ECSE426 Microprocessor Systems

Fall 2017

Final Project

IoT –Sensor Data Management from Hardware to Cloud

Group 04

Mingrui Zhang: 260623296

Yanhao Gong: 260546543

Kechen Qian: 260559868

Xinyi Lu: 260567881

December 9, 2017

Table of Contents

1. Abstract

In this project, we focus on the interaction of embedded peripherals and sensors with cloud-enabled services, which is one of the main hallmarks of Internet of Things (IoT) design. All these things will be achieved by using a F4-Discovery board and a STM32F401RE Nucleo board. The main idea is using the F4-Discovery board to record the message generated by microphone. With the function of UART, which is short for Universal Asynchronous Receiver/Transmitter, we could build the communication between these two boards. For the STM32F401RE Nucleo board, it works as a blue tooth interface by the connection to an IDB04A1 BLE daughter board. After getting the message, we generate, STM32F401RE Nucleo board could send the message to our smart phone by blue tooth.

2. Problem Statement

The project aimed at developing a system to explore the interaction of embedded peripherals and sensors with cloud-enabled services. The project is mainly divided into four parts: the F4-Discovery board, the Nucleo board, Bluetooth Low Energy (BLE) application on smartphone and cloud services. The whole process will be introduced as follow:

- A peripheral microphone will be connected to discovery board to accumulate ADC value
- The discovery board will transmit the set of modified value to Nucleo board
- The Nucleo board will use built BLE function to transmit the ADC value to application on Android platform
- Once transmitting the data is finished, it will be send to our firebase through accessing cloud service.
- The cloud service will store the data and manipulate the files, as well as free access for clients to download.

To be more specific, the data will be collected at a particular frequency by discovery board once the blue button is pressed. The LED should be capable of indicating the beginning and end of a data recording process. The whole serial communication should be done via a UART connection under a maximized baud rate that we can achieve.

The BLE app on Android platform should be able to search for the Nucleo board and read the data transmitted from Nucleo board by BLE once getting paired. Since the application cannot communicate with the discovery board directly. The Nucleo board with a daughter board provides a way for user to modify and access the services and their unique attributes for the application.

Overall, the system combines multiple functions and physical devices as we discussed above. During the time of dealing with the project to meet all requirements, few challenges are encountered, such as

- ADC data collected by microphone has no variation

- Data loss while communicating between different devices
- Properly data transmitting and collecting frequency
- Properly calibrating New algorithm for Discovery board and Android application
- Signal processing on cloud service, especially for voice recognition

3. Theory and Hypothesis

3.1 ADC & Microphone

In this lab, we used an Electret Mic Breakout microphone (100Hz–10kHz) with a 60x mic preamplifier to amplify the sounds received. It translates the amplitude by capturing sound waves between two conducting plates in the microphone and converting them into electrical waves. This analog signal that is generated by the microphone can be fed directly to into the analog-to-digital converter of a microcontroller. Each breakout comes fully assembled and works from 2.7V up to 5.5V. A schematic diagram is shown below [1].

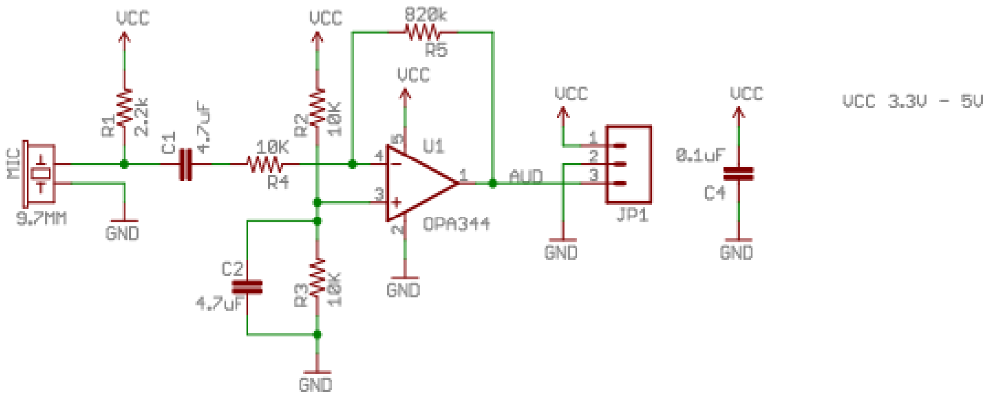


Figure. 1 Schematic Diagram of Electret Mic Breakout Microphone

As we can see from the diagram, there are three pins on the microphone including V_{cc}, ground, and Audio. V_{cc} is the pin to connect to a voltage source that has a magnitude between 3.3V and 5V. Audio is the pin to send analog signal amplified internally. In our lab, the sent signal would be fed into the ADC pin of the STM Discovery board to convert to discrete numbers that can be saved and further processed. The sampling rate for ADC will be set to 8K since it is a durable value for both ADC and microphone and additional timer is needed to control the starting of the conversion to achieve this rate.

3.2 USAT & USART

USAT is short for Universal Asynchronous Receiver/Transmitter. This technique is used in this lab to establish communication between Discovery board and Nucleo board. It can take bytes of data and transmits the individual bits in a sequential fashion. At the destination, a “receive” UART re-assembles the bits into complete bytes. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Comparing with parallel transmission through multiple wires, serial transmission of digital information (bits) through a single wire or other medium is less costly. [2] An example of digital transmission wave is shown in Figure. 2.

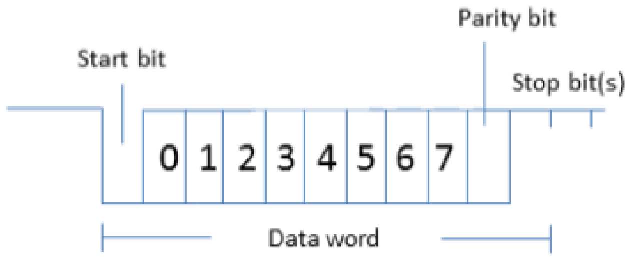


Figure. 2 Digital Wave Transmission Example

In this lab, USART which is the synchronous mode for UART is not used since there is no need for synchronous data transmission and Nucleo board does not support it. However, USART equals to UART when setting USART in asynchronous mode and this would be useful in our lab.

