time: ≤15 seconds
ADXL345	I2C	4 mg/LSB on +- up to 16 g
Wind Sensor Rev. C	Analog	Wind speeds 0-60mph, 4-5V
Water Contact Sensor	Analog	Low power consumption High sensitivity

2.3 Prototype Diagram



*Green blocks are Debug/Evaluation parts that are not necessary for normal system functionality.

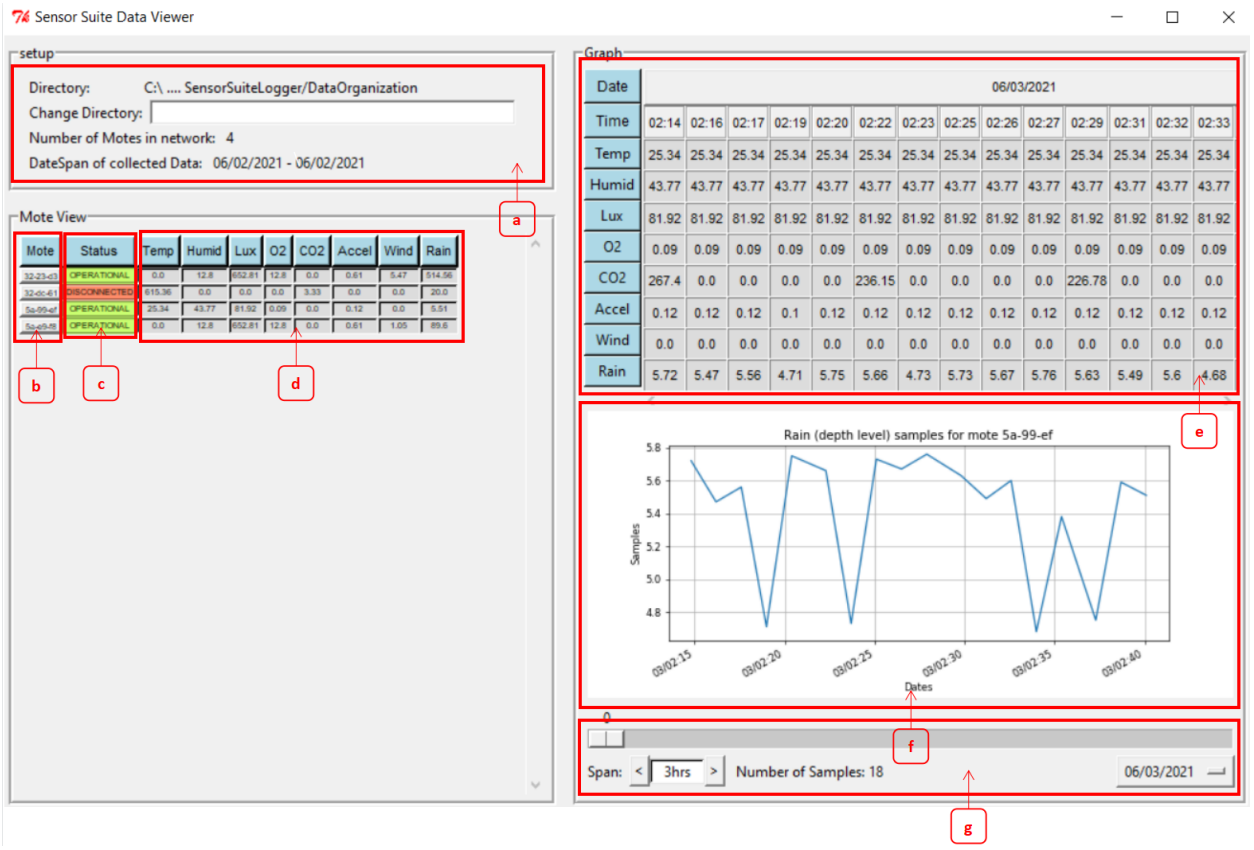
3. Software

3.1 Goal/Purpose

This project has a corresponding software that is SensorSuiteGUI interface. It could help users to check and monitor data from sensors in real time. It also provides mathematical analysis(Plots) to help users to see how the environment changes.

3.2 Function & Performance

SensorSuiteGUI interface has high speed and frequency to fetch the data from the sensors. It makes users see the data and corresponding plots very fast. High integration between software and hardware create a reliable monitoring system.



