ECSE 426 Microprocessor Systems

Final Project

Interaction of Embedded Peripherals on Smartphone

Group 08

Yan Qi	260457263
Sheng hao Liu	260585377
Chuan Qin	260562917
Wei Wang	260580783

April 23rd, 2016

Contents

1 Abstract

This report introduces our approach to establish an IofT Project: BLE connection and data exchange Between STM32F4Disocvery board and Android App via a Nucleo board with BLE shield. First of all, the Discovery board establishes connection with the Nucleo board wirily via SPI to send and receive data. Then, the Nucleo board (with BLE shield) communicates with the android App using Bluetooth Low Energy connection, which is a new technology that greatly reduces power consumption. The Discovery board is required to send its core temperature, Roll angle and Pitch angle to the Android app and display them on it, while the android app needs to control the Led lights on the Discovery board.

2 Problem Statement

This project aimed at building the communication between the Discovery board and an Android phone via Bluetooth Low Energy connection that is supported by STM32F401RE Nucleo board with an IDB04A1 shield. More specifically, the Discovery board is responsible for measuring its temperature and its orientation. Also, it requires a double tap feature using the accelerometer in order to wake up the smart phone. On the other hand, user can control LEDs on the Discovery board, including the intensity (PWM), the speed of rotation, and the direction of rotation through the application on the Android device.

Several operating functions appear to execute simultaneously on Discovery board. First, it digitizes temperature readings in analog voltage retrieved from the sensor at a particular frequency using the analog to digital converter (ADC). During the conversion, there are few forms of noises are introduced such as electromagnetic noise, thermal noise, and quantization noise. Second, the system calculates the roll and pitch angle based on its three dimensional measurements using the accelerometer. Last, the board detects whether there is a double tap action via an accelerometer algorithm then notifies the smart phone.

The Android device is used to display the temperature and the values of roll and pitch angles received from the operating processor which is the Discovery board. Additionally, the user has to be able to turn on and off all four LEDs and determine the order of how the lights rotate. However, the application cannot communicate directly with the Discovery board. The Nucleo board along with a shield provides a way for user to easily create and access the services and their characteristic attributes for the application. The data transmission between the Nucleo board and the smartphone occurs after the application searches for BLE device and set up the connection.

As the Nucleo board is not responsible for operating any measurements, a simple built-in communication protocol is implemented for two processors. While the data (e.g., the temperature and the angles) is ready, the Discovery board transfers data to the Nucleo board over serial peripheral interface (SPI) bus. Two boards can send and receive data in both direction using a master/slave architecture by specifying the four signals: clock, MISO (master data input, slave data output), MOSI (master data output, slave data input), and slave select.

Therefore, the system employs multiple functions such as temperature sensor, position sensor, communication between boards, and display. There were few challenges encountered while dealing with the specification listed above. Our challenges include: properly transmitting data between processors with different frequencies, defining whether the data transferred is meaningful or a dummy byte, designing the required services with their characteristics for BLE device, properly calibrating new Discovery board algorithms, and of course, communication with the Android app to exchange data.

3 Theory and Hypothesis

3.1 Theory and Hypothesis behind double tap

The accelerometer uses a system of weights and springs to determine acceleration by generating (usually) more voltage on a special crystal when the springs are more stressed. Generally, at rest, the accelerometer displays a total of 1g downwards in either X, Y, Z axis or a combination of the axis. If subjected to a sudden displacement (tapped), the stress on the springs would change since the weights would want to stay where they are by law of inertia. Thus, tapping on the board would make the accelerometer change its value since there would be less acceleration downwards because the springs are less stressed because the weights wants to stay where they are. The idea behind deciding if a motion was a tap on the board or just a displacement of the board is the difference in acceleration. If tapped, the difference in acceleration would be drastic and if moved (not jerked towards a position), the change in acceleration would be mild. We believe it would be hard to set the threshold where it we detect a tap since it would affect our ability to move the device around. Hypothetically speaking, we believe that if a change of more than 100 mg in any direction would be a tap. This value would need to be tested for the final product.

3.2 Theory and Hypothesis behind PWM

PWM or pulse width modulation is a technique in displaying brightness on a light source. When a light is on, the voltage is kept at a stable constant so the power is stable. When the voltage is decreased, the power is also decreased and the light grows dimmer. However, in our case, our only options are either on or off. So we need to space out the total energy used for our LED. We need to keep in mind that if the toggling of on and off is too slow, then the eye will see on and off instead of a decrease in brightness. For example, if for 1 second, we turn it on for half a second and then turn it off for the other half, the total energy used is for half a second. However, it does not appear dimmer only that it is off for a half second. This is why a small period for a cycle is needed. Say for example that our whole cycle is 2ms, then for 1ms we turn the LED on, and 1ms we turn the

LED off. So that means, for 1 second, there will be 500 toggles. The energy used is still for half a second (ignoring toggle energy) but the eye cannot distinguish between the toggles, and the LED will appear dimmer. The idea for our implementation is to have a “duty cycle” of light activation where it is a fraction of the whole cycle. In other words, we want to keep the light on for a certain percentage of time so we can control brightness. We believe that implementing a total brightness value of 100 will keep things simple. So if our duty cycle is 10, then the light will be on 10% of the time.

4 Implementation

The implementation for this project was done separately in parallel at first where each member of the team had a specific part to complete. The work breakdown will be discussed more in the conclusion. Then towards the end of the project, we have combined all of our code into a readily usable android app which can explore all relevant embedded peripherals.

4.1 Discovery Board

The discovery board’s features for this project were built on top of lab 3 and lab 4. The temperature (ADC), accelerometer, LEDs, and threading functions for the final project are added on top of what we have built before. The main changes were to variables to be used in a master slave context and the addition of algorithms. Some integer variables have changed into unsigned 16 bit integers to be able to send towards the Nucleo board, then to the smartphone.

4.1.1 Temperature Sensor

The discovery board’s temperature mode was unchanged from lab 3 and 4. The conversion algorithm was unchanged from the previous labs. Values for filtering, threading, and parameters also did not change. The only small change is the creation of a new variable “tempSent” as an unsigned integer to be sent towards the Nucleo board. The new variable is the filtered temperature times 100 for two decimals places since we are passing an unsigned integer. This value will be divided by 100 once it has been successfully passed towards the smartphone.

4.1.2 Accelerometer

The board uses the calibrated accelerometer conversion algorithm from lab 3 and 4 to determine its difference in acceleration value. A new algorithm was made in order to determine changes in acceleration. First set a dummy previous value of the acceleration as 0, 0, and 1000 mg in X, Y, and Z axis values. This means when first starting the application, the board must be flat on the table to work properly. Then subtract the new value of the accelerometer by the previous value to get the difference in acceleration. The algorithm continues the process of replacing the previous values by the new accelerometer values to have a continuous tapping detection capability. If the absolute value of the difference in acceleration is more than a certain threshold (100 mg when first implemented), then it would record that as a tap. We have chosen to use the absolute value since it was easier to debug and to understand. We had to include a higher delay value as a result of using an absolute value. Whenever two taps are detected within a certain time frame (one second when first implemented) the double tap counter would go up and that flag can be used elsewhere. The double tap algorithm was set to be much more sensitive by choice (more discussed in testing); this ensures we can detect taps for use. We could have put the algorithm on java (android side), but we need to be careful of the nyquist rate and the refresh rate for the phone side for it to work properly. We considered that a hassle, so we implemented the double tap algorithm on the discovery board for simplicity and seamless data fetching. Finally, we pass along the tilt values as an unsigned integer and the double tap value towards the SPI module.

