TABLE OF CONTENTS

Android GATT Server and Arduino GATT Client Communications

This document details the interaction between the Android GATT server and Arduino slave / client. It will discuss typical interactions between the two during the interval of a connection, from advertising on the client side, to disconnection on both the client and server sides.

For more details on the methods called, please refer to the source code in the repository and the official documentation BGLIB provided by Bluegiga here:

https://www.bluegiga.com/en-US/products/bluetooth-4.0-modules/ble113-bluetooth--smart-module/documentation/

For details on the frequency sweep parameters, please refer to the AD5933 datasheet here:

http://www.analog.com/static/imported-files/data_sheets/AD5933.pdf

Advertisement (Client Only)

Client

BGLIB callback method:

```
void my_ble_evt_system_boot(const ble_msg_system_boot_evt_t *msg) {
…
}
```

On boot up, the Arduino system initializes it's required variables, and calls its setup function. The setup function performs the following tasks:

1. Starts Arduino I2C library (Wire)
2. Uses AVR code to clear bits in port
3. Resets the AD5933, set default parameters and get its gain factor.
4. Initializes status LED
5. Initializes BLE112 reset pin
6. Sets up BGLIB event and status handlers
7. Opens hardware serial for BLE connection.
8. Boots Ble112
9. Writes default parameters set earlier to BLE characteristics (sample rate & frequency sweep parameters)
10. Starts timer inter interrupt at default sample rate: 50 Hz (20 microseconds)

The BGLIB boot command sets the advertising parameters for the module, including the MAC address and checks if any given command timed out. If no time out occurred, the boot was successful and any scanning BLE device can discover the GATT client.

Currently, the connection is unencrypted. This will be updated later.

Connections and Disconnections (Server and Client)

Server

Connection:

On the server side, a connection is initiated when after scanning, the user taps on the client from a provided list. This creates a new intent to start the DeviceControlActivity class, which in turn is bound to by BluetoothLeService.

Once this connection is established, the server loops through all available services and characteristics on the client, utilizing bit masking to determine which characteristics are which. It especially finds and assigns variables to 3 characteristics to exposed by the GATT client:

1. GATT_HANDLE_C_BIOIMPEDANCE_DATA
2. GATT_HANDLE_C_SAMPLE_RATE
3. GATT_HANDLE_C_AC_FREQ

Once identified, the latter 2 are immediately read to present a summary of the client's current state. The former is identified to trigger the state of notifications when data acquisition is about to begin.

Disconnection:

A disconnection occurs when the Gatt client is out of range or the user requests it from the app. Once disconnected, the app is able to immediately reconnect if the current activity in the foreground is DeviceControlActivity. If not, the user must go back to the Scan activity, scan and reconnect.

Client

BGLIB callback method:

For connection:

```
void my_ble_evt_connection_status(const ble_msg_connection_status_evt_t *msg)
{…}
```

Once a connection request has been made, this call back is triggered internally from BGLIB, with a constructor carrying a "msg" struct. This struct can then have its elements retrieved acted upon for details of the connection.

For disconnection:

```
my_ble_evt_connection_disconnect(const struct
ble_msg_connection_disconnected_evt_t *msg) {
…
}
```

Like in the connection above, the void function provides a msg struct that provides details on the disconnection event.

Enabling / Disabling Notifications (Server and Client)

Server

Enabling notifications:

This occurs when the user taps the “start” button within the app, or triggers the notification switch.

This causes the GATT_HANDLE_C_BIOIMPEDANCE_DATA characteristic identified during the connection event to have its client configuration written to it, after which the characteristic is written to signaling to the client to start collecting data.

Disabling notifications:

The same as above, but instead of writing the value for enabling notifications, it writes the value for disabling it.

Client

BGLIB callback method:

```
void my_ble_evt_attributes_status (const struct ble_msg_attributes_status_evt_t *msg) {
```

Again this callback provides a struct “msg” that provides details on the notification event. The element “flags” of the struct “msg” is the important detail here. If the value for flag is 1, notifications are enabled. If the value is 0, they are disabled.

Writing to Characteristics (Server and Client)

Server

There are only 2 characteristics the app is able to write to, they are:

1. GATT_HANDLE_C_SAMPLE_RATE
2. GATT_HANDLE_C_AC_FREQ

The first characteristic has a size of 1 byte, and carries the sample rate of the app in Hz. The user is limited by the user interface to only write values between 5 and 90 Hz with increments of 5 Hz in between.

The second characteristic has a size of 3 bytes, and carries the following:

- Start frequency of the frequency sweep
- Step size of the frequency sweep
- Number of increments in the frequency sweep

In the GUI of the app, the user is able to decide whether he / she wants to utilize frequency sweeps or not. If the user opts out of frequency sweeps, the start frequency will have the user-selected value written to it, while the remaining two will have the value of 0 written to both. This helps the client decide whether to perform the sweeps or not. If the user opts in to frequency sweeps, all 3 bytes will have user-selected values written to them.

Client

BGLIB callback method:

```
void my_ble_evt_attributes_value(const struct ble_msg_attributes_value_evt_t *msg)
{…}
```

This callback method also provides a struct “msg” that provides additional information about the write event.

An element of this struct is handle, which is used to identify which characteristic was written to. There are 2 scenarios:

If GATT_HANDLE_C_SAMPLE_RATE was written to, the client simply has the timer interrupt disabled, and is reset with the new value obtained from the BLE connection.

If GATT_HANDLE_AC_FREQ was written to, the client first needs to identify if frequency sweeps are enabled or not. To do so, it checks the value of the second byte in the characteristic: step size. If the value is 0, sweeps are disabled, so it simply resets the AD5933, and provides the single start frequency for the alternating current. Otherwise, it performs all necessary functions to perform a frequency sweep.