(Functional diagram)

a: Directory model -- Users could change the directory path where all data history store

b: Mote list -- Here are all motes. Users could click anyone so that history of selected mote and plot of it can be showed on right half interface

c: Current status of motes -- Users could check the launch status of motes here. Red means disconnecting, and Green means connecting.

d: Latest environment data -- Users can know the latest environment data here.

e: history of data (usually unit by day) -- Here is history of one selected mote by users. Users could check all history data about selected mote.

f: mathematical plots -- Users could see how the concrete environment condition changed here by clicking the blue button in the history block.

g: Condition filter & Sample counter -- Users can change date or time search scope here, and also could see how many samples for plot.

3.2 Sensor Suite API Guide

Overview:

Sensor suite API gives you the ability to load and parse Data from the log files. The sample is logged into the log files in a CSV format. When the samples are loaded from the log files they are converted from string into float values and then divided by 100 in order to move the decimal point up two digits to match the precision the data was measured in. Here we will explain the data logging format as well as Class structure used to hold the data in ram.

Data Log Format “one log per mote”:

Each mote has a unique Mac address, moreover each mote has unique meta data such as status, user assigned description/ID and Coordinates in case a mapping functionality would be implemented in the future. Creating one file per mote allows us to integrate this unique meta data for each mote in an easy readable format, the way it works is the first block of text at the beginning is metadate, and after this initial block all the sample data is stored in a CSV format.

There is one issue currently and that is we didn't know about the Unix timestamping method until after we created the logging script. As a result we use the ‘--’ marker to place dates in the string format every now and then in the logs, using the Unix time stamping method would have allowed us to put both time and date as a simple number that represents the the seconds since 1970/1/1, but we didn't have time to implement it into the logging script:

- Log format Example:

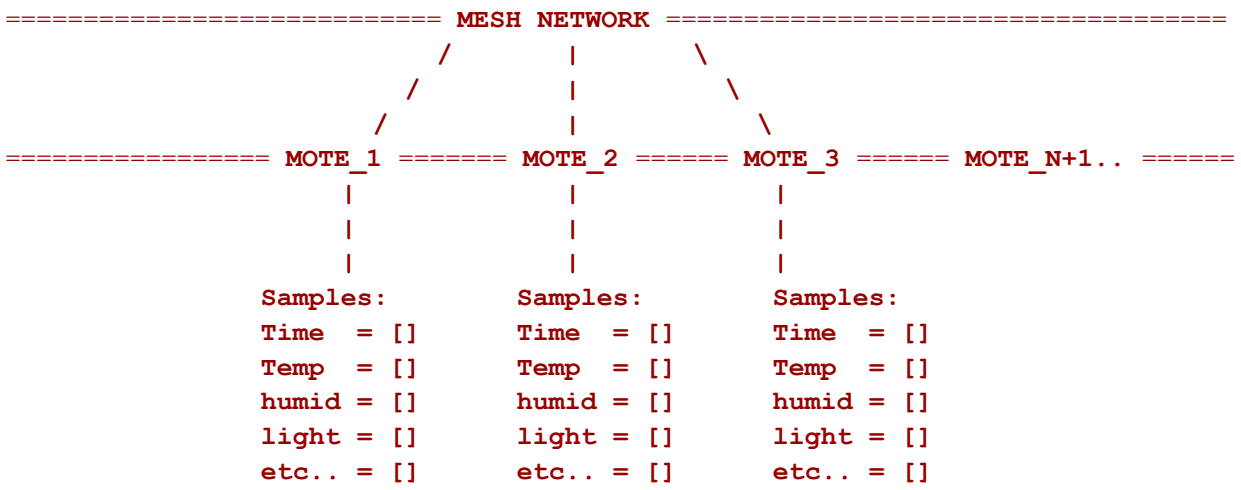
```
~ MAC: 00-17-0d-00-00-32-dc-61
~ Status: OPERATIONAl
~ Coord: (None,None)
~ User ID: None
"~" Marks Unique mote metadata

-- 05/17/2021
“--” Marks the dates of all samples collected from this point till the next Date marker

{TIME}, {Temp}, {Humid}, {Lux}, {O2}, {CO2}, {Accel X}, {Accel Y}, {Accel Z}, {Windspeed}, {Rain} //<-- CSV format
19:38:29, xxxxx, xxxxxx, xxxx, xxxxx, xxxxx, xxxxxxx, xxxxxxx, xxxxxx, xxxxxx, xxxxx
19:39:02, 61536, 0, 0, 0, 0, 0, 0, 0, 0, 2758
19:40:03, 61536, 0, 0, 0, 0, 0, 0, 0, 0, 2759
…. more CSV samples....
….
….
-- 05/18/2021                                    // <------ New Date Marker
19:38:29, 61536, 0, 0, 0, 8384, 0, 0, 0, 0, 2834
19:39:02, 61536, 0, 0, 0, 0, 0, 0, 0, 0, 2758
19:40:03, 61536, 0, 0, 0, 0, 0, 0, 0, 0, 2759
. more CSV samples….
```

Python Database Structure:

_____In order to use this logged data we need a way to convert it back from string to floating point values. Moreover it would be helpful if we can also parse the data in order to quickly and easily find sections of data of a particular type. *SensorSuiteAPI.py* gives us this ability, it all allows use to index all that data along multiple corresponding arrays, one array for each sensor type along with one array for the time stamps, the index of the timestamp array is time synchronized with each sensor array, meaning calling temp[x] and timestamp[x] you would get a temperature value and the timestamp that that temperature value was sampled.