4.1.3 LED

The LEDs uses GPIO on pins 12 to 15 with HAL Clock D. We set the mode to be OUTPUT_PP (push/pull), NOPULL to disable pull resistor, and speed as SPEED_MEDIUM since we did not believe it needed to be high for our use. These values were used and tested in the previous labs. For our implementation, these settings worked fine.

4.1.3.1 LED Spin logic

The LED algorithm for spin speed is simple, for a certain amount of time, let one of the LEDs be on and all others off, then shift to the next one either clockwise or counterclockwise. We chose for simplicity to have the input speed as the delay. Thus, for a bigger amount of delay (input), the LEDs will spin slower. If the input is zero, then there is logically no delay and all LEDs will be on. To have it spin counterclockwise, have the input to be negative and the algorithm itself will deal with which LED has to be on.

4.1.3.2 LED PWM logic

In our implementation of the PWM algorithm, we made it so that it takes in an integer number which will represent the duty cycles. The total time the algorithm will run for is 100 “cycles” for simplicity. For those 100 cycles, the duty cycle input is the fraction of time it is on. So for example, an input of 20 will mean that the LEDs will be on 20% of the time. This will reduce the overall brightness, thus successfully implementing the PWM.

4.1.3.3 LED combined logic

For the implementation towards the final project, we wanted to combine both algorithms to control speed and brightness at the same time. To let the four LEDs spin in succession, we turn on one LED for a specific time interval, and after that the LED will be turned off and the next one will be turned on. The speed of spinning determines the duration of the turning on interval. The time interval is a multiple of the PWM cycle which can ensure that the function of spinning and brightness control can both work properly. When the value of rotating speed is 0, we won’t make the four LEDs rotate but we turn on all of them. To implement these functions, we create a method called void Scenario (int ledflag). The value of ledflag determines which LED will be turned on(explained in the chart below.)

Table 4.1: LED flag operation principle

ledflag	Orange	Blue	Green	Yellow
0	On	Off	Off	Off
1	Off	On	Off	Off
2	Off	Off	On	Off
3	Off	Off	Off	On
6	On	On	On	On

When ledflag is from 0 to 3, only one LED is turned on and when it becomes 6 all of them will be turned on which will happen when spinspeed becomes 0.

Table 4.2: “Spinspeed” representation

Spinspeed	Time interval(ms)
1	500
2	450
3	400
4	350
5	300
6	250
7	200
8	150
9	100
10	50

If the value of the spin speed is positive, the four LEDs will spin clockwise, otherwise, the LEDs will spin anticlockwise when it’s negative .

LED PWM Brightness:
The period of TIM3 is 0.01ms and a single cycle of PWM is 1 ms. The TIM3 was changed from the previous lab in order to keep the 100 cycles logic while maintaining the stable visual updating effect of 1kHz.

4.1.4 Real-time Operating System Threads

The 7 segments display thread from lab 4 was not used by choice. The thread was made in the context of lab 4 where the display is on a 7 segments display with a keypad to interact with different features. However, in the final project, we have no need for such a display since we have the Android app to display all relevant information and to control the LEDs on the discovery board.

4.1.4.1 Temperature thread

The temperature (ADC) thread was not changed from lab 3 and 4. We have kept the priority to above normal as it is a sensor. The stack size is set to 400 to save some space. The thread itself only initializes a kalman filter struct then proceeds to update temperature values at 100 Hz. The updated temperature value gets sent to the Nucleo board.

4.1.4.2 Accelerometer thread

The accelerometer thread remains largely unchanged except for the fact that we have added the tapping detection function. The thread is set to above normal because it is a sensor and the stack size is set to 800. This thread initializes three kalman filters for each axis and initializes the “previous” variable for the tapping algorithm. It follows a 25 Hz signal initialized from the accelerometer parameter Power_Mode_Output_DataRate and the data is taken from the interrupt driven on EXTI0 which is pin PE0. Then it updates tilt values and the double tap flag at the 25 Hz frequency until the application stops.

4.1.4.3 LED thread

The LED thread uses the modified logic to combine the PWM and the LED rotation. It takes in the spin speed and brightness values as input and gives flags for which state the LED should be in. Each LED will be on for a certain amount of time determined by the speed input. To control the PWM, we have set the brightness control to be in the TIM file. To successfully implement the 100 “cycles” logic, we have accelerated our hardware clock by 100 (from 1kHz). This will ensure that both PWM and LED rotation will work in parallel as they are embedded in each other. For example, if we need a brightness of 20%, the algorithm will activate the LED in a specific direction 20% of the total duration it is supposed to be on. The spin speed will determine the actual duration. This thread closely links to the TIM file with a flag and only has visual outputs. The thread is set to normal priority and default stack size (800).

4.1.4.4 SPI thread

The SPI thread is used for handling a master slave communication between the Discovery board and the Nucleo board. We have set the thread to be at normal priority and made it follow the default stack size (800) for simplicity. It follows a 40 Hz hardware signal from TIM for clock speed. The thread itself has various hardware signals to be sent (like the temperature) and received (like the brightness) from the Nucleo board. The SPI functionality will be discussed in more detail below.

4.2 Nucleo Board

The Nucleo board and Nucleo-IDB04A1 shield is based on the BlueNRG network process, and then it creates Bluetooth Low Energy, which is a technology used to build a custom profile for specific application on Android phone. BlueNRG uses an Application Controller Interface (ACI) that is built upon the SPI interface to interact with user's application.

The phone plays the role of master that scans for the Bluetooth signal. On the other hand, Nucleo board is slave role waiting for the connection and accepting any incoming request. In a BLE network, the board can only connected to a single master. More specifically, the server device, computer, provides the data resources using the attribute protocol to the remote client, Android phone. All data transactions are started by the phone's request to Nucleo board, and phone receives the response from it. The GATT client can read or write attributes found in the server according to their access permissions controlled by the GATT server. The server is responsible for storing and organizing data.

In general, the profile application is defined through a GATT structure, and GATT functionality establishes the information exchanges over BLE. Each profile is a collection of services, and each services is made up of characteristics. Both services and characteristics are represented as attributes. Each service are declared by an attribute to mark the beginning of a service so that it must be the first attribute of that service before adding any characteristic. Every characteristics has a single main attribute that have its own database to store the actual value. Each attribute is defined by a 128-bit universally unique identifier called UUID with specific access permissions. Every attribute handle on a server represents one attribute for client's reference. Permissions can only apply to the attribute value to determine whether the attribute is readable or writeable. Also, attributes need to set up their authorization to determine whether the client requires access permission. There are three different primary services such as double tap service, LED service, environment sensor service.