3.3 BLE

BLE (Bluetooth low energy), compared to the classic Bluetooth, is designed for lower power consumption, which allows our Android app to communicate with the BLE device based on STM Nucleo board. It is useful for devices that send data infrequently, or in small amounts and it has good standard for Internet of Things (IoT) applications as power-efficient, and devices don't typically need continuous connection. [3] The connection between Android phone and BLE device is in the mode of GATT server and GATT client, where the Android app is the GATT client and receive data from the GATT server, which is our BLE board that transfers sound data.

GATT is generic attribute profile specified for sending and receiving short pieces of data known as attributes over the BLE link. Inside the GATT, there is an attribute protocol that contains a universally unique identifier (UUID), which is a standardized 128-bit format for a string ID used to uniquely identify information. The attributes transported by this protocol are formatted as characteristics and services. Service is a collection of characteristics, and a characteristic contains descriptors that describe the characteristic's value. [4]

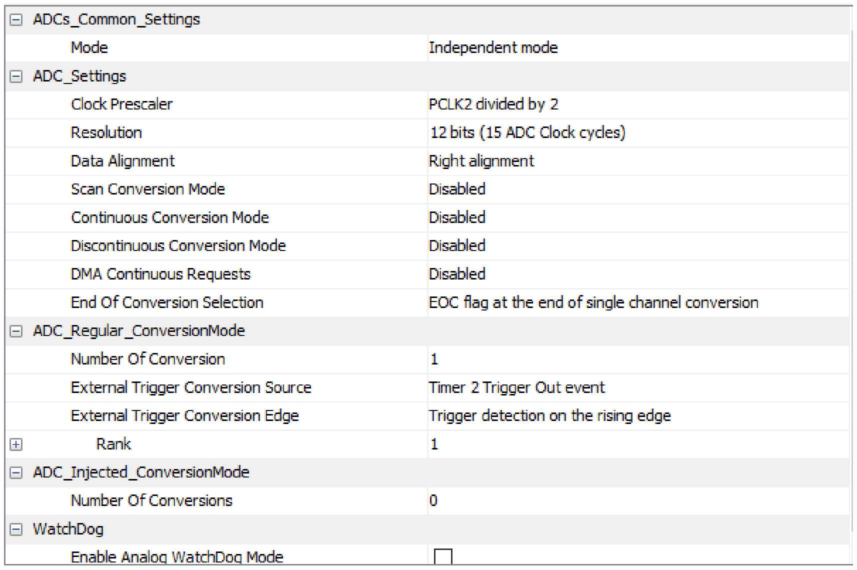
4. Implementation

To implement the whole system, group members are divided into software team and hardware team. Different targets within the software or hardware part were done by the cooperation between team members. Towards to the end of project, different codes were combined and modified accordingly. In addition, a detailed pinout diagram will be given in the appendix.

4.1 Discovery Board:

1. Microphone:

To implement the microphone, a STM32 CubeMX project is created to simplify the work. ADC 2 for Channel 11 is chosen to avoid possible confliction between future timer or UART setup. The detailed setting is shown below.



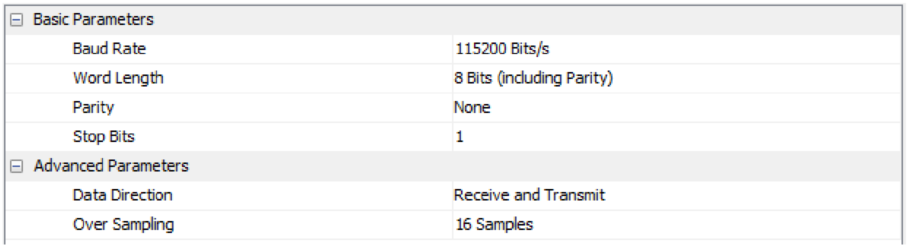
⊟ ADCs_Common_Settings	
Mode	Independent mode
⊟ ADC_Settings	
Clock Prescaler	PCLK2 divided by 2
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Disabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion
⊟ ADC_Regular_ConversionMode	
Number Of Conversion	1
External Trigger Conversion Source	Timer 2 Trigger Out event
External Trigger Conversion Edge	Trigger detection on the rising edge
⊞ Rank	1
⊟ ADC_Injected_ConversionMode	
Number Of Conversions	0
⊟ WatchDog	
Enable Analog WatchDog Mode	☐

Figure. 3 ADC 2 Settings

Instead of using polling mode, we used Timer 2 to control the rate of conversions. Timer 2 is set to be in interrupt mode which has an interrupt frequency of 8K. Once Timer 2 enters in the IRQ handler, ADC will do one conversion and save the converted value. The resolution is also set to be 12 bits to get the best conversion quality. Inside the project, the saved conversion value will be extracted by using HAL_ADC_GetValue(&hadc2) command.

2. UART:

UART is also initialized inside the same CubeMX project. UART 5 is chosen and the mode is selected to be asynchronous. The parameters are shown in Figure. 4. The global interrupt for UART 5 is enabled this time to activate the IT method for UART.



⊟ Basic Parameters	
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
⊟ Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples

Figure. 4 UART 5 Settings

Thus, the interrupt method for UART transmission or receive can be used. However, in Discovery board, we will only use the transmission functionalities. For example, HAL_UART_Transmit_IT (&huart5, trans_buffer, 1) is used to transmit one element inside the trans_buffer array. Once this method is called, the complier will wait until the next interrupt for UART and transmit the data. When the transmission is completed, HAL_UART_TxCpltCallback will be called if user defines it in other files.