- **MESH NETWORK:** The top level Class is the MeshNetwork Class, Fundamentally it Holds an array of Mote objects as well as the other metadata concerning the entire MeshNetwork. Like the directory all the files are stored in and the last time they were updated. Needs logfile directory as the only arg when instancing
    - **Functions:**
        - LoadMesh(): Instances a Mote object for each log file in the directory and places them in an array, then loading data for each mote from its log file using loadMote() function within the Mote class.
        - UpdateMesh(): Checks the Metadata of all the files in the directory to see if any of them have been modified since the mesh was loaded, and if it finds a log file that was modified it reloads it. This function needs to be called at regular intervals from another script in order to function properly.
        - UpdateStatus(): If a motes log file hasn't been modified in 10 mins it opens it and writes it as DISCONNECTED, if it has been modified it rewrites it as OPERATIONAL.(Warning this funx is somewhat buggy and will not update status if modifications happen while the GUI is closed)
        - SetAlertTriggers() & CheckAlerts(): the set function sets the limits for the alerts, the arguments are (sensor string , (low limit, high limit)), the limit arguments come in the form of a two element tuple using float values. The check function simply checks the

latest sample against the limits and returns an array that describes the exact limits broken by the samples.
    - EX:
    - > SetAlertTriggers(‘temp’ , (5, 40)) // Set low and high celsius limits
    - > CheckAlerts() // you must periodically call this function.

- **MOTE:** The Second lowest level Class is the Mote class, it describes the Mote object, Fundamentally it holds several arrays of samples, a timestamp array and a sample object array, the indexes of these arrays are synchronized. Moreover it holds unique metadata such as MAC address name of the logfile the mote corresponds to and more.
    - **Functions**:
        - LoadMote(): load a single mote from log file into ram, this function loads every piece of information about the mote inside the log file, all the sample values are converted from string to float and divided by 100 in order to move the decimal place to its original place before smart mesh converted them to integers for transmission, and puts them in an easy to call format:
            - Mote_1 = Mote(Directory, Logname) // Instance a mote object
            - Mote_1.LoadMote() // load all data
            - Mote_1.MAC // returns MAC address
            - Mote_1.status // returns status
            - Mote_1.timestamp // returns array of timestamps
            - Mote_1.temp // returns array for samples

- **SAMPLE:** The Lowest level class is samples, you don't need samples to access the sample arrays as they can be accessed from the mote class, however the sample object is an easy way to call all the data corresponding to a single timestamp and without calling multiple arrays.
    - **Functions**:
        - NONE

Application of API:

In the SensorSuiteGUI.py script you can see the API being applied to deliver the sensor data to the user in a meaningful way. However the GUI only scratches the surface of the potential, the GUI script uses the API as a simple way to call large amounts of relevant data in order to be used by the GUI Widgets. As a easy example the mote table is able to access the latest sample collected for each sensor by doing this: Mote_1.temp[-1], the -1 index calls the last element of the temperature sensor array, which is the latest value since the sample arrays are synched with the timestamp array and the time array is sorted by earliest to latest.

3.3 Energia Code SensorMote_V1.2 Guide

Overview:

There are three main parts to this code which are: lines 71-95 that are responsible for transmission of data via SmartMesh, lines 98-145 that are the setup and lines 149-261 that is the main running loop .The DFRobot O2 sensor takes lines 392-end and that is needed to run the o2 sensor, which was pulled from the original DFRobot o2 library. Most of the sensor libraries are pulled from their libraries in the credit section except for the Humidity/Temperature sensor.

In terms of sensor connections, there are 3 analog sensors(rain, co2, wind) that are connected to analog pins on the cc3200 and the rest of the sensors are all connected to I2C pins(SDA,SCL) with their corresponding addresses.