4.2.1 Double Tap Service

The double tap service consist of a single unsigned 1-byte characteristic. When a user double taps the Discovery board, there is a signal sent from the Discovery board to the Nucleo board through

SPI communication. This service is responsible for waking up the screen of any Android phone. This double tap characteristic has a single operation which is notify because the server is able to send data to the client without any request. Then, there is no authorization required for that attribute. Once the Nucleo board receives a double tap signal, it sends a notification to the phone by writing and updating a new value on that characteristic.

4.2.2 Environmental Sensor Service

The environmental sensor service is the service that contains the three unsigned 16-bit characteristics followed by presentation format descriptors. The descriptors are read only without any authorization. Each descriptor includes five parts as listed: format, exponent, unit, name space, and description of that attribute. The three characteristic values represents temperature, pitch angle, and roll angle of the Discovery board. These attributes are defined to be readable and do not require authorization by the client. As read operation in BLE, the client must send read requests, and the server then responses with the updated data which is transmitted from Discovery board. In fact, the discovery board periodically measures the temperature, calculates the angles using accelerometer, and passes the results to Nucleo board. However, the characteristic attributes are being read by the phone at different rate. The attributes are only updated when the client sends the requests. Once there is a request, the flag EVT_BLUE_GATT_READ_PERMIT_REQ of HCI function is trigger then it activates Read_Request_CB( ) method.

4.2.3 LED Service

On the server, the LED service contains a unsigned 2-byte writeable (both write and write without response) characteristic with write authorization allowing client to modify. This write operation is requested by the client so that user can send data to the server. User can control the intensity of lights and how fast he/she wants the 4 lights to rotate either clockwise or counter-clockwise. All the LED data is written and transmitted then the GATT server stores the information. When user writes a value, BLE detects an ACI event and invokes Host Controller Interface (HCI) event. In HCI event function, there is a flag called EVT_BLUE_GATT_ATTRIBUTE_MODIFIED which is triggered when a GATT attribute is modified by the client. Then, Attribute_Modified_CB( ) method extracts the received value from event packet in order to pass the information to Discovery board.

4.3 SPI

4.3.1 Master

In our project the Nucleo board is the master and uses SPI2 to build communication with its slave - discovery board. Nucleo will send the data of LED attributes to discovery and discovery will send the data of roll/pitch angles, temperature and double tap flag to Nucleo. As shown in the figure below, PINB12, PINB13, PINB14, and PINB15 are set as slave select, SPI clock, master output slave input(MOSI) and master input slave output(MISO) respectively. Furthermore, we set PINB2 as the slave data ready pin. When slave has some data to transmit, this pin will be set and then the Nucleo board can know and ready to acquire the data from discovery board.

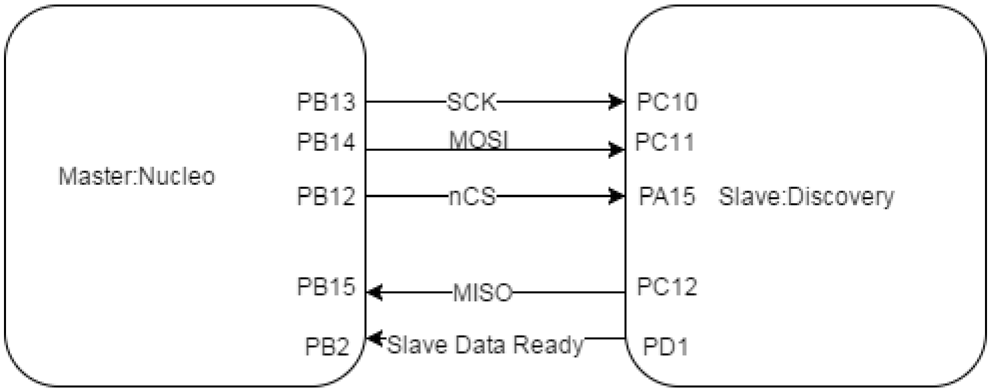


Figure 4.3: SPI connection between Master and Slave

The flowchart on the next page shows a whole process of Nucleo’s side in SPI communication.

At the beginning, Nucleo checks whether the slave has data to transmit (if so, slave will set the PIND1). If the PINB2 is set, it will select the slave (reset the PINB12). And it would send the data to discovery when the transmit data buffer becomes empty. Then when the receive data buffer becomes filled it will read the data come from discovery. Before it deselects the slave (set the PINB12), it needs to wait SPI to become free and PINB2 becomes reset. Finally it will return the data received.

