ELEC5514 Lab2: Serial Commands to Board

In this lab, we will try to communicate with the development board through the PC serial port and send different instructions to the development board to execute. Note that so far, we have only tried some simple features using the serial port. Starting next week, we will really start using the BLE protocol to implement some simple transmitting and receiving functions. Today's lab will be divided into three parts:

1. Design the code, use the PC to send a message to the development board through the serial port, and ask the development board to reply to the same message.
2. Design the code, use the PC to send instructions to the development board through the serial port for specified reply from the development board.
3. (Optional) Add an empty service to the BLE device.

Section 3 is an **optional** part of the BLE communication experiment for next week, and this part will not be included in this week's lab report. If you have completed the first two sections, you can choose to familiarize yourself with this section in advance.

Section 1

UART is short for Universal asynchronous receiver/transmitter and it is the most widely used wireline data exchange protocol. Before using it, you need to define the right baud rate and set the correct port number. After connecting the device with the PC, you can find the "COM" number in "Ports" in "Device Manager", you can find it in "Control Panel".

In Section 5 of the last lab, you are asked to realize the communication between the board and the PC, for example, let the board send message to the PC via the serial port and read it via Putty on the screen. The process can be realized by the function：

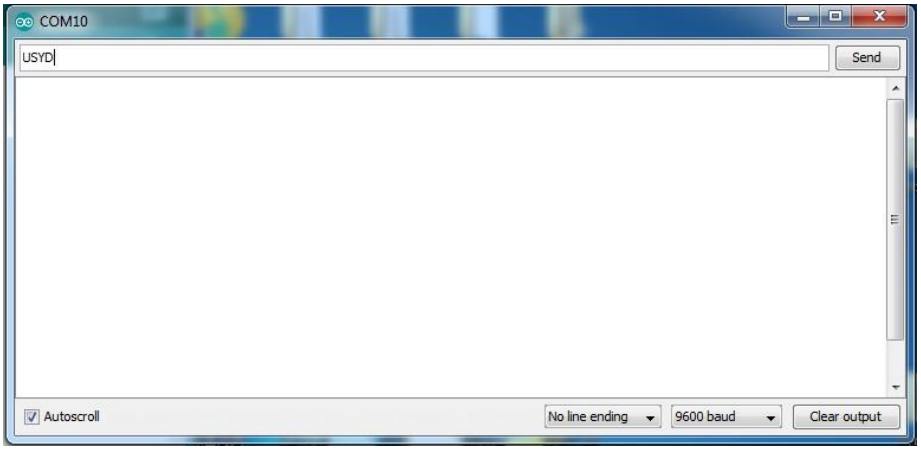
"*HAL_UART_Transmit()*"

Please use the code in "**Lab2-1_UART**"

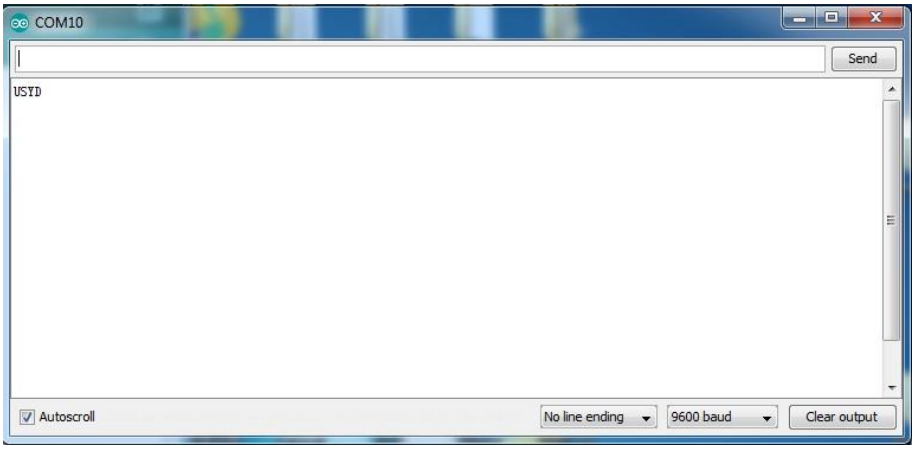
In this section, you need to realize the receiving function of the board. In order to complete this:

1. Open the *Source Code\Lab2-1_UART\MDK-ARM\ Lab1_UART_Tx*, and you can modify the "**main.c**" file.
2. You can find the functions related to the UART Rx/Tx in "{}Functions" / "**stm32l0xx_hal_uart.c**".
3. Design your own code for the following function: use your PC to send a message to the development board through the serial port and ask the development board to reply to the same message you sent.
4. After finishing coding, you can use "**Arduino**" to check what you are transferring from the PC to the board via the serial port you used (check "port" to select the port number you use). Then click "**serial monitor**" and **input your group number** (like 'Group 1') in the top window and click "Send". The configuration of the baud rate is at the bottom.