Low Power Mode:

One part of the low power mode is turning off/on sensors. This includes setting GPIO pins “LOW” and “HIGH” and in the setup the co2 and wind are automatically set to low to begin with, the rest are set high meaning they will be on. There is also sampling times for the sensors, meaning we will not want to sample a sensor if its still within a period range and the sensor will remain off.

```
230
231   //------------------------------------------------------------------ WIND SPEEED SAMPLE LOOP
232
233    bool CO2_TRIG = (holdValue * Lowpower_Period) >= 60*Co2_period;          // Check if if sampling perioud elapsed
234    if(CO2_TRIG){
235      Serial.println("Waiting for CO2..........");
236      digitalWrite(CO2_ACTIVATE_PIN, HIGH);              // TURN ON WIND Sensors
237      beginTime = millis();                              //  Setting up constaints for sample loop
238      }
239    else{Serial.print("CO2 Timer..... "); Serial.println(holdValue);}
240
241
242    SetupTime = 1000 * 15;           //  wait 30 sec before sampling Co2
243    while(CO2_TRIG){
244
245      if((millis() - beginTime) > SetupTime){
246        CO2_READ();                                 // CO2 SENSOR
247        CO2_TRIG = false  ;                          // Exit sample loop
248        holdValue = 0;
249      }
250    }
251    digitalWrite(CO2_ACTIVATE_PIN, LOW);   // TURN OFF WIND Sensors
252
```

Above image is an example of the sampling of the CO2 sensor. If the desired co2_period is greater than the low power period it will turn on the co2 sensor. Then we will give it 30 seconds before we conduct a reading and after that reading sample is completed the co2 will power off via GPIO pin.

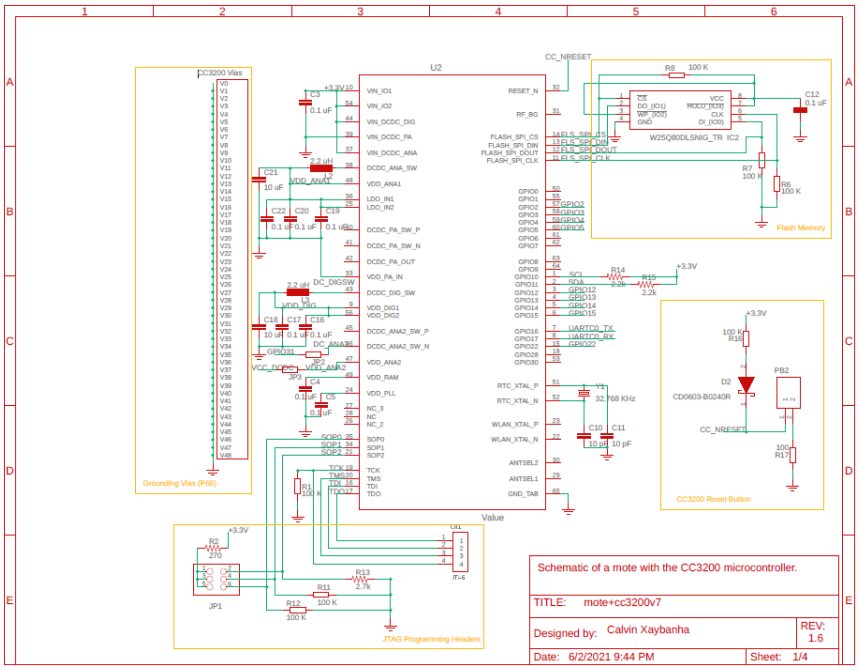
```
118
119   //------------------------------------- LOW POWER SLEEP SETUP
120   SleepDuration = 32768 * Lowpower_Period;         // 30 second sleep durtion, 10 minute sleep
121   pinMode(RED_LED, OUTPUT);
```

In line 120 we set how long we want to leave our mote in low power mode. In V1.2 it is set at 60 seconds, which is a global variable named “Lowpower_Period”. To verify it is in low power mode a red led will emit on the cc3200.