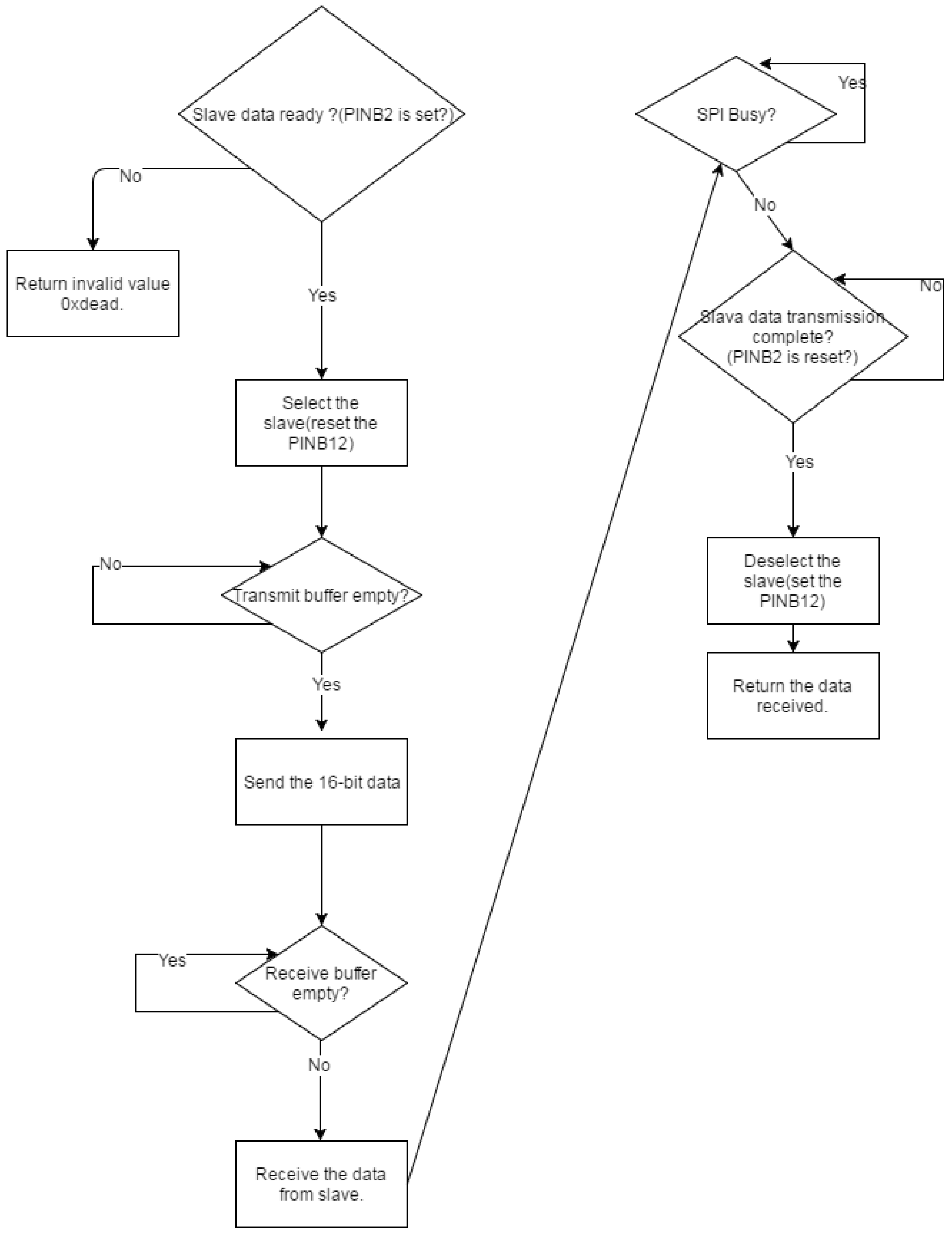

Slave data ready ?(PINB2 is set?)
No
Return invalid value 0xdead.
Yes
Select the slave(reset the PINB12)
No
Transmit buffer empty?
Yes
Send the 16-bit data
Yes
Receive buffer empty?
No
Receive the data from slave.
SPI Busy?
Yes
No
Slava data transmission complete? (PINB2 is reset?)
No
Yes
Deselect the slave(set the PINB12)
Return the data received.

Figure 4.4: Flow chart of SPI connection (Master)

The data from Nucleo to discovery and from discovery to Nucleo are all 16 bits. A 16-bit number can represent 2^{16} which is 65536 different values. And the range of LED brightness is from 0 to 100 and the range of LED spinning speed is from -10 to 10. In total, there are 101×21 combinations that is smaller than 65536, which means one 16-bit number is enough to represent the two LED attributes altogether. In detail we define the 16-bit number sent to discovery by the following formula:

$$Number = Brightness * 256 + (Spinningspeed + 10).$$

The data sent from discovery has two types: one is the data of temperature, angles and double tap flag. The other is the index which represents what will be the next received data. The detail is seen in the following table.

Table 4.5: Index and received data

Index	Next received data
0x0000	temperature
0x0001	Pitch angle
0x0002	Roll angle
0x0003	Doubletag flag

For instance, if Nucleo gets a number of 0x0000 from discovery, then next time the data sent from discovery will be temperature and Nucleo will update the temperature's value which will be sent to Android mobile phone.

4.3.2 Slave

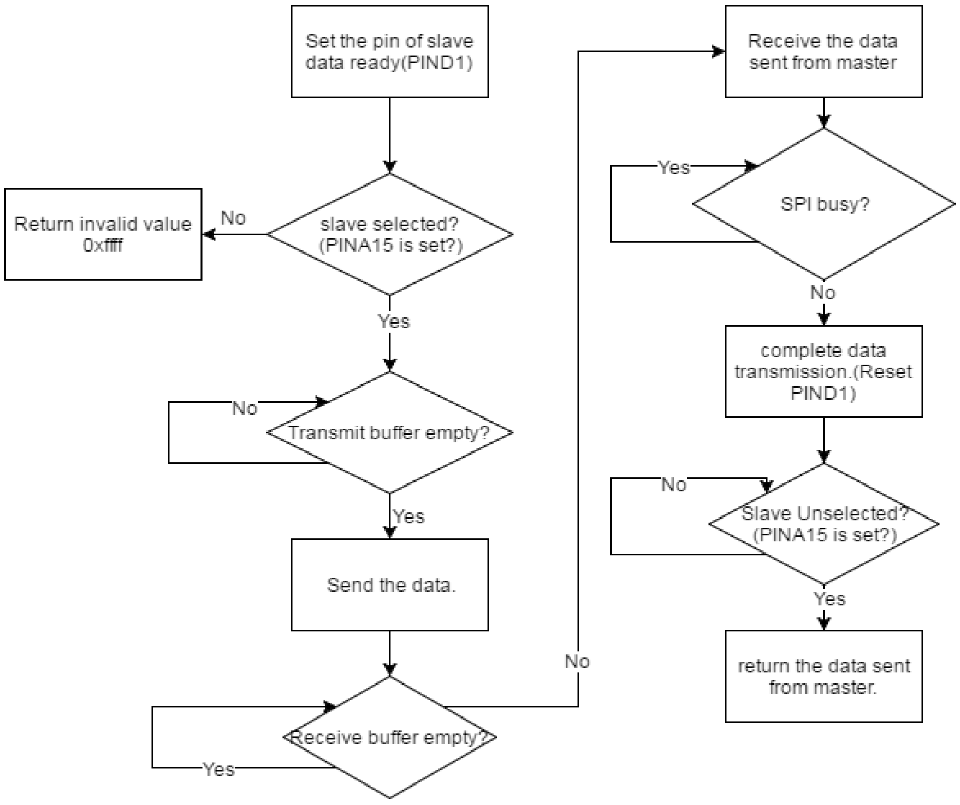

Set the pin of slave data ready(PIND1)
slave selected? (PINA15 is set?)
No
Return invalid value 0xffff
Yes
Transmit buffer empty?
No
Yes
Send the data.
Receive buffer empty?
Yes
No
Receive the data sent from master
SPI busy?
Yes
No
complete data transmission.(Reset PIND1)
Slave Unselected? (PINA15 is set?)
No
Yes
return the data sent from master.

Figure 4.6: Flow chart of SPI connection (Slave)

The discovery board in our project is the slave board and it uses SPI3 to communicate with its master - Nucleo. The flowchart above shows the process of Discovery's side in SPI communication. At the beginning, to let the master know it has data to transmit, it will set the pin of slave data ready (PIND1). Then it will wait to be selected by master. After transmitting, the data buffer becomes empty it will sent the data to master. After receiving operation is complete, the data buffer gets filled, it will receive the data from master. After the SPI becomes free again it will reset the PIND1 to notice master that it has finished data transmission this time. Finally it will wait master to deselect itself and return the data received from master.

The data we send to Nucleo is accurate to percentile. The range of both pitch angle and roll angle are from 0.00 to 180.00 degree (18001 kinds of values). And the range of temperature is from about 30.00 Celsius degree to no more than 100.00 Celsius degree (7001 kinds of values). Since a 16-bit number can represent 65536 different values so one 16-bit number is enough to express the data of temperature/angle. In detail we define the data sent to Nucleo by the following formula:

$$TemperatureData = Int(temperature * 100)$$
$$AngleData = Int(angle * 100)$$

As for doubletag flag, if a doubletag is detected, discovery will send 0xbbbb to Nucleo; otherwise it will send 0xaaaa.
Before sending data to Nucleo, discovery will send a number to indicate what the next data will represent. The detail information is shown in the table above (table 4.5). For instance, if discovery wants to send the measured temperature data to Nucleo, it will first send a "0x0000" to Nucleo and then send the measured data.
Once one operation of sending is completed, the discovery will get the data of LED. And the brightness and spinning speed of LED will then be updated.