4.2 Nucleo board:

1. UART:

Since the Nucleo board project which contains the BLE core code is given, UART is best set up within the given Keil project. The setting will be similar to UART 5 in Discovery board, however, with typed code. In addition, we shall use receive functionalities. For instance, HAL_UART_Receive_IT (&huart6, rece_buffer, 1) is used to receive one element transmitted from another board and store it in rece_buffer. Once this method is called, the complier will wait until the next interrupt for UART and trying to receive the data. When the data is detected and received, HAL_UART_RxCpltCallback will be called if user defines it in other files.

2. BLE

Most of the BLE code is implemented for us, thus, we need not to initialize the settings. In general, the phone is the master that scans for the Bluetooth signal and Nucleo board is a slave waiting for the connection and accepting any incoming request. More specific, the board can only connect to a single master in BLE. For sending data to the phone, a new service should be added and only one characteristic service is needed to be added to the service to enable the update method that continuously transmit the data by using aci_gatt_update_char_value. The code structure will be similar to the example service and UUIDs are created by following certain rules that are recognizable by phone.

4.3 Discovery to Nucleo board:

In our method, we first wait for the blue button on Discovery board press event and Nucleo board will always wait for receiving the data from Discovery board. Once the button is pressed, the Blue LED on Discovery board will turn on to indicate the starting of the sound record. The recoding will last around 2 second and the Orange LED will turn on to indicate the ending of the sound record and the beginning of the data transmission. 10000 ADC sound data points will be transmitted one by one by using UART. However, before the Discovery board transmitting the real sound data, it will first transmit a key data to Nucleo board representing the beginning of the data transmission. Once the Nucleo board receives the key data, the Green LED of it will turn on to indicate the starting of the data reception. Once all data are transmitted, the Green LED on Discovery will turn on and the one on the Nucleo board will turn off. After these, Nucleo board will begin to set up BLE. The flow diagram of the logic is shown in Figure. 5.

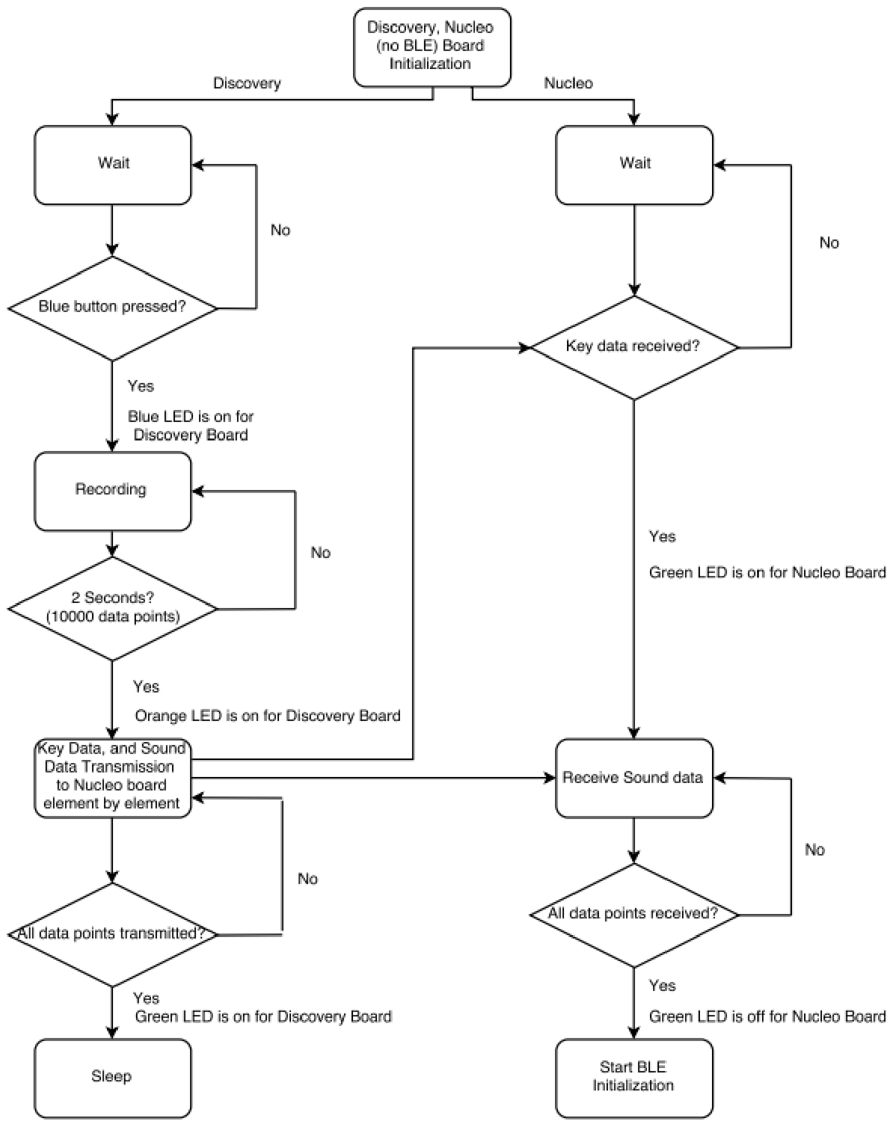

Discovery, Nucleo (no BLE) Board Initialization
Discovery
Nucleo
Wait
No
Blue button pressed?
Yes
Blue LED is on for Discovery Board
Recording
No
2 Seconds? (10000 data points)
Yes
Orange LED is on for Discovery Board
Key Data, and Sound Data Transmission to Nucleo board element by element
No
All data points transmitted?
Yes
Green LED is on for Discovery Board
Sleep
Wait
No
Key data received?
Yes
Green LED is on for Nucleo Board
Receive Sound data
No
All data points received?
Yes
Green LED is off for Nucleo Board
Start BLE Initialization