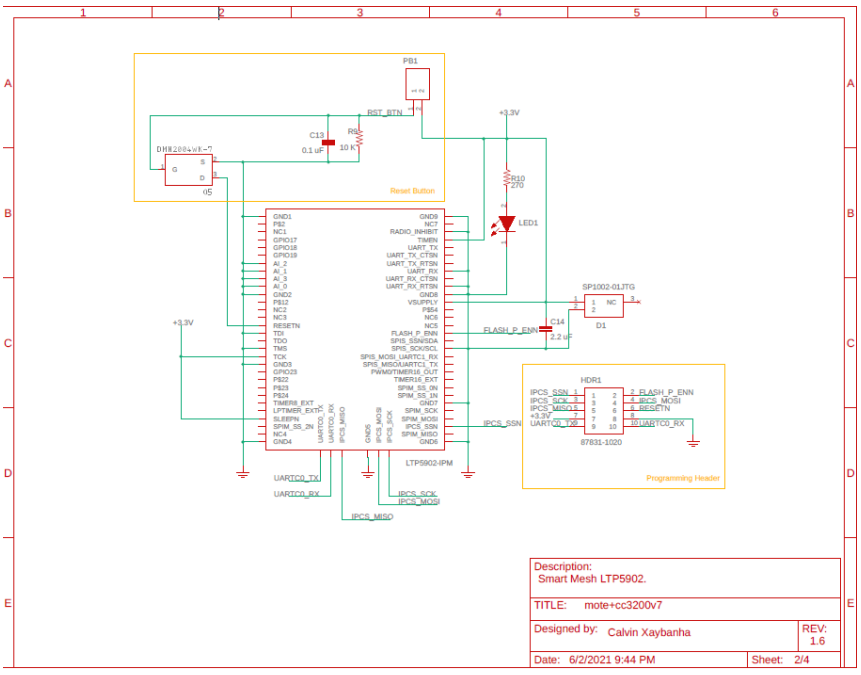
4. Instructions

4.1 Connection Diagram

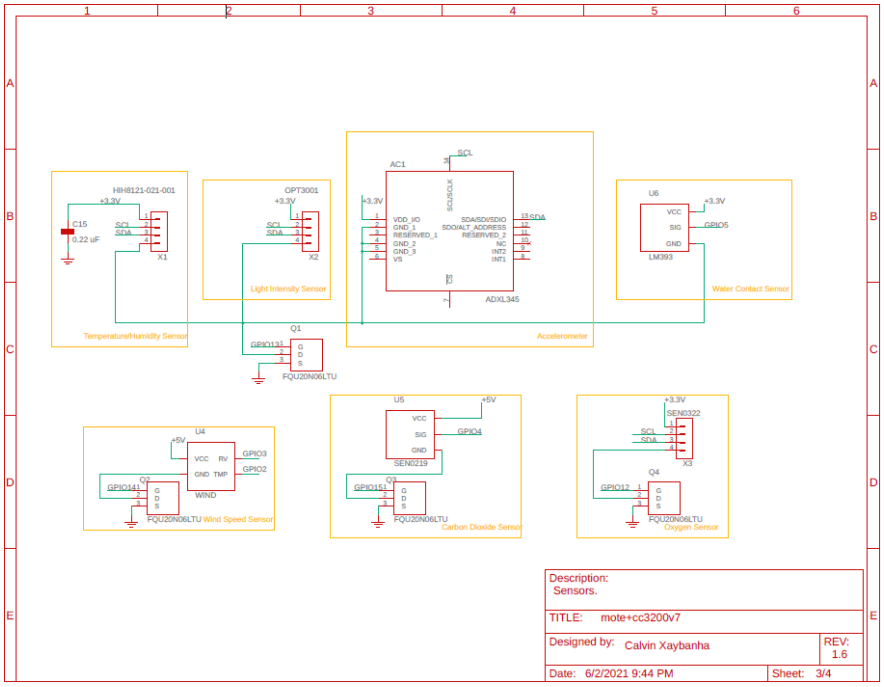
PCB schematic of connection



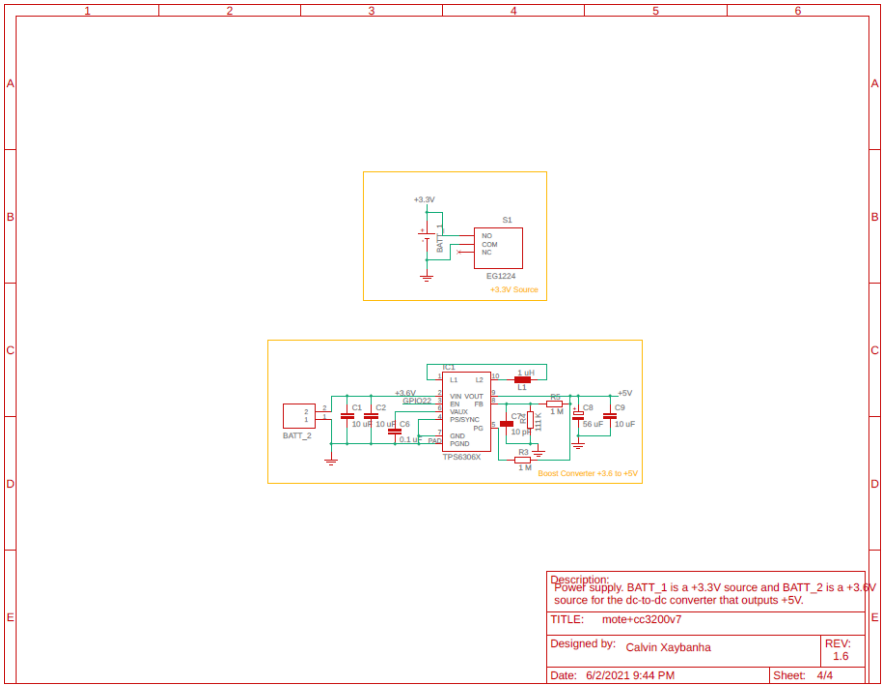
CC3200



Smart mesh module



Sensors

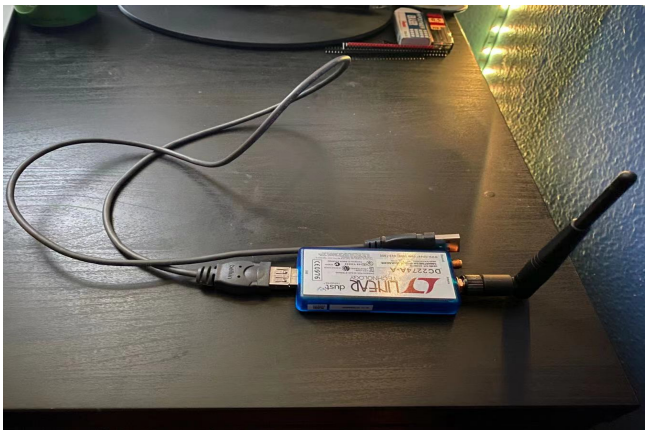


3.6V to 5V voltage Converter

4.2 Operation Process

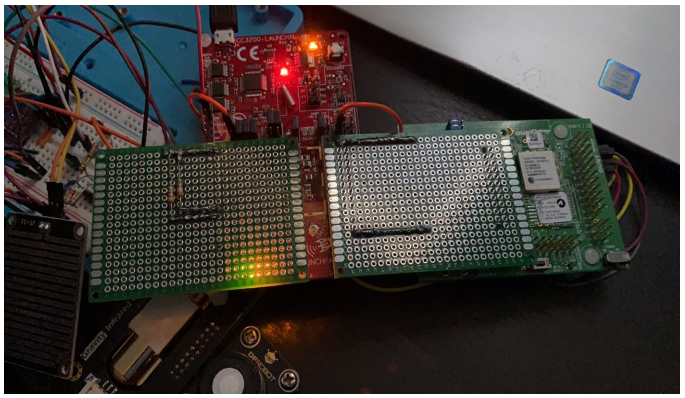
1.Connect devices(USB manager) with the computer.

- Install FTDI drivers & other recommended software, *no need to install node-red/node.js*
- Download Smart Mesh SDK, place in the same parent folder as SensorSuiteLogger