4.4 Android App

The App design is based on the "BluetoothLeGatt" sample project provided in the Google Sample project base. In order to be able to make use of the board via phone connection with the BLE function, we need to modify the existing app's second activity, i.e.: "Device Control Activity" to make sure it functions correctly.
To start, the android app needs to satisfy the following functions:

- Display the Roll, Pitch, and Temperature value measured from the STM32F4-Discovery board
- Control the LED pattern on the STM32F4-Discovery board, namely:
    - Intensity of the lights
    - Rotation speed and direction of the lights.

All of the above functions need to be achieved in accordance with the protocol with the Nucleo board with BLE shield.

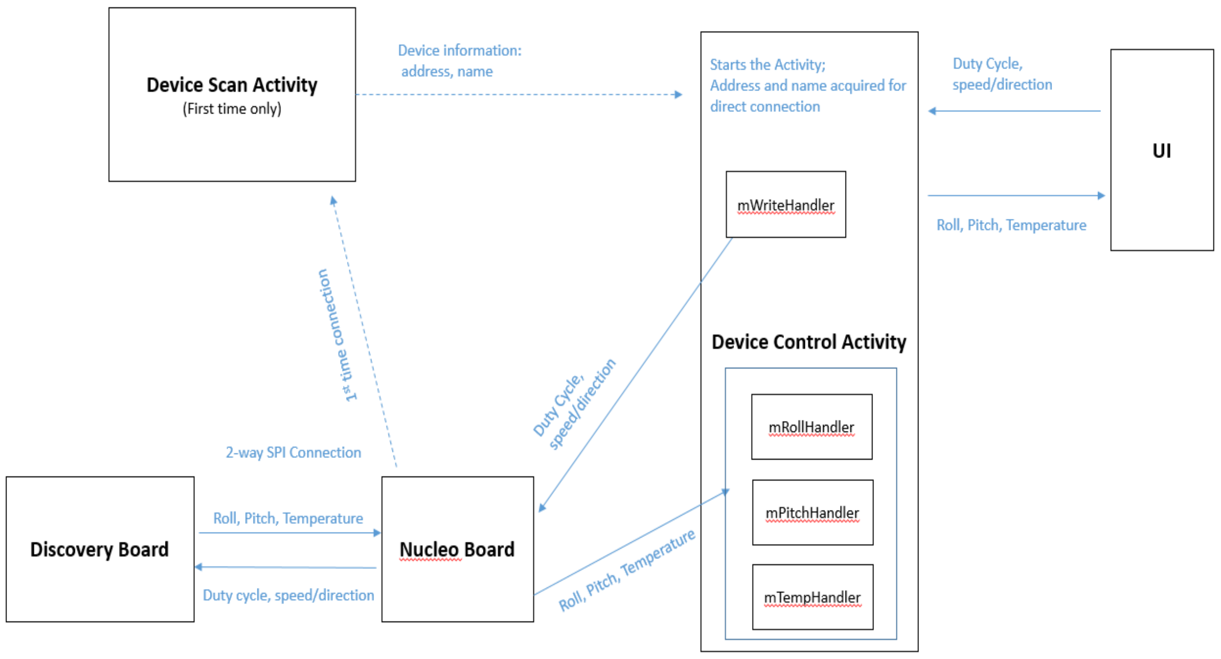


Figure 4.7: Interaction between the App and the Discovery board

4.4.1 UI Design

For simplicity of the Layout, we've decided to use the Relative Layout format to set all the data and functions, with MVC concept.

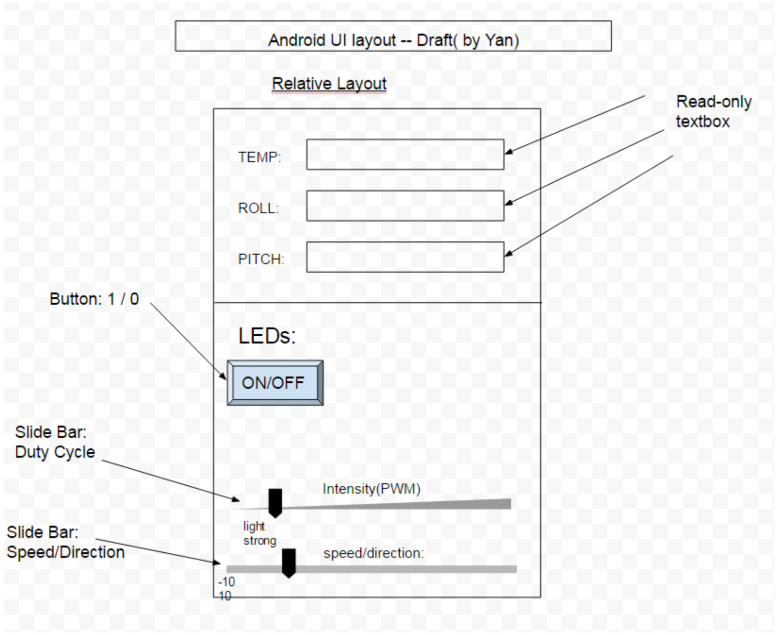


Figure 4.8 Android App UI Layout Draft

- **Data Display Section:** This part includes the 3 textboxes shown in the top part of the Layout, and will update themselves based on the value received. To achieve the required functionalities, we've decided to use TextView object(Textbox) to display value(temperature, roll and pitch) acquired from the Discovery board.

  - As we only need 2 decimal points to be displayed on the in the TextBox, the value that received from the board would be the (real value * 100) to simplify the data-transferring process with less bits. After that, the data will be displayed via dividing by 100 and shown on the Textbox. In this way, the data is dealt within the android app, which makes the data transfer easier and more reliable.
  - The 3 textboxes will update separately, they are not connected in anyway.