Figure. 5 Board Communication Logic Flow Chart

4.4 Android App

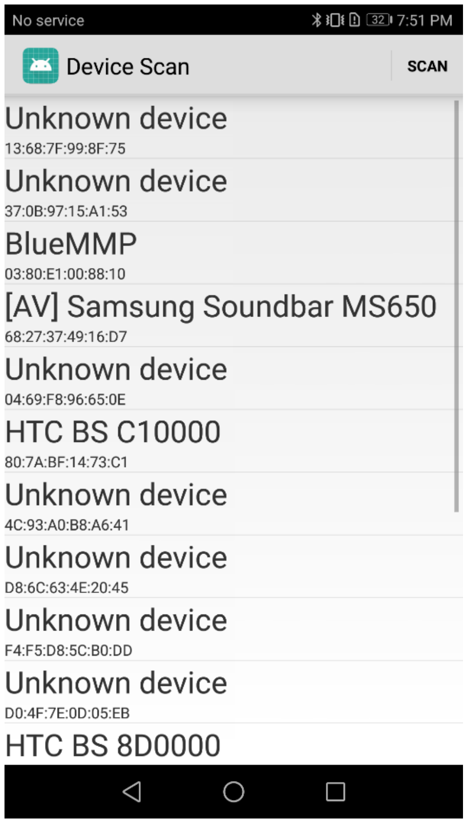
The android app is designed based on the sample project provided in GitHub of "Google Samples" called BluetoothLeGatt. The app has following classes:

- DeviceScanActivity: control the activity on the first UI page to scan and display the list of all the available BLE devices with its name and address.
- BluetoothLeService: handle all GATT events including callback, service discovering, characteristic data read, update and notification.

- ControlActivity: control activities on the second page after choosing the device in scan page. Receive the change of characteristic's values and show in text labels. Send raw data as text file into firebase cloud storage when trigger the button click event.
- GattAttributes: store all service and characteristic UUID.

1. UI Design

There are two UI pages in this android app written by xml file in res/layout folder. The first page shown in Figure. 6 is to display all available BLE devices when scanning. The menu button in the upper right corner is to control to start or stop scanning.


No service 7:51 PM

Device Scan SCAN

Unknown device
13:68:7F:99:8F:75

Unknown device
37:0B:97:15:A1:53

BlueMMP
03:80:E1:00:88:10

[AV] Samsung Soundbar MS650
68:27:37:49:16:D7

Unknown device
04:69:F8:96:65:0E

HTC BS C10000
80:7A:BF:14:73:C1

Unknown device
4C:93:A0:B8:A6:41

Unknown device
D8:6C:63:4E:20:45

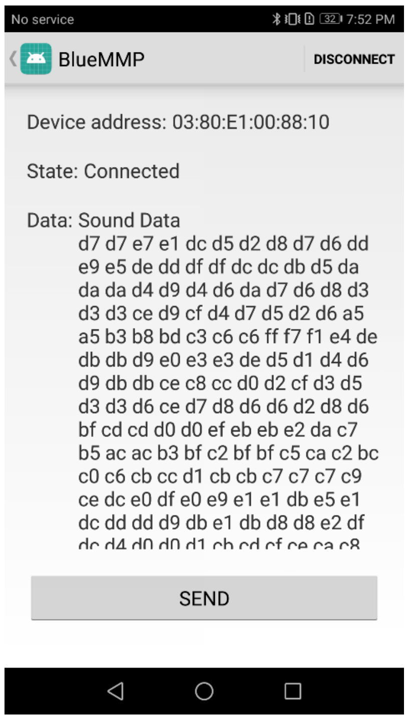
Unknown device
F4:F5:D8:5C:B0:DD

Unknown device
D0:4F:7E:0D:05:EB

HTC BS 8D0000

Figure. 6 Device Scan UI

The second page (Figure. 7) shows the detail information about the connected device and received data. The first line shows the mac address of the target device. The second line shows the connection state between app and device. If it is connected and the app found all services and characteristics, the words in that label would be "Connected", otherwise, it would be "Disconnected". The third item in the middle displays all data received from the Nucleo board. In this project, the data is in hex format of unsigned 8-bit sound data buffer. The "Send" button in the bottom is designed to upload all data to cloud storage once being clicked.


No service 7:52 PM

BlueMMP DISCONNECT

Device address: 03:80:E1:00:88:10

State: Connected

Data: Sound Data

```
d7 d7 e7 e1 dc d5 d2 d8 d7 d6 dd
e9 e5 de dd df df dc dc db d5 da
da da d4 d9 d4 d6 da d7 d6 d8 d3
d3 d3 ce d9 cf d4 d7 d5 d2 d6 a5
a5 b3 b8 bd c3 c6 c6 ff f7 f1 e4 de
db db d9 e0 e3 e3 de d5 d1 d4 d6
d9 db db ce c8 cc d0 d2 cf d3 d5
d3 d3 d6 ce d7 d8 d6 d6 d2 d8 d6
bf cd cd d0 d0 ef eb eb e2 da c7
b5 ac ac b3 bf c2 bf bf c5 ca c2 bc
c0 c6 cb cc d1 cb cb c7 c7 c7 c9
ce dc e0 df e0 e9 e1 e1 db e5 e1
dc dd dd d9 db e1 db d8 d8 e2 df
dc d4 d0 d0 d1 cb cd cf ce ca c8
```

SEND

Figure. 7 Service Info UI

2. Logic

The overall logic flow diagram is shown in Figure. 8. Android app is mainly responsible for device connection, data receiving and cloud storage uploading.