- OPTIONAL: go through Dust Academy lab-01 to lab-15

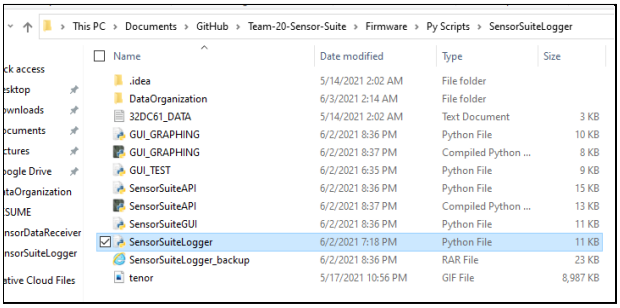


2. Turn on the motes

- There are alot of fly wires so make sure to handle the mote carefully as to not disconnect them.

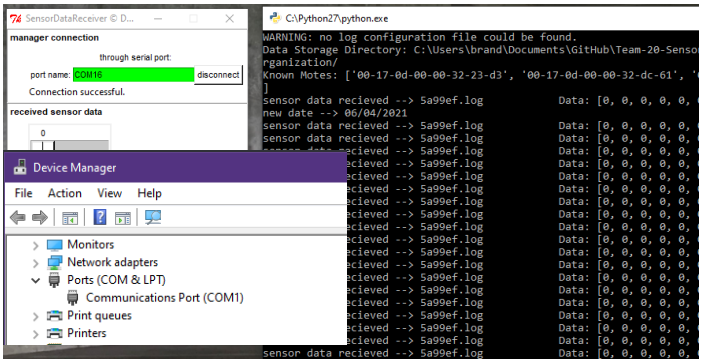


3. Open SensorSuiteLogger in computer: (path:\Team-20 Sensor-Suite\Firmware\Py Scripts\SensorSuiteLogger)

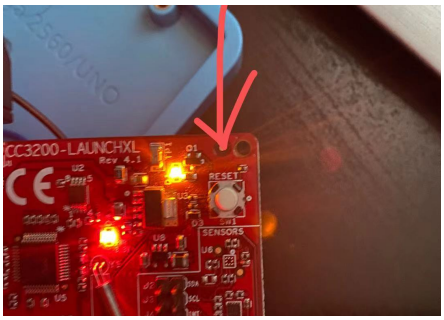


4. Enter COM# in port name

- Check the windows device manager to find port names
- You will likely find 4 port names try all of them
- Showing green on port name bar means connecting to the E-manager successfully