5. You can find that your message will be sent to the board and the PC can get feedback from the board, showing the message you send. Following is an example sending 'USYD':



Remember, the baud rate is the same as your code in the main.c, which should be 115200.



Then you can receive the message from the board, and the message is exactly what you sent in the last step. Once the screen can receive and display what you typed, the result may be not perfect. Please try some methods to perfect it.

When you finish Section 1, call your tutor to check you result. You also need to record the results as the note mentions, two figures and some explanations about your understanding of this process. Also provide the code you edited.

Section 2

In Section 2, we will further upgrade the code we designed in Section 1.

Continue to use the code you designed in the previous section to add the following features:

1. When '1' is sent to the board through the serial port, the board should reply the name of one of your groupmates, like 'Alex'.
2. When '2' is sent to the board through the serial port, the board should reply the name of another groupmates, like 'Bob'.
3. When '3' is sent to the board, reply the name of your last groupmate if you have.
4. For other inputs, the board should still return the original message like we did in Section1.

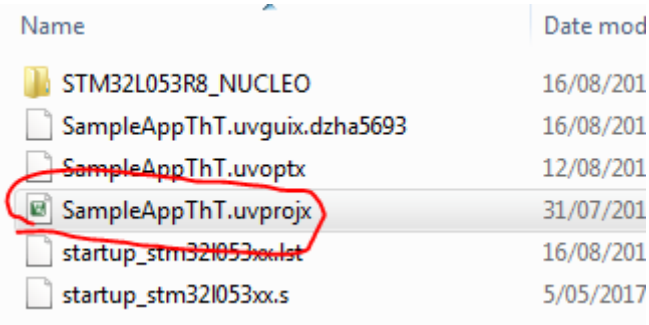
When you finish Section 2, call your tutor to check you result. You need to take a screenshot for use in the report.

Section 3 (Optional)

Section 3 is an **optional** part of the BLE communication experiment for next week, and this part will not be included in this week's lab report. If you have completed the first two sections, you can choose to familiarize yourself with this section in advance.

Please use the code in "**Lab2-2_SampleApp**":

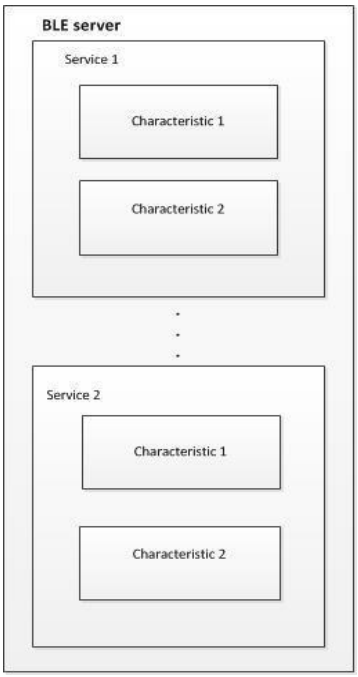
Source Code/ Lab2-2_SampleApp\Projects\Multi\Applications\SampleAppThT\MDK-ARM\STM32L053R8-Nucleo



In this section, you are beginning to touch the BLE world. Here we will use a specific example to explain how BLE works. To achieve the data exchange, a BLE device can be configured as either the client or the server.

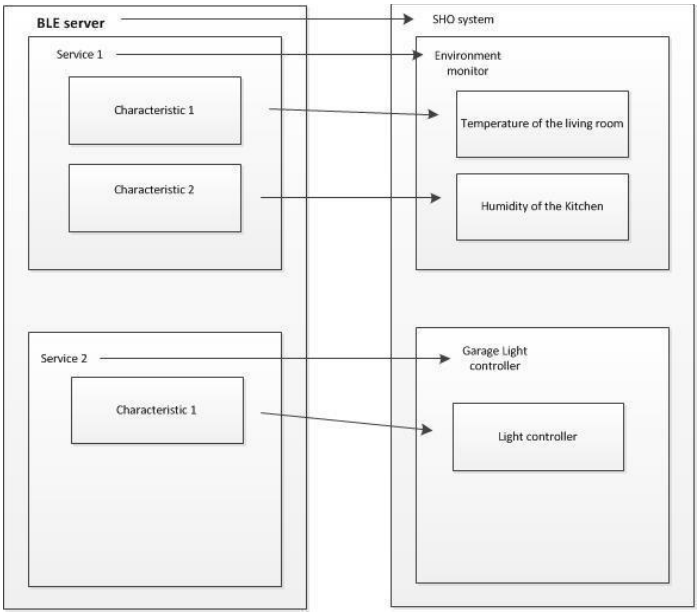
The client plays a role of sending the data request to the server.