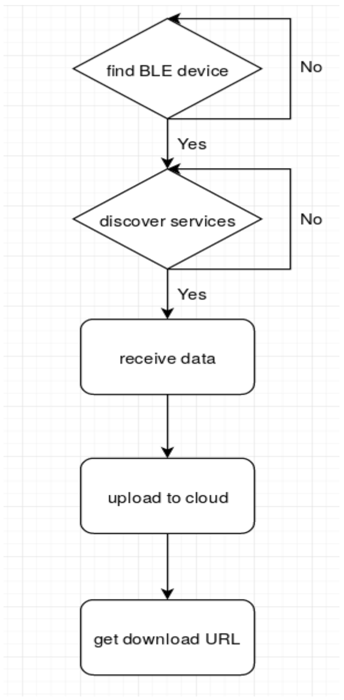
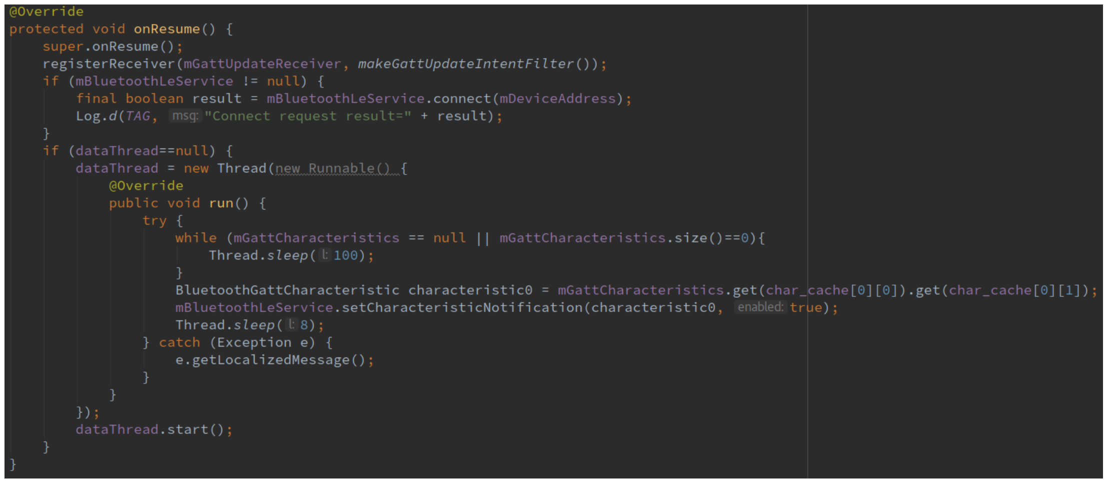


Figure. 8 Flow Diagram of Android App

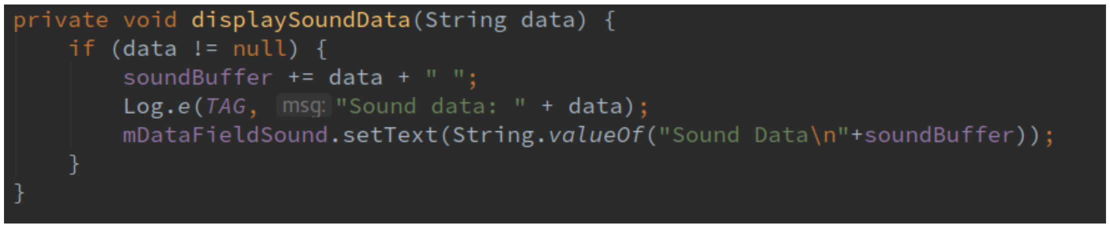
The first step after initializing the Bluetooth low energy is to collect the list of devices that support to BLE and show their name and address. Followed by the documentation in the android developer website, all information of devices is stored in "LeDeviceListAadpter" class and added into the list in the callback function. Click event is handled by "onListItemClick" function, which will stop scanning and direct to the next page. If the target device is not found in the list, we can press the "scan" menu button to rescan until we find it.

After the target device is chosen, we would try to establish a connection between android app and BEL device by pressing the menu button in the upper right corner. If the state is still disconnected, we will retry until it is connected. Once the app connects to the device, it will locate the position of the service and characteristic that we are looking for, which contains UUID of sound characteristic, and store into a temporary array. When the app is running, the thread in "onResume" function, show in Figure. 9, will start keeping calling the notification function with the characteristic in the temporary array as a parameter in order to update received data when the characteristic value changes and display sound data buffer with the function shown in Figure. 10.



```
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mGattUpdateReceiver, makeGattUpdateIntentFilter());
    if (mBluetoothLeService != null) {
        final boolean result = mBluetoothLeService.connect(mDeviceAddress);
        Log.d(TAG, msg: "Connect request result=" + result);
    }
    if (dataThread==null) {
        dataThread = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    while (mGattCharacteristics == null || mGattCharacteristics.size()==0){
                        Thread.sleep( l: 100);
                    }
                    BluetoothGattCharacteristic characteristic0 = mGattCharacteristics.get(char_cache[0][0]).get(char_cache[0][1]);
                    mBluetoothLeService.setCharacteristicNotification(characteristic0, enabled: true);
                    Thread.sleep( l: 8);
                } catch (Exception e) {
                    e.getLocalizedMessage();
                }
            }
        });
        dataThread.start();
    }
}
```

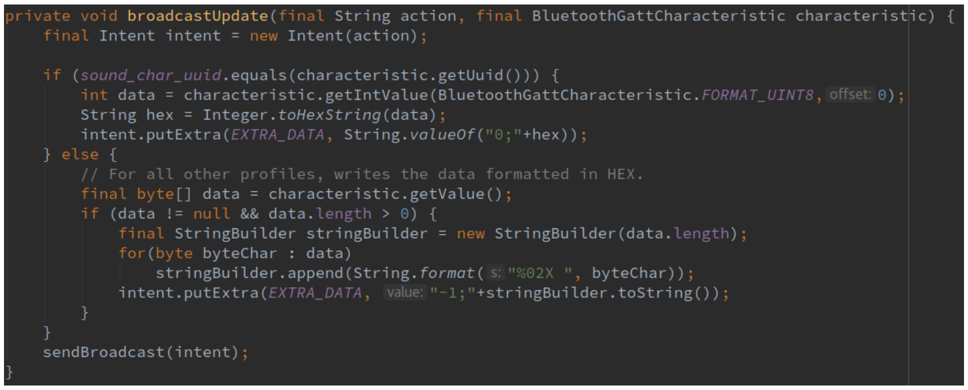
Figure. 9 OnResume Function with a Runnable Thread



```
private void displaySoundData(String data) {
    if (data != null) {
        soundBuffer += data + " ";
        Log.e(TAG, msg: "Sound data: " + data);
        mDataFieldSound.setText(String.valueOf("Sound Data\n"+soundBuffer));
    }
}
```