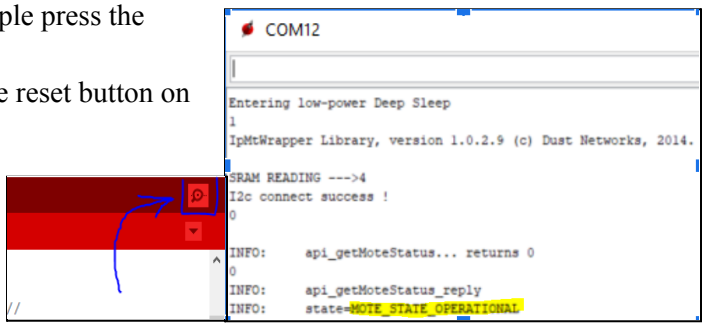


5. Push the button on motes to reset data (Activates the sensor communication)



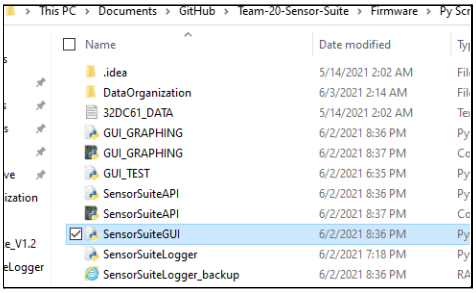
6. Open Energia serial monitor to see if Smart Mesh mote has connected to the Emanager (path:\Team-20-Sensor-Suite\Firmware\ENERGIA CODE\SensorMote_V1.2\SensorMote_V1.2.ino)

- If the sensor monitor displays “INFO: state=MOTE_STATE_OPERATIONAL” then it is connected
- If the device is taking too long to connect or sample press the reset button on the CC3200 launchpad
- If the mote is printing a “timeout!” error press the reset button on the smart mesh mote

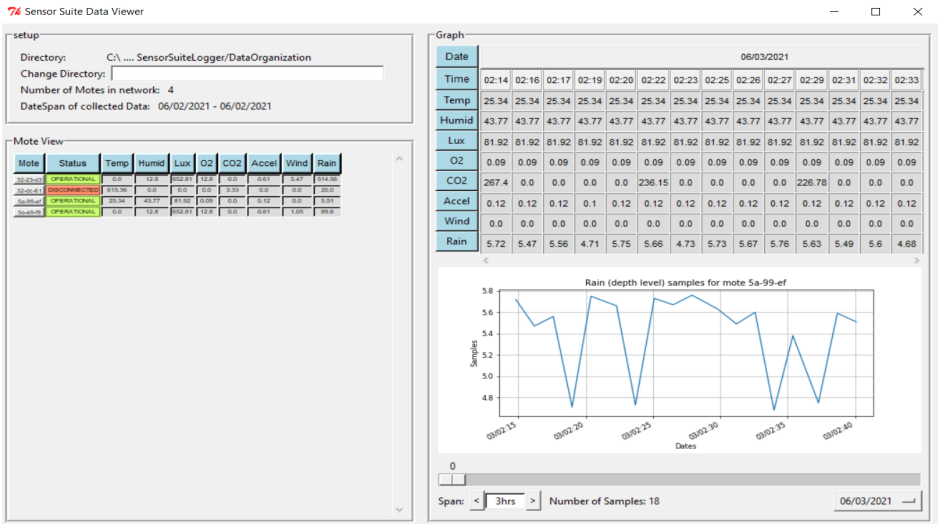


7. Check system is working properly:

- Check sensor suite logger console to see if data has been received.
- Check SensorSuiteLogger/DataOrganization Folder to see if more files are being Created/Updated by SensorSuitelogger.
- OPTIONAL: Check the Energia Serial monitor to see if mote is sending data



8. Open SensorSuiteGUI interface and view data (path:\Team-20 Sensor-Suite\Firmware\Py Scripts\SensorSuiteGUI)



5. Bugs and Issues

5.1 CC3200 Issues

- 5.1.1 Mote status misreporting:

  When first powering ON the mote after it being powered off for a long time, it will wait for the Smart mesh Module to connect to the mesh network, the CC3200 knows that the smart mesh has connected by checking the mote status variable that is reported by the SmartMesh C-library. This bug happens when the Mote status variable is misreporting the motes actual status, leading to CC3200 waiting forever.