The server plays a role of responding the data request from the client, e.g., data provider. Each server may have more than one services, each service can have at least one characteristic. The service is a predefined application. The characteristic can be regarded as the sub-function within a service. The exchanged data between server and client is called the attribute in BLE protocol. Based on the brief explanation above, a BLE service structure can be described as the graph below：



To make it easy to understand, we will use an example to explain.

Tom wants to set a SHO (Smart Home) system in his house. He prefers to add the environment monitor (service 1) and the light controller in garage (service 2) to the SHO system. For the environment monitoring, he wants the system to report the temperature state in the living room (characteristic 1 in service 1) and the humidity of the kitchen (characteristic 2 in service 1). After Tom left his house in the morning, he will get the corresponding temperature (attribute from characteristic 1) and the humidity (attribute from characteristic 2) every 5 hours. When Tom comes back home, he wants to use his phone to switch on (characteristic 1 in service 2) or off (characteristic 2 in service 2) the light in his garage. When Tom sends the number "1" (attribute of characteristic 1, service 2), the light will be turned on. In contrast, "2" (attribute of characteristic 2, service 2) indicates the light turned off.



The introduction for the "service", "characteristic" and "attribute" in detail can be found from "BLE_stacks programming guidelines.pdf" in the folder of document in Lab 1 content.

To avoid the connection confliction among groups, we will allocate each group a specific address.

Please follow the form below (We strongly recommend that each group uses the corresponding local name, device name and mac address in the following labs):

Group N0.	Local name	Device name	MAC address
1	BLE_LOC_G1	Dev1	0xF1, 0x33, 0x01, 0xE1, 0x80, 0x03
2	BLE_LOC_G2	Dev2	0xF2, 0x33, 0x01, 0xE1, 0x80, 0x03
3	BLE_LOC_G3	Dev3	0xF3, 0x33, 0x01, 0xE1, 0x80, 0x03
4	BLE_LOC_G4	Dev4	0xF4, 0x33, 0x01, 0xE1, 0x80, 0x03
5	BLE_LOC_G5	Dev5	0xF5, 0x33, 0x01, 0xE1, 0x80, 0x03
6	BLE_LOC_G6	Dev6	0xF6, 0x33, 0x01, 0xE1, 0x80, 0x03

7	BLE_LOC_G7	Dev7	0xF7, 0x33, 0x01, 0xE1, 0x80, 0x03
8	BLE_LOC_G8	Dev8	0xF8, 0x33, 0x01, 0xE1, 0x80, 0x03
9	BLE_LOC_G9	Dev9	0xF9, 0x33, 0x01, 0xE1, 0x80, 0x03
10	BLE_LOC_G10	Dev10	0x10, 0x33, 0x01, 0xE1, 0x80, 0x03
11	BLE_LOC_G11	Dev11	0x11, 0x33, 0x01, 0xE1, 0x80, 0x03
12	BLE_LOC_G12	Dev12	0x12, 0x33, 0x01, 0xE1, 0x80, 0x03
13	BLE_LOC_G13	Dev13	0x13, 0x33, 0x01, 0xE1, 0x80, 0x03

After getting familiar with the principle of service, characteristic and attribute, in this section, you are required to add

1) A blank service, the service name can be determined by yourself but please use the UUID:

0x1b,0xc5,0xd5,0xa5,0x02,0x00,0xb4,0x9a,0xe1,0x11,0x3a,0xcf,0x80,0x6e,0x36,0x5d

2) The service must include one blank characteristic, the characteristic name can be determined by yourself but please use the UUID

0x1b,0xc5,0xd5,0xa5,0x02,0x00,0xb4,0x9a,0xe1,0x11,0x3a,0xcf,0x80,0x6e,0x36,0x5e

Hints: In the files “sample_service.c” and “sample_service.h”, the source code has defined an example service and the corresponding characteristics, like “sampleServHandle” and “TXCharHandle”, you can use the “Ctrl” + “F” to find the corresponding operations, like how to define the service and characteristic. Please note that after you finish defining your own service in “sample_service.c”, you need to relate this “adding” action to the “sample_service.h”, and then, in the “main.c” file, you need to execute it so that the application is added online.

1. In the “sample_service.c” file, you need to customize your own handle or variables, and then define your own service and add your own service.
   In order to figure out some functions, you can select them and click the right button and click “Go to definition of xxx” to find the definition of this code. For example, in the adding service process, you may use the code “aci_gatt_add_serv”.
2. In the “sample_service.h” file, you also need to register the adding function.
3. In the “main.c” file, you need to execute the function of adding your service.

(Please note that the line number may be not exact right as the different local PC setting environment. The corresponding content should be found around the proposed line number.)

Our first lab report will be leased today, please find the requirement on Canvas.