**Lab 10: Bluetooth**

**Instructor: Prof. Yifeng Zhu**

**Fall 2019**

## Goals

1. Gain an understanding of Bluetooth low energy.
2. Understand Bluetooth stack.
3. Write a function that sends a values via Bluetooth and read that value on a laptop.

## Pre-lab Assignment

1. Find out your STEVAL-FCU001V1 BLE address.
2. Read Bluetooth low energy basics
   * More in-depth explanation can be found in the BlueNRG programming manual
3. Complete Bluetooth worksheet

## Lab

1. Write a function that sends a value via Bluetooth to a laptop.

# Bluetooth Low Energy Basics

This board uses Bluetooth low energy (BLE). BLE transmits at 1Mbps and has a range of 2 to 5 meters, and the longer the range increases the battery consumption. BLE operates in the 2.4GHz – 2.48GHz range also known as the industrial, scientific, and medical (ISM) spectrum. BLE chips are also relatively cheap.

BLE Host

1. ***Generic Access Profile*** (GAP)

GAP is in control of advertising and connections. This layer specifies how devices perform control procedures such as device discovery, and connection.

* + Helps maintain a consistent and interoperable communication

1. ***Generic Attribute Profile*** (GATT)

GATT defines how data is organized and exchanged between different applications.

* + Defines the way that services, characteristics and descriptors are defined and used.
  + Built on top of the attribute protocol (ATT) and security manager (SM)

1. ***Attribute Protocol*** (ATT)
   * This allows the device to say what certain pieces of data are. Each piece of data has an attribute type saying what it is and is defined by a 16 bit universally unique identifier (UUID). An example of an attribute is shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute handle | Attribute type | Attribute Value | Attribute permissions |
| 0x008 | “Battery voltage UUID” | “Battery Voltage Value” | “Read only, No authorization, No authentication” |

1. ***Security Manager*** (SM)

Bluetooth low energy link layer supports encryption and authentication. Not really used on the drone board.

1. ***Logical Link Control and Adaptation Protocol*** (L2CAP)
   * It takes multiple protocols and encapsulates them into the standard BLE packet format.
   * It takes long bits of data and separates them into the 37-byte maximum payload size for BLE for transmit and vice versa for receive.
   * In charge of routing data through the ATT to get UUID’s and through SM to make sure the connection is secure.
2. ***Host Controller Interface*** (HCI)

Host side (microcontroller) Communicates via a serial interface SPI, UART, etc. To the HCI on the controller side (Bluetooth chip). Most of the host controller interface comes from Bluetooth low energy specifications. The user codes this, ST has made it easy by making a application control interface (ACI) that implements the HCI and the other host elements. So simple function calls through the ACI can update values and setup the different data packets.

BLE Controller

1. **Host Controller Interface** (HCI)

The controller side (BLE device), communicates with the micro controller. Sends data packets to the link layer (LL), which then transmits the data. If it has received data it pulls a line high signaling to the micro controller that it has a received a data packet and wants to transmit the data to the micro controller.

1. **Link Layer** (LL)

LL handles packets of data, and interfaces with the physical layer, and in charge of establishing connections.

* + Information is exchanged via packets. No streaming is available
  + Advertising packets are used for find and connect to other devices or to broadcast data. Channels 37, 38, and 39 are used for advertising.
  + Data packets are used once a connection has been established by master and slave. Channels 0-36 are used for data packets. The table below shows the Packet structure

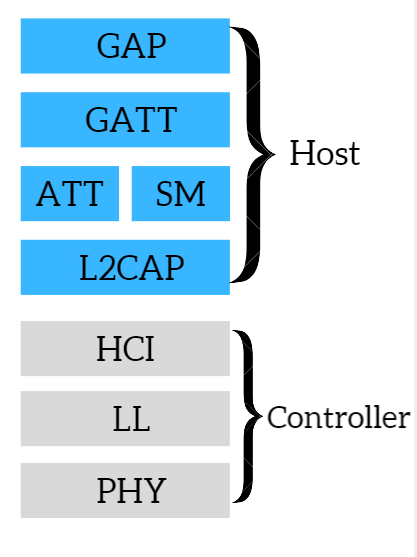
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bits | 8 | 32 | 8 | 8 | 0-296 (37 Bytes) | 24 |
| Contents | Preamble | Access Address | Header | Length | Data | CRC |

1. **Physical Layer** (PHY)

PHY contains the analog communications circuity used for modulating and demodulating analog signals and transforming them into digital symbols.

Bluetooth module SPBTLE-RF

The Bluetooth module used on the STEVAL-FCU001V1 board is a SPBTLE-RF, which contains a BlueNRG microcontroller. The BlueNRG programming manual provides more in-depth explanation of Bluetooth low energy.