  WORKAROUND: simply reset the CC3200 when its sending data without sampling, this happens when the sensors are connected but the Serial monitor prints 0 for all the sensors, and also the serial monitor prints the number “22” randomly (this is the mote status number, it's supposed to be 0 for IDLE and 5 for OPERATIONAL)

- 5.1.2 Sram Timer inconsistency:

  One of the issues we faced is the fact that the CC3200 LPDS mode leads to a complete reset of the program every time it exits the mode. This is a problem because we have motes that we want to sample at greater than sample/10 mins such as the CO2 sensor, so we needed a way to track time at longer intervals than 10 mins which is hard to do when the CC3200 enters low power mode every 10 mins and wipes all of its memory, so we used the Sram retention feature to set up a very basic timer, but this involves choosing a random memory location to write to, the problem is that a memory location that works for one CC3200 might not work for another.

  WORKAROUND: The easy working around is when programing a new CC3200 Choose a new Memory offset and check if its working, to check if its working see if the printed HOLD VALUE variable is increased by one after each 10 min sample period.

5.2 Api Issues:

- 5.2.1 Mote status does not update if it was modified whe GUI was closed:

  The api has a Mote Status updating function, this works by checking if the motes log file has not been modified for longer than 10 mins, if not then the API script opens the log file and rewrites its status as DISCONNECTED. The issue is that the date timezone between windows and the default python Datetime library is different, leading to inaccurate time comparisons. Moreover the entire mote status update function was rushed due to running out of technical work time so it needs to be reworked.

5.3 GUI issues

- 5.3.1 Needs more optimization:

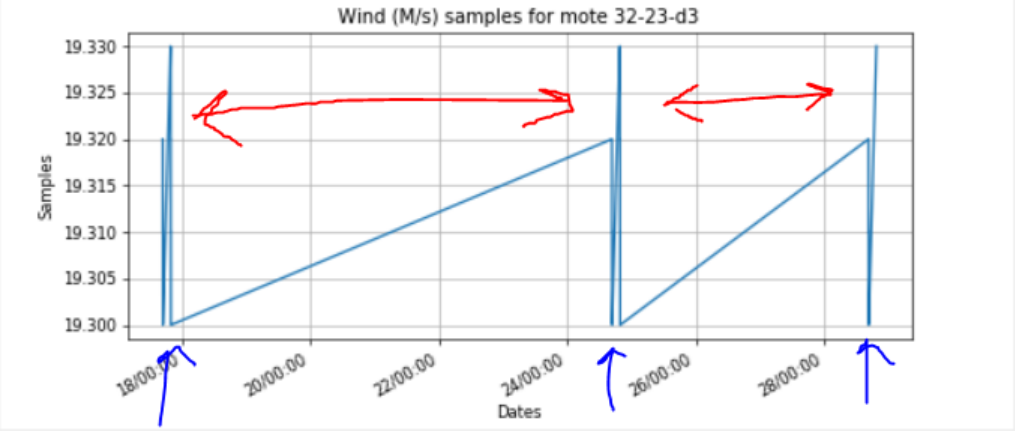
  The Tkinter GUI is quite slow at times and need to be optimized, someone needs to have a good understanding GUI runtime optimization and improve upon its performance

- 5.321 Graphing distortion due to missing Timestamps

  We are using Matplotlib to perform the graphing on the GUI, the issue is that we are graphene Date time objects as the X-axis of the plots, the side effect of this is that matplotlib is aware of any missing timestamps and samples, so if you have have any time gaps where the log file has no samples (for example if you only run the logger from 1pm to 3pm for a week) the plot will properly space each sample sample period meaning that your data will get squished and the majority of the plot space will be taken by the missing time stamp.

  As you can see from the screenshot below, the red parts are the date times where no sample were collected, and the blue are the periods are where the samples were collected, the plotter places all of them on the same time scale leading to the data occupying less space ten the empty space, this wouldn't be issued if you ran the motes and logger all the time since there wouldn't be gaps in time without and samples.

Wind	19.32	19.32	19.32	19.31	19.31	19.31	19.31	19.31	19.31	19.3	19.3	19.33	19.33	19
Rain	440.32	327.68	230.4	624.64	519.68	409.6	302.08	273.92	15.36	558.08	458.24	647.68	550.4	46



0

Span: < 12hrs > Number of Samples: 163 06/02/2021