Figure. 10 Function to Display Sound Data Buffer

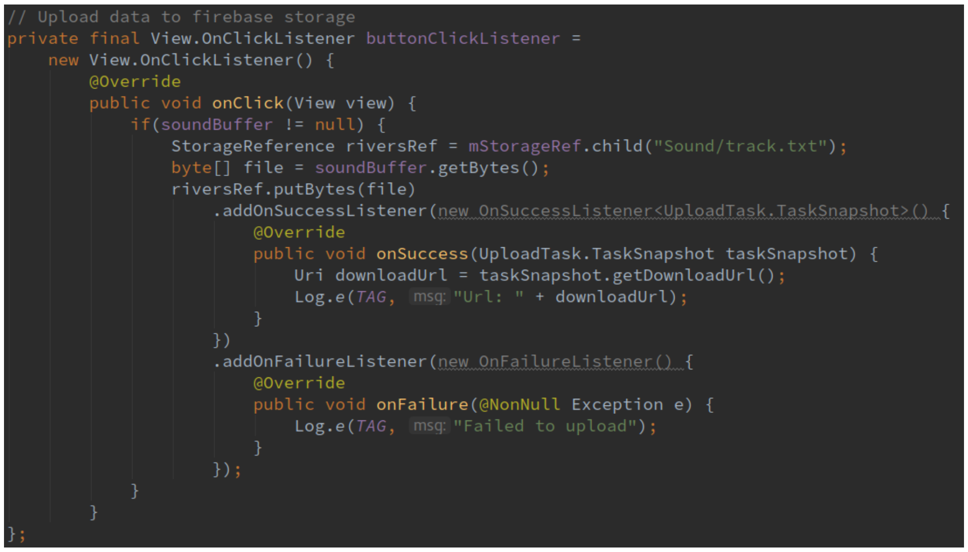
In the period of data receiving, the "broadcastUpdate" function in Figure. 11 is called frequently. Data is stored into intent object with string format to transmit to broadcast receiver. If the characteristic UUID is for sound service, the data is captured in integer and converted into hex format. Broadcast receiver function in the "ControlActivity" class will read the data in intent object when there is a characteristic notification.



```java
private void broadcastUpdate(final String action, final BluetoothGattCharacteristic characteristic) {
    final Intent intent = new Intent(action);

    if (sound_char_uuid.equals(characteristic.getUuid())) {
        int data = characteristic.getIntValue(BluetoothGattCharacteristic.FORMAT_UINT8, offset: 0);
        String hex = Integer.toHexString(data);
        intent.putExtra(EXTRA_DATA, String.valueOf("0;"+hex));
    } else {
        // For all other profiles, writes the data formatted in HEX.
        final byte[] data = characteristic.getValue();
        if (data != null && data.length > 0) {
            final StringBuilder stringBuilder = new StringBuilder(data.length);
            for(byte byteChar : data)
                stringBuilder.append(String.format( s: "%02X ", byteChar));
            intent.putExtra(EXTRA_DATA,  value: "-1;"+stringBuilder.toString());
        }
    }
    sendBroadcast(intent);
}
```

Figure. 11 Function to Broadcast the Updated Data

When data transmission is finished, we could click the send button to upload received data into firebase cloud storage. The event listener, shown in Figure. 12, put sound data buffer into a byte array and upload it as a text file through firebase build-in function. If it success to upload the file, the callback function is triggered and will print out a URL for downloading the file in cloud storage.



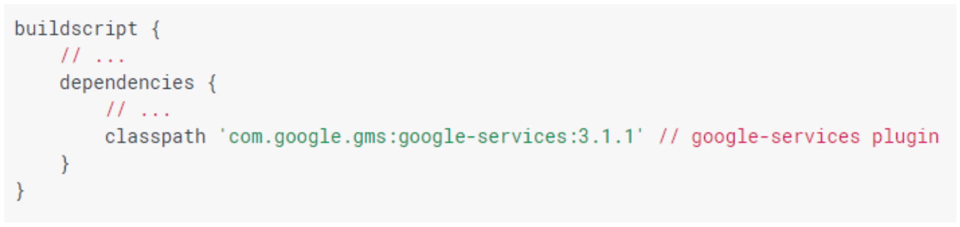
```java
// Upload data to firebase storage
private final View.OnClickListener buttonClickListener =
    new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if(soundBuffer != null) {
                StorageReference riversRef = mStorageRef.child("Sound/track.txt");
                byte[] file = soundBuffer.getBytes();
                riversRef.putBytes(file)
                    .addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
                        @Override
                        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                            Uri downloadUrl = taskSnapshot.getDownloadUrl();
                            Log.e(TAG,  msg: "Url: " + downloadUrl);
                        }
                    })
                    .addOnFailureListener(new OnFailureListener() {
                        @Override
                        public void onFailure(@NonNull Exception e) {
                            Log.e(TAG,  msg: "Failed to upload");
                        }
                    });
            }
        }
};
```

Figure. 12 ButtonClickListener Function to Upload Data to Cloud Service

For this android app, we did not implement the cloud authorization checking. Since all code will be uploaded to the GitHub and the android phone is not belonging to our developer, it is much more insecure to disclose people google account in the code or other's phone than just open the storage to public in the cloud service.

