# SpectraMini Developer Documentation

## Overview

The STM32 interacts with all the physical components, it is programmed in C using the STM32CubeIDE. The IDE communicates with an ST-LINK V2 that is connected to the computer via usb and which also connects to the STM32 via the programming pins. The Android SpectraMini app is programmed in Java using Android Studio. The Bluetooth Module is programmed via a serial connection to an Arduino Uno that has C code which configures the settings. When using the android app on a new phone, make sure to both connect the Bluetooth in the settings, and to also give the app any permissions necessary.

## Microcontroller Function

The function of the microcontroller is to combine all the peripheral components into a single working system. The STM32 produces and changes all the signals according to the CCD Specifications to drive it and receive data from the CCD. The STM32 also interacts with the Bluetooth module to receive and transmit data with a Bluetooth-enabled application. If you wish to explore the details of the operation of the STM32, read this article while following along in my code and comments.

Some important notes for uploading code to the STM32: all calculations done for timing intervals are made with an 84 MHz clock speed, all code must be uploaded via the ST-LINK (I used the magenta V2 stick), there may issues where when the settings of ioc file are changed it may delete important code when making the changes.

You may notice that there is a lot of buffer casting, so here's an explanation of the lifecycle of the CCD data:

We first initialize an array of 16-bit values, the CCD will give us 12-bit integers, containing those values in a 16-bit integer makes the process easy:

```
#define CCDBuffer 3694  //Pixels in a line read
volatile uint16_t CCDPixelBuffer[CCDBuffer];
```

Then we give the array in 32-bit form to the ADC, however the ADC is initialized with 12-bit resolution, meaning that it will fill in 12-bits every 16-bits since it increments by powers of 2. The data should look something like this in memory (disregarding the commas): 0000111111111111, 0000111111111111, 0000111111111111…

```
HAL_ADC_Start_DMA(&hadc1, (uint32_t*) CCDPixelBuffer, CCDBuffer);
```

Finally, we send the bits through UART to the Bluetooth module. The UART protocol only allows us to send the bits through an 8-bit array, so we cast our 16-bit array into an 8 bit array, thus turning the data representation into something like: 00001111,11111111,00001111,11111111,00001111,11111111…

Accordingly, since the increment of the buffer has changed by the casting, our array's length is doubled.

```
HAL_UART_Transmit_IT(&huart1, (uint8_t*)CCDPixelBuffer, CCDBuffer*2);
```

## Communication Protocols

The current protocols that I have set up to work with Bluetooth Classic are as follows. First, register the device in the Android Settings for Bluetooth, if it prompts you for a password, it is 1234. To communicate with the device, you must then connect to the device through your program. You will be successfully connected to the device once the Bluetooth module starts blinking noticeably slower.

To request data, send a string with a lowercase s as the first letter followed by as many string terminators (ascii 0, not '0') until you reach a length of 20 total characters in the string. Send those 20 bytes to the CCD and it should spit out 7388 integers of 8 bits each. Each two 8-bit integers correspond to one 16-bit unsigned integer, so you'll have to reconstruct the 16-bit integers from each two 8-bit integers giving you effectively 3694 data points. Some of the ends of that data will be dummy values, remove them according to the TOSHIBA data sheet.

To request an integration time change, send a string with a lowercase e as the first letter followed by your number (an integer greater than 9), followed by as many string terminators as needed until a length of 20 is reached. Send that byte array and note that the next read should be discarded as it may contain artifacts from the change in integration time.

When developing the algorithm for converting the 8-bit integers to 16-bit integers, I recommend you change the STM32's code and use the debug array when sending so that it is easier to see and confirm you are getting the correct values. I do a bit of tweaking of the values later in the code depending on the power the CCD is getting and the graphing software to set the range to (0,1). Here's some examples of how I did the conversion in a few different languages:

Python (Left)

```python
def getData():
    data = ser.read(linereadN)
    dataArr = []
    for i in range(0, len(data)-2, 2):
        dataArr.append(3150 -
            ((data[i+1] << 8) + data[i])
        )
    dataArr = dataArr[dataStart:dataEnd]
```

Java (Right)

```java
private final int numData = 3694 * 2;
3 usages
private byte[] mmBuffer = new byte[numData];
1 usage
public void gatherData() {
    System.out.print("Start Read\t");
    try {
        for (int i = 0; i < numData; i++) {
            int c = in.read();
            if (i == 0 ) { System.out.print("Read First\t"); }
            if (c == -1) {
                System.out.println("End Of Stream Hit");
                break;
            }
            mmBuffer[i] = (byte)c;
        }

    } catch (Exception e) {
        System.out.println("Data Gather Failed");
        System.out.println(e);
        requestStop();
    }
    System.out.print("End Read\t");
}
```

# Android SpectraMini Project Overview

MainActivity.java

Starts up the ConnectBluetoothCycleThread which continuously attempts to connect to the Bluetooth device and removes data content from the graph to give the user suggestions. Once connected, starts up the UpdateGraphThread which will continuously request, gather, and graph data from the STM32 as long as the Bluetooth connection is active. If the UI requests an integration time change, then it also handles sending the new integration time and updating the graph accordingly. If the Bluetooth connection is interrupted, then the thread will start up the ConnectBluetoothCycleThread and end itself. This cycle between the two threads continues until the program is closed.

Bluetooth Setup

Bluetooth Requests and Transmission

Graphing Data

Saving Graph Data to Excel File Format

Updates Integration Time

HomeFragment.java

UI interaction/Functions

Graph Setup

Integration Time formatting

Screenshot Function

Developer Notes:

I used multithreading quite liberally in this portion of the project and believe that perhaps it can be reduced for simplicity if necessary. Certain libraries are slow with their graphing features which is why I used the MPAndroidChart library, it is arbitrarily fast. I've also tested CorePlot on IOS which updates smoothly with a high volume of data if you disable fancy drawing techniques such as Bezier curves before graphing.

## Setting up a new HC-05

To upload the settings to the HC-05, first upload the Bluetooth Module Programming program onto an Arduino (preferably an Arduino uno). Once it is uploaded, you can connect the respective pins described in the ino sketch between the HC-05 and the Arduino (make sure that you press the little black button as the HC-05 is getting power so that the device is entering settings mode, if it blinks fast then you are in transmission mode, if it blinks slowly you are in settings mode. If you are not in settings mode, then no settings will be changed!). Once those pins are connected, launch the serial monitor. Launching the serial monitor restarts the Arduino so that it runs the setup code again, which will effectively program the Bluetooth module such that its name will be SpectraMiniCCD and the baud rate will be 230400 (the highest baud that the STM32 can handle). Once the module is set up, connect it to the STM32, power the STM32, then power off the STM32, then power it on again. Following this process should achieve a working HC-05 component.

Now you can connect it to the android, and it should work. If when you open the app and it doesn't say Bluetooth Connected, then the Bluetooth was not setup properly. If it does connect but data is not being displayed to the screen, then try unplugging and replugging it in again. If it continues to not work, then something may be wrong somewhere along the communication protocol.