- **LED Controller Section:** 3 functions are required. For this part, we are required to send 3 data set to the Discovery Board to control the led lights via the Nucleo board.
  - *On/Off Button-Button Object:* send either a 0 or 1 to the Discovery board.
  - *Intensity(PWM) - Slidebar object:* send the value 0 to 100 to the Discovery board. This data serves as a multiplier for the duty cycle to control the brightness for of the led lights on the board.
  - *Speed/Direction - SlideBar object:* send a value from 0 - 20 to the Discovery Board (however on the app it will be displayed from -10 to 10), whereas -10 to 0 is making the led lights rotate in counterclockwise direction, whereas the positive values in clockwise direction. The bigger the absolute value of the data is, the faster it rotates(ex: -10 is faster than -5, -5 and 5 have the same speed of rotation)

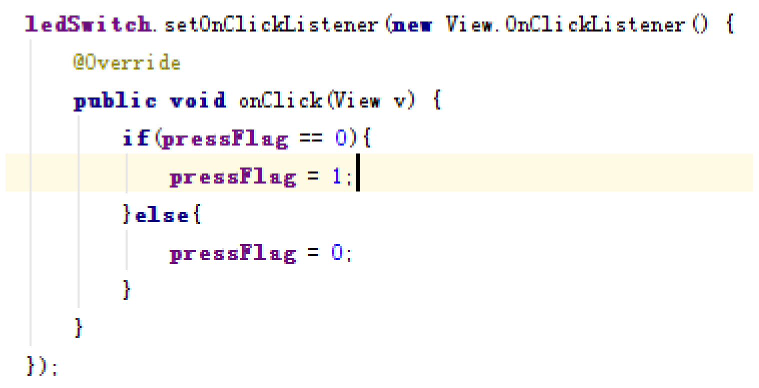
4.4.2 Life Cycle

In order for the apps to function correctly, we need to implement different processes and methods in its life cycles. For example, what variables should be initialized on startup, what should be working and what shouldn't be when the user no longer interacts with the UI, etc.

4.4.3 Organizing System Features

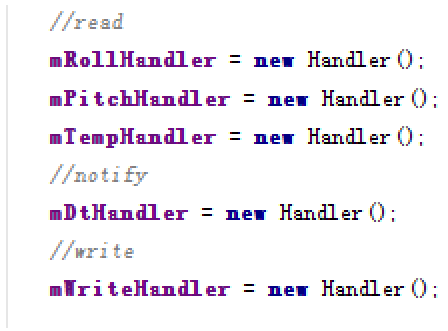
We've made full modification to the 'DeviceControlActivity" class, mainly:

- **onCreate():** The first interaction that the User have when interacting with the UI. This service starts when user clicks on the app for the first time.
  - *UI initialization:* This part associates all the corresponding components to its UI objects, so that the data can be modified in the background and updated on the UI, namely:
    - Speed/Rotation: speedBar(Seekbar), speedBarValue(TextView)
    - Intensity: intensityBar(Seekbar), intensityBarValue(TextView)
    - Roll/Pitch/Temperature data: mRollValue(TextView), mPitchValue(TextView), mTempValue(TextView)
    - Led Button Switch: ledSwitch(Button), as well as the button's click setup:



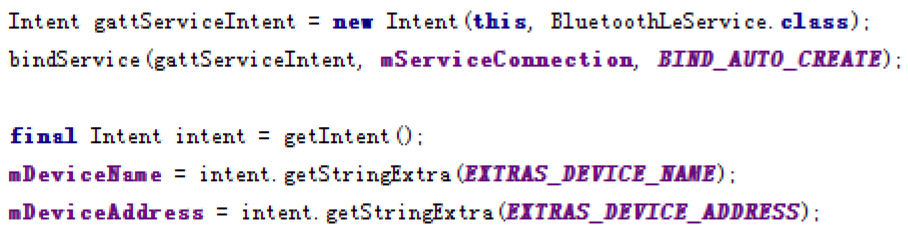
```
ledSwitch.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(pressFlag == 0){
            pressFlag = 1;
        }else{
            pressFlag = 0;
        }
    }
});
```

- The button initially has a value of 0 when passing to the Nucleo board; when pressed, the value would toggle.
    - This works directly with the Intensity method: the intensity that the led light will be is the "IntensityValue x ButtonValue", which guarantees the on/off operation.
- *mHandler initialization:* for this part, the following Handlers are initialized:



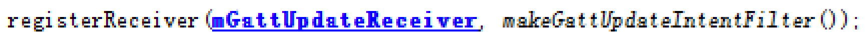
```
//read
mRollHandler = new Handler();
mPitchHandler = new Handler();
mTempHandler = new Handler();
//notify
mDtHandler = new Handler();
//write
mWriteHandler = new Handler();
```

- "mDtHandler" handles the Double-tap notifications.
- "mWriteHandler" handlers the data that needs to be sent to the Discovery board to process.
- *Intent object initialization:* Intent serves as a intermediate to exchange the information between the UI and the background operations, which closely follows the MVC concept.



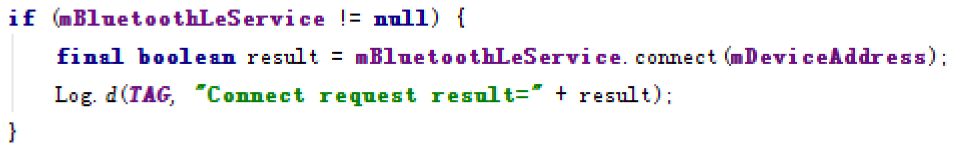
```
Intent gattServiceIntent = new Intent(this, BluetoothLeService.class);
bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);

final Intent intent = getIntent();
mDeviceName = intent.getStringExtra(EXTRAS_DEVICE_NAME);
mDeviceAddress = intent.getStringExtra(EXTRAS_DEVICE_ADDRESS);
```

- **OnResume():** This service is activated when user interacts with the UI, or UI is available to users.
    - *Data updates:* register broadcastreceiver:



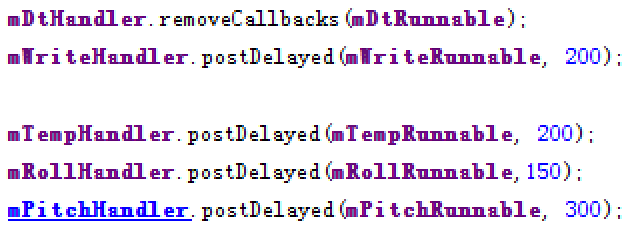
```
registerReceiver(mGattUpdateReceiver, makeGattUpdateIntentFilter());
```

- *Connect to the designated Service*(Figure 4.6):



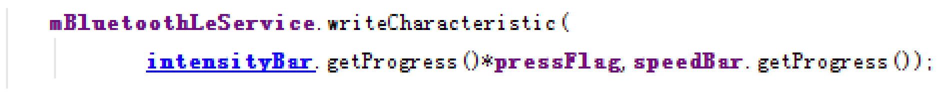
```
if (mBluetoothLeService != null) {
    final boolean result = mBluetoothLeService.connect(mDeviceAddress);
    Log.d(TAG, "Connect request result=" + result);
}
```

- *mHandler execution:* each Handler is activated with a delay to save battery and get more reliable data.



```
mDtHandler.removeCallbacks(mDtRunnable);
mWriteHandler.postDelayed(mWriteRunnable, 200);

mTempHandler.postDelayed(mTempRunnable, 200);
mRollHandler.postDelayed(mRollRunnable,150);
mPitchHandler.postDelayed(mPitchRunnable, 300);
```

  - *mDtHandler:* as this is the OnResume() part of code which means the UI has already been opened by the user, we don't need to activate the double-tap detection feature.
  - *mWriteHandler*: This part simply passes the value of intensity Bar, button Value, as well as the speed Bar value to the Nucleo:



```
mBluetoothLeService.writeCharacteristic(
        intensityBar.getProgress()*pressFlag, speedBar.getProgress());
```

  - *mTempHandler, mRollHandler, mPitchHandler:* These 3 simply reads the data on the Gatt server of the Nucleo board.