4.5 Cloud Service

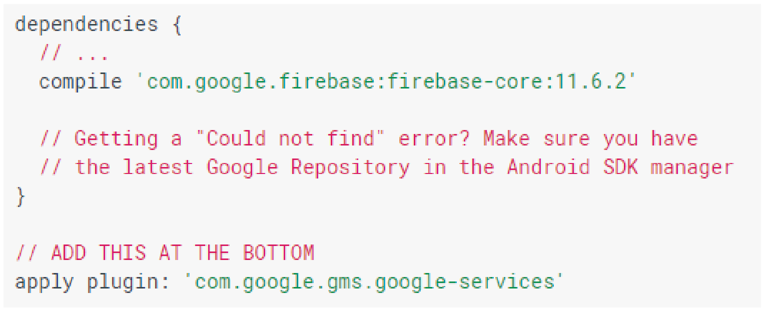
We used the Firebase cloud service from Google. Basically, the cloud should store Discovery board data, manipulated files, set authorization for clients and some other features like signal filtering. Once we wrote our own BLE application on Android platform, we need to add the firebase service to our app and enable the storage service on firebase. [5] The prerequisite of this operation is to download the json file from firebase after creating your project on it. Then, we can add google gms service to 'build.gradle' file of project level as follow:



```
buildscript {
    // ...
    dependencies {
        // ...
        classpath 'com.google.gms:google-services:3.1.1' // google-services plugin
    }
}
```

Figure. 13 Build.gradle Service

Under the dependencies section of 'build.gadle' file of app level, we need to add codes as follow:

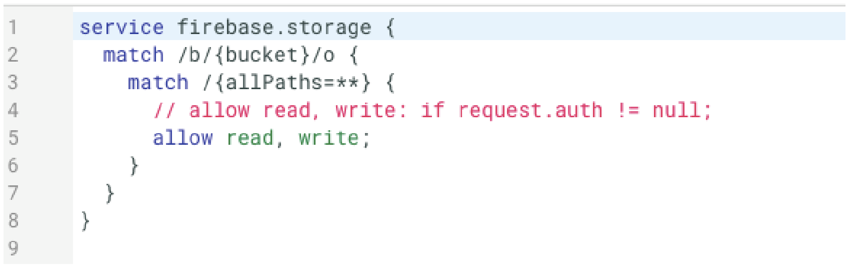


```
dependencies {
  // ...
  compile 'com.google.firebase:firebase-core:11.6.2'

  // Getting a "Could not find" error? Make sure you have
  // the latest Google Repository in the Android SDK manager
}

// ADD THIS AT THE BOTTOM
apply plugin: 'com.google.gms.google-services'
```

Figure. 14 Dependencies Code

The version of the google service should be upgraded to meet your software environment.

After linking the BLE application to firebase, we need to enable the storage in firebase to store any files uploaded. Under the RULES section in storage, we need to change the script to the one shown below:



```
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      // allow read, write: if request.auth != null;
      allow read, write;
    }
  }
}

```

Figure. 15 Storage Service

The next step is to add the function for uploading the files transmitted from discovery board, in our case the sound data. Under the 'ControlActivities.java' file we need to import several modules as follow:

```java
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.net.*;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.storage.*;
import java.io.*;
```

Figure. 16 Library Imported

Same as some several variables:

```java
FirebaseStorage storage = FirebaseStorage.getInstance();
// Create a storage reference from our app
StorageReference storageRef = storage.getReference();
StorageReference ourFileRef = storageRef.child("boardData/testfile.txt");
```

Figure. 17 Storage Reference Variables

In our design, we built our own function to encode sound data into Hex value and 1000 samples will be stored in a text file. The text file will be uploaded to storage in firebase after the 'SEND' button is pressed in BLE application and free for download.

The text file can be converted to '.wav' file using a software called AUDACITY and played after that. The figure attached below is the sound spectrum.

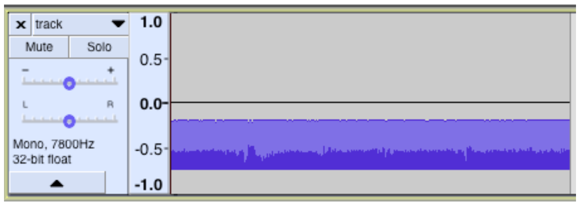


Figure. 18 Wave Form Diagram

Signal processing like filter can be modified within firebase. However, this part is not realized in our design due to the time limit, we will continue to work on this part to obtain a satisfied sound output, as well as voice recognition.

5. Testing and Observations

In our final project, we implement each part separately and simultaneously. Firstly, we try to ensure every part of this design would work properly. We do the testing for the code respectively and then gather them together. The final step also needs some adjustment of the codes before to solve the communication problems of each part.

To make sure the microphone work, we connected it to the Discovery Board. We used the Discovery Board to record the message generated by the microphone and printed them in the debug mode. As we could see some different data displayed, this meant the microphone worked properly.

After that we started to test whether UART worked well. We added some LED implementations into the codes on the discovery board. They worked as the signal of whether the data were recorded or transmitted correctly.

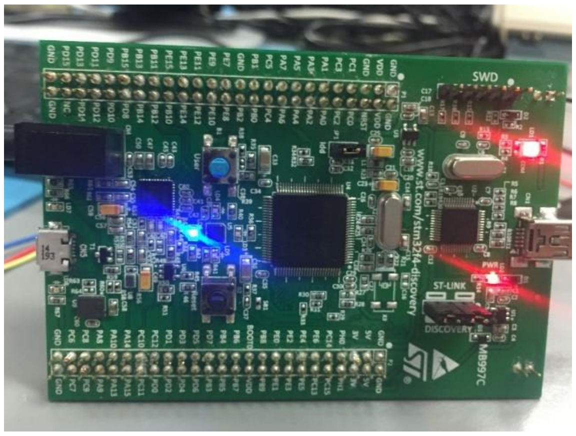


Figure. 19 Blue Button Pressed

From the Figure 19, we could see that the Blue LED was on, which meant the discovery board was recording the data generated by microphone.