1. Indicated the channels used for advertising packets with an ‘X’, and a ‘O’ for data packets in the Type row below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Type** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Channel** | **37** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **38** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **181** | **19** | **20** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **39** |
| **Frequency** | 2402MHz | 2404MHz | 2408MHz | 2410MHz | 2412MHz | 2414MHz | 2416MHz | 2418MHz | 2420MHz | 2422MHz | 2424MHz | 2426MHz | 2428MHz | 2430MHz | 2432MHz | 2434MHz | 2436MHz | 2438MHz | 2440MHz | 2442MHz | 2444MHz | 2446MHz | 2448MHz | 2450MHz | 2452MHz | 2454MHz | 2456MHz | 2458MHz | 2460MHz | 2462MHz | 2464MHz | 2466MHz | 2468MHz | 2470MHz | 2472MHz | 2474MHz | 2476MHz | 2478MHz | 2480MHz |

How data is received?

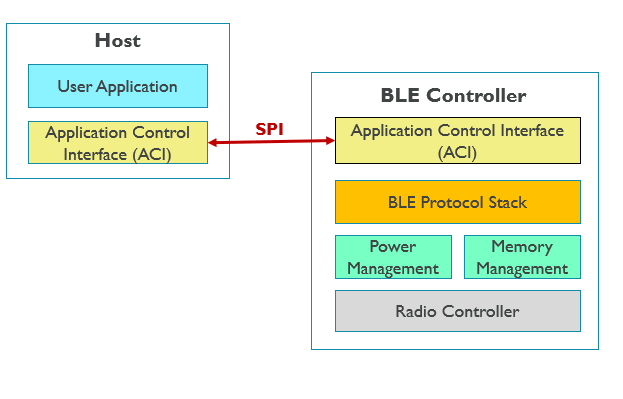
1. When the Bluetooth chip receives a packet and holds it in a buffer, an interrupt request is sent to the microcontroller.
2. The micro controller then receives the interrupt and runs the interrupt handler called HCI\_isr() in hci.c.
3. The data packet is then transferred via SPI, and hciReadPktPool is filled with the data and put the data packet into hciReadPktRxQueue.
4. The microcontroller periodically runs HCI\_process(), which processes any packet in the queue and places it back into hciReadPktPool.
5. The data is parsed in GETT\_Notification\_CB() in ble\_service.c, in which it gets parsed into the attribute format shown above.

# Pre-Lab

1. What is a universally unique identifier (UUID)?
2. What does the Attribute Protocol do and why is it important?
3. For the STEVAL-FCU001V1 board what type of serial communication is used for the host controller interface?

# Lab

1. For this lab you will be programing at the host level using the application control interface (ACI) which is the Bluetooth drivers that ST has developed. The ACI for the most part replaces the HCI while implementing GATT, ATT, SM, and L2CAP. These drivers make it very easy to setup and transmit data.
2. The main ACI functions used in the main.c to setup the device include:
   * **aci\_hal\_write\_config\_data** – sets up the public address
   * **aci\_gatt\_init** - initializes the GATT
   * **aci\_gap\_init\_IDB05A1** – initializes the GAP
   * **aci\_gap\_set\_auth\_requirement** – GAP sets up authentication where you can change the pin/password for the device
   * **aci\_hal\_set\_tx\_power\_level** - Set output power level
   * **Add\_ConfigW2ST\_Service/Add\_ConsoleW2ST\_Service/Add\_HWServW2ST\_Service –** Setup and initialize the different services name and UUID that data is going to be exchanged through
   * **aci\_gatt\_update\_char\_value** – used to update data sent via Bluetooth, all data is associated with a handle that tells it which service and UUID to send it with.
3. When implanting the ACI into the BLE stack, it gets simplified down to what is shown below.



1. More documentation on the ACI is available at <https://www.st.com/content/ccc/resource/technical/document/user_manual/6d/a1/5b/6c/dc/ab/48/76/DM00162667.pdf/files/DM00162667.pdf/jcr:content/translations/en.DM00162667.pdf>
2. A specific UUID and service has already been set up, and the UUID is shown in step 10. The goal of this lab is to transmit a value across that UUID and service. The UUID and service is referred setup under LabHandle.
3. In the sensor\_service.c file finish writing the ***Value\_Update()*** function that updates the LabServHandle, and LabHandle handle.
   * The LabServHandle , and LabHandle and Lab service initializing code has already been written for you(UUID/service setup).
   * Use the aci\_gatt\_update\_char\_value(), and STORE\_LE\_16() functions.
   * Update the LabHandle data be the value that is passed by reference (value1).
   * All data needs to be stored in an 8-bit buffer, in little endian format.
   * Call Value\_Update() and send it a value in the main.c while(1) loop.
   * Example code is shown below, and a few other examples can be seen in the sensor\_service.c file.