- **OnStop():** when the user no longer interacts with the UI or the UI can't be seen.
  - Handler processes update: when the UI is no longer to be seen, there is no point to keep reading the values from the Gatt Server, so the temperature, roll and pitch values don't need to be kept updating. On the Contrary, the Double-tap feature needs to be activated in this case to light-up the screen and give a notification.

Using MVC concept, the processes are handled in the Handler methods for simplicity as well as controllability. The data that is being exchanged with UI and background are transferred via the Intent methods, which passes the background data to the UI or vice-versa.

4.4.4: Background: BLE Connection and Read,Write,Notify

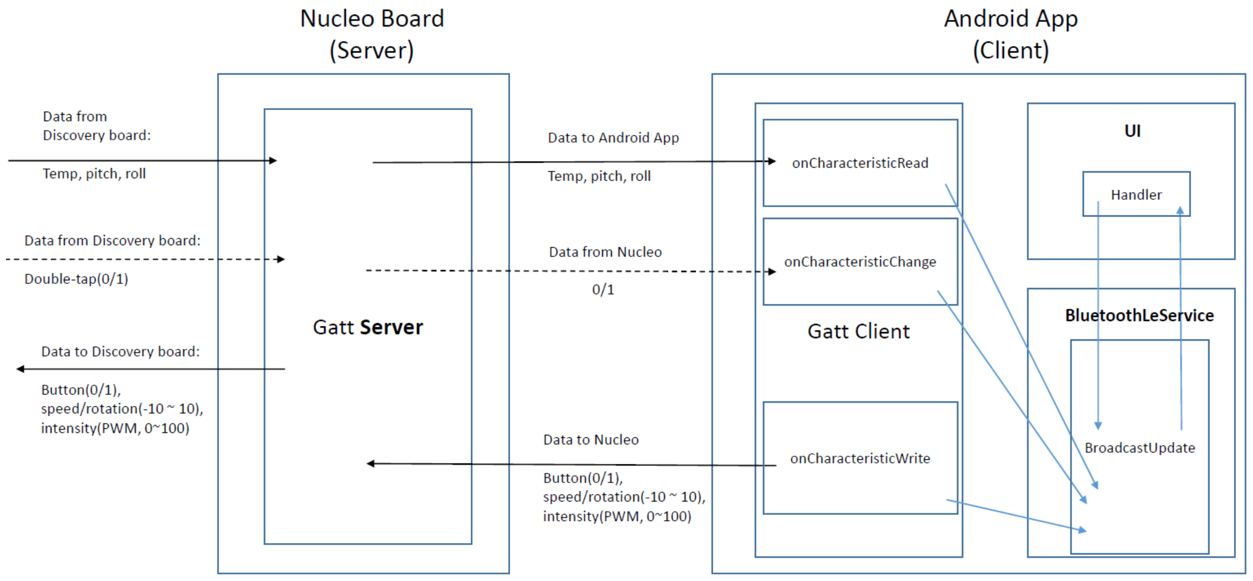


Figure 4.9: Data Exchange between Nucleo Board and Android App

For the background operations, which includes BLE connection and data updates, a separate class of “BluetoothLeService” is provided to handle the background information. The following are the main function of this class:

For the setting up initial connection, we are using the sample app’s “DeviceScanActivity” as the main approach. It scans for all the available devices and let you choose the one you wish to pair with. After you’ve chosen the one you wish to do so, it will save the device’s information and address in the background and triggers the “DeviceControlActivity” which starts the main data exchange operation.

BLE only starts to consume battery when read, write or notify request has been received, which greatly reduces the battery consumption, thus, we require a callback method each time such a request has been established.

- readCharacterstic: This function basically reads the value of the characterstic via the characterstic’s UUID, and pass the value read via “onCharacteristicRead” method in the BluetoothGattCallBack; the following 3 characteristics will be read: Temperature, Roll, Pitch; with each of them having a different UUID.
- BluetoothGattCallBack: This is the main method that is being used to handle the request after the connection a connection has been established, this method contains the following functions that is necessary for the read, write and notify operations:

    - onCharactersticRead(Read): This method is called when a read operation has been performed and the relative data has been received. The data would be passed on the to “broadcastUpdate” method to update the corresponding value on the UI
    - onCharacteristicWrite(Write): this method is called when a write operation has been performed and the relative data is ready to be sent to the Gatt Server(Nucleo), as we do not need any response after we’ve sent the data, we will leave this part empty.
    - onCharacteristicChange(Modify): this method is called when the App(client) detects a change of the characterstics in the Gatt Server(i.e.: double tap’s notification received). When this happens, a push notification will be sent on the screen of the phone.
- Broadcast: This is the method that handles the interaction between UI and background (BluetoothLeService), using BroadcastUpdate method to update the data on the UI when any of the read,write or notify method has been called.

5 Testing and Observations

We have done our testing in parallel with the development stage. As in, we have made sure a small component works in a confined module before integrating it into the final product. We made sure the logic was sound, then implemented it. Every separate module implementation worked before joining it into the collaborative project module. Towards the end, small changes in code from development stage to implementation stage were made to make sure all modules are seamless.

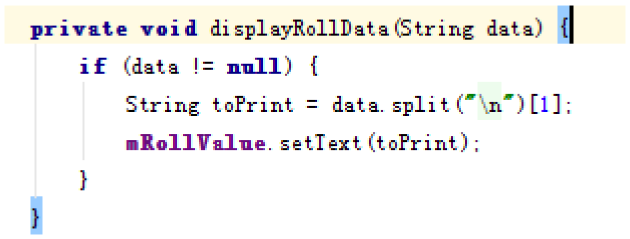
5.1 Tests on Discovery Board

The double tap sensitivity tests were first done with matlab and printf statements of the acceleration. We made a plot of when the board was tapped and noted the threshold (unfortunately unsaved). This caused some grief as different people have different strength when tapping the board. Thus, for our purposes, we have made our board as sensitive as possible to detect even the smallest of taps. For the final product, the tapping threshold is 40, 40, 30 mg in X, Y, and Z axis. However, this caused some other problems as the values were not decreasing fast enough. Our Kalman filter was the problem due to it being very stable and reducing the noise to display tilt angles accurately (less than 1 degree error). Unfortunately that means our acceleration values will not change as quick as we want. Thus, a small workaround was to add a delay function to skip over some readings (80ms;2 readings). This comes with its own share of problems as it causes a small hiccup over the seamless design, but it worked since it also took care of the rebound that happens. In our design, we use the absolute value of the difference, meaning the plot of the change in acceleration would never go below zero. However, that means every tap becomes a double tap in the eyes of the system (two peaks instead of a heartbeat). So to avoid confusion, it needed a delay value. When testing in the context of lab 3, the module worked as it should. In lab 4's context however, the system would always start at two double taps. This is because in lab 4, the threading starts the system so every acceleration axis starts at 0 instead of the expected 0, 0, 1000 mg. Even when changing the "previous" variable to all 0, the system would still break as the acceleration would increase rapidly to 0, 0, 1000 mg and pass the threshold in the Z axis. This did not cause trouble in the final design, as the notification only happens when we have connected the smartphone to the boards. Meaning, even though the Discovery board does give an extra notification when it first starts, the notification would need to go to the Nucleo board correctly at startup then to the smartphone at startup. Since we only connect the smartphone after the Discovery and Nucleo boards have started for some time, the extra notification is not a problem. We could have made this extra notification as a feature saying the boards are connected and working since it happens at startup. We increased the time frame of our tapping design to 1.5 second as we thought 1 second was too short for a double tap since it might not recognize the second tap. In the end, all double taps worked when demonstrated so it was an unfounded fear. The tests for the LEDs were done visually and were easily implemented. There were not a lot of trouble with the LEDs since it required more theory and logic instead of active testing.