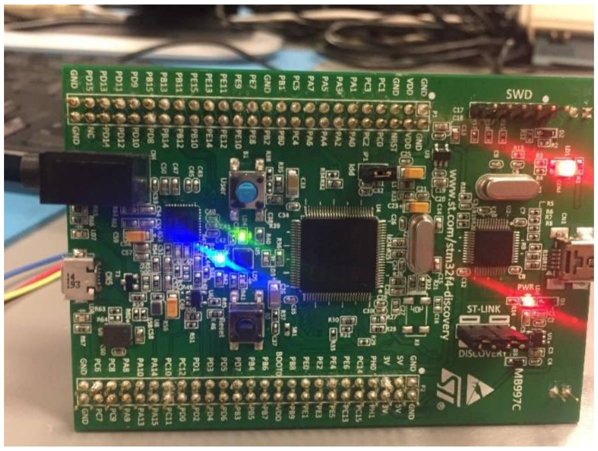


Figure. 20 Data Transmission & Recording Complete

After some time, from the Figure 20, we could see that the green light was on. In this case, the data had been recorded. Note that the orange light is bright as well, however, is blinking as we intend. This represents the recording is complete.

To test if the message could be transmitted to the STM32F401RE Nucleo board, we used an Oscilloscope to test whether we could get the output of the board.

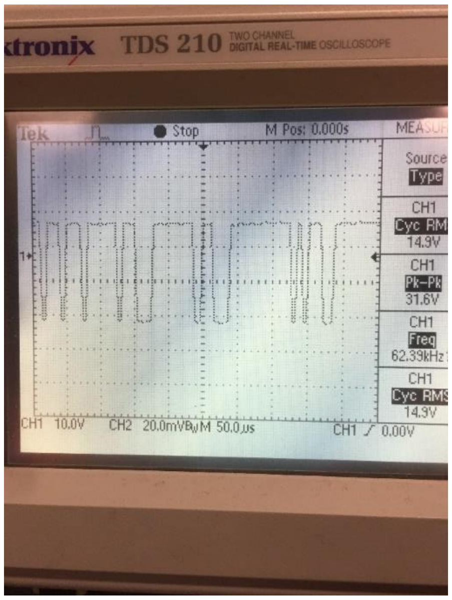


Figure. 21 Oscilloscope for Data Transmission

From the Figure 21, we could see that there were some data from the output pin, which meat we were correct in this part.

The next step is to test the STM32F401RE Nucleo board, we would like to know if this board could receive data correctly. We firstly used the example project to test whether the blue tooth worked or not. After everything initialized, we started to do the data transmission. As we could see the Figure 22 below, the green LED light was on, this meant the board had received the key data. After some time, the green light would turn off and told us that the data has been received.

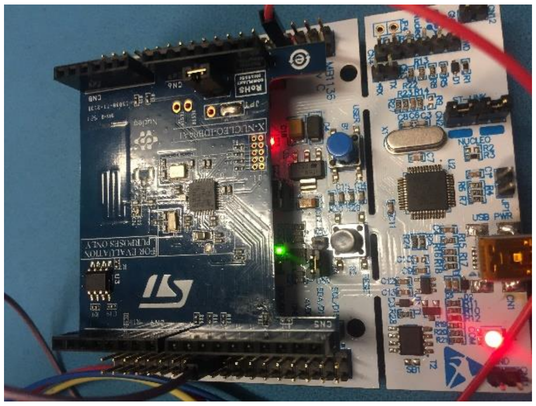


Figure. 22 Data Reception

As mentioned before, the final step was to transfer the data from the board to the smartphone. After we got the ACC data on the smartphone, we uploaded the data to the cloud. However, we did not complete the sound processing procedure.

6. Conclusion

During this final project, we could successfully get the data from the microphone and could send the data to the cloud. However, the final data we got was not that accurate. There are several reasons for this problem. Firstly, we did not have a filter for the raw data, this could add more errors to our data. The second reason for this is that for the data format, we change the raw data in hex by using AUDACITY to unsigned 8 bits directly without using .wav file format. This will cause a problem in the data encoding. If possible, we will add more features including these unimplemented features.

7. Work Break-down

Team Members	Work Done	Time
Yanhao Gong	• Android Application development • Cloud Service Set up • Sound data transmission between app and cloud testing	2:00 pm to 6:00pm every single day from Nov 22 to Dec 3. Meeting are within the Lab time
Xinyi Lu		
Mingrui Zhang	• Discovery and Nucleo board set up and algorithm development • Sound data transmission testing • UART implementation and sound recording	
Kechen Qian		

8. References

[1] S. Breakout, S. Arduino, B. (White), B. Straight and J. 10, "Electret Mic Breakout Board Hookup Guide - learn.sparkfun.com", Learn.sparkfun.com, 2017. [Online]. Available: https://learn.sparkfun.com/tutorials/electret-mic-breakout-board-hookup-guide?_ga=2.51630830.635717613.1512852221-1373513439.1509393769. [Accessed: 11- Dec- 2017].

[2] From ECSE 426 Slides

[3] From ECSE 426 Slides

[4] B. Energy, "Bluetooth Low Energy | Android Developers", Developer.android.com, 2017. [Online]. Available: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html. [Accessed: 11- Dec- 2017].

[5]"Add Firebase to Your Android Project | Firebase", *Firebase*, 2017. [Online]. Available: https://firebase.google.com/docs/android/setup?authuser=0. [Accessed: 11- Dec- 2017].

9. Appendix

Device	Pinout
UART 5_TX (Discovery)	PC 12
ADC 2 (Discovery)	PC 1
USART 6_RX (Nucleo)	PC 7

Appendix. 1 Device Pinout

Name: BlueMMP
Device Address: {0x10, 0x88, 0x00, 0xE1, 0x80, 0x03}
DATATRANSMIT_UUID
(0x99,0x36,0x6e,0x80, 0xcf,0x3a, 0x11,0xe1, 0x9a,0x38, 0x00,0x02,0xa5,0xd5,0xc5,0x1b)
DATATRANSMIT_CHAR_UUID
(0x98,0x36,0x6e,0x80, 0xcf,0x3a, 0x11,0xe1, 0x9b,0x39, 0x00,0x02,0xa5,0xd5,0xc5,0x1b)

Appendix. 2 BLE Device Setup