The following program shows how the BLE host uses the ACI functions to send the battery, environmental and RSSI values via the Bluetooth controller.

/\*\*

\* @brief Update Battery, Environmental and RSSI characteristic value

\* @param float Press Pressure in mbar

\* @param uint16\_t Batt Battery level as percetange of full battery

\* @param int16\_t Temp Temperature in tenths of degree second sensor

\* @param int16\_t RSSI level in dB

\* @retval tBleStatus Status

\*/

tBleStatus **Batt\_Env\_RSSI\_Update**(int32\_t Press, uint16\_t Batt, int16\_t Temp, int16\_t RSSI) {

tBleStatus ret;

uint8\_t BuffPos;

uint8\_t buff[2 + 4/\*Press\*/+ 2/\*Batt\*/+ 2/\*Temp\*/+ 2/\*RSSI\*/];

STORE\_LE\_16(buff, (**HAL\_GetTick**() >> 3));

BuffPos = 2;

STORE\_LE\_32(buff + BuffPos, Press);

BuffPos += 4;

STORE\_LE\_16(buff + BuffPos, Batt);

BuffPos += 2;

STORE\_LE\_16(buff + BuffPos, Temp);

BuffPos += 2;

STORE\_LE\_16(buff + BuffPos, RSSI);

BuffPos += 2;

ret = **aci\_gatt\_update\_char\_value**(HWServW2STHandle,

EnvironmentalCharHandle,

0,

2 + 4 + 2 + 2 + 2,

buff);

**if** (ret != BLE\_STATUS\_SUCCESS) {

**if** (W2ST\_CHECK\_CONNECTION(W2ST\_CONNECT\_STD\_ERR)) {

BytesToWrite = **sprintf**((**char** \*) BufferToWrite,

"Error Updating Environmental Char\r\n");

Stderr\_Update(BufferToWrite, BytesToWrite);

} **else** {

PRINTF("Error Updating Environmental Char\r\n");

}

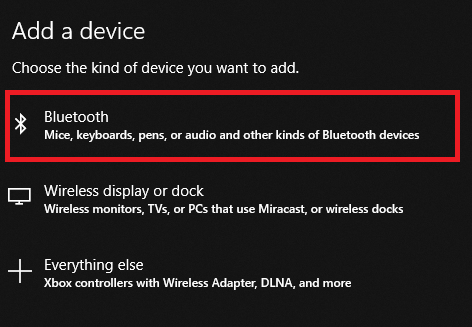
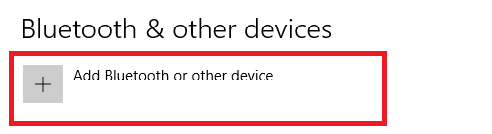
**return** BLE\_STATUS\_ERROR;

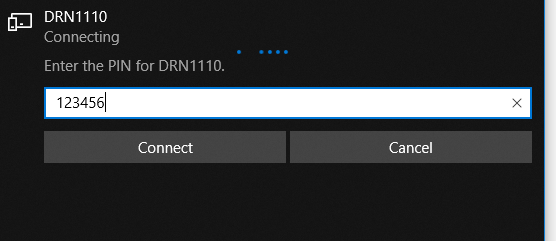
}

**return** BLE\_STATUS\_SUCCESS;

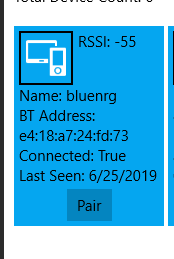
}

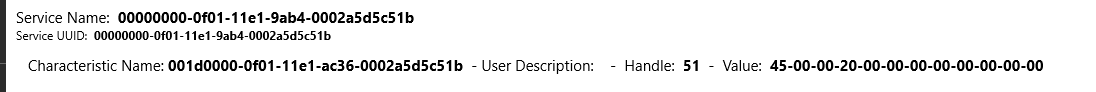
1. Download and install ***Bluetooth LE Explorer***
   * <https://www.microsoft.com/en-us/p/bluetooth-le-explorer/9n0ztkf1qd98?activetab=pivot:overviewtab>
2. Use a laptop with Bluetooth and in System settings connect to the drone board
   * The pin for the board is 123456.





1. Open Bluetooth LE Explorer and wait for the drone to show up and connect, then click on the device.



1. Scroll down and find this service name and UUID, the value is the hex value of what you sent it. For this example, the value being sent is 69. 

tBleStatus **Value\_Update**(uint16\_t value1)

{

tBleStatus ret;

**return** BLE\_STATUS\_SUCCESS;

}