5.2 BLE: Android App - Nucleo Testing

Firstly, we are required to set up connections with the Nucleo board with the BLE shield. However at first we noticed that Gatt Server (Nucleo)'s address is by default, the same with everyone, we've modified it to end with ":08", which is our group number to distinguish our connection from others, rather than the default set up of ":12".

UI display problem: when displaying the temperature, roll and pitch data, there are always some funny signs that appear right before the data value. After checking the data-fetching on "onCharacteristicRead", we realized that we only need the second portion of the data fetched(which is the numerical value that we need). So in the function where I displayed the data, I only used the 2nd half:



```
private void displayRollData(String data) {
    if (data != null) {
        String toPrint = data.split("\n")[1];
        mRollValue.setText(toPrint);
    }
}
```

Which resolved the issue successfully.

(For the specific Device specs, please refer to the appendix)

6 Conclusion

We’ve successfully demonstrated our IofT project with both displaying the Temperature, Roll and pitch data of the Discovery board on the Android app, and controlling the LED lights on the discovery board via Bluetooth Low Energy Connection. More features can be added in the future, as well as some modification towards a better UI display and more battery consuming algorithms.

6.1 Timeline and Breakdown of Teamwork

Design Week 1: Mar. 21st - Mar.27th
Summary: All members set up the Android Development Environment and finish the “HelloWorld” display on the phone and set up connections of the Bluetooth of the phone.

Member Name	Task
Yan	• Display ‘HelloWorld’ on your Android phone and setup the Bluetooth connections of the phone. • Read the documentations and get familiar with the BLE boards; connect all the required components between F4-discovery and BLE set. • Finish Android development(By Mar.27th)
Sheng	• STM32F4 Discovery : double tap, LEDs(By Mar.27th)
Chuan	• Setup BLE Connection (By Mar.27th) • Setup SPI Connection
Wei	

Design Week 2: Mar.28th - Apr.3rd (progress Demo starting Apr.4th)
Meeting: Monday, Mar.28th, afternoon.

- Summary: task confirmed:(as it is indicated in the chart below)

Member Name	Task
Yan	• Done: Android App able to connect to Nucleo, without BLE communication • TODO: • Android App development • Assist with SPI to establish communication between boards • Assist with the blueNRG BLE communication • Prepare for the progress demo
Sheng	• Android App: Helping Yan with some Android Studio java/xml structure (not much)

	• Prepare for the progress demo • Better double tap version. PWM working but flickering
Chuan	• SPI development • Prepare for the progress demo
Wei	• BlueNRG BLE communication • Prepare for the progress demo

Design Week 3: Apr.4th - Apr.11th(Final Demo on Apr.12th)
Meeting 1:Monday, Apr.4th, Morning (prepare for the final Demo)

Member Name	Task
Yan	Finished Android apps testing
Sheng	Finished PWM, threaded all Discovery visual updates. Finished testing all Discovery board peripherals.
Chuan	Finished SPI communication for Discovery board and Nucleo board. Finished bonus part.
Wei	Finished adding 3 Services as following: Double Tap Service (Notify), Environmental Sensor Service (Read), LED Service (Write) Finished Notify method Finished Read Request Callback or three of the attribute (Temperature, Pitch, Roll) Finished Attribute Modified Callback for LED Attribute, which will be trigger when client change char. value

7 Appendix

- Android phone & app(Gatt Client):
  Device Address: **18-59-36-1D-0E-B6**
- Nucleo Board Setup(Gatt Server):
  Name: BLE-G08
  Device Address: 03:80:E1:00:34:08 ({0x08, 0x34, 0x00, 0xE1, 0x80, 0x03})
  DT Service UUID: 02366e80-cf3a-11e1-9ab4-0002a5d5c51b
  (0x02,0x36,0x6e,0x80, 0xcf,0x3a, 0x11,0xe1, 0x9a,0xb4,
  0x00,0x02,0xa5,0xd5,0xc5,0x1b)

  Double Tap UUID: e23e78a0-cf4a-11e1-8ffc-0002a5d5c51b
  (0xe2,0x3e,0x78,0xa0, 0xcf,0x4a, 0x11,0xe1, 0x8f,0xfc,
  0x00,0x02,0xa5,0xd5,0xc5,0x1b)

Environment Service UUID: 42821a40-e477-11e2-82d0-0002a5d5c51b
(0x42,0x82,0x1a,0x40, 0xe4,0x77, 0x11,0xe2, 0x82,0xd0,
0x00,0x02,0xa5,0xd5,0xc5,0x1b)

Temperature Char UUID: a32e5520-e477-11e2-a9e3-0002a5d5c51b
(0xa3,0x2e,0x55,0x20, 0xe4,0x77, 0x11,0xe2, 0xa9,0xe3,
0x00,0x02,0xa5,0xd5,0xc5,0x1b)

Pitch Angle Char UUID: cd20c480-e48b-11e2-840b-0002a5d5c51b
(0xcd,0x20,0xc4,0x80, 0xe4,0x8b, 0x11,0xe2, 0x84,0x0b,
0x00,0x02,0xa5,0xd5,0xc5,0x1b)

Roll Angle Char UUID: 01c50b60-e48c-11e2-a073-0002a5d5c51b
(0x01,0xc5,0x0b,0x60, 0xe4,0x8c, 0x11,0xe2, 0xa0,0x73,
0x00,0x02,0xa5,0xd5,0xc5,0x1b)

Sample Service UUID: 8263e608-cf3a-11e1-9ab4-0002a5d5c51b
(0x82,0x63,0xe6,0x08, 0xcf,0x3a, 0x11,0xe1, 0x9a,0xb4,
0x00,0x02,0xa5,0xd5,0xc5,0x1b)

Sample Char UUID: 340a1b80-cf4b-11e1-ac36-0002a5d5c51b
(0x34,0x0a,0x1b,0x80, 0xcf,0x4b, 0x11,0xe1, 0xac,0x36,
0x00,0x02,0xa5,0xd5,0xc5,0x1b)