2016년 2학기
컴퓨터하드웨어 실험 보고서
7조 8주차

한성희 이호욱 한경수

Table Of Contents

개요

UART 를 통한 개발 보드와 스마트폰간의 블루투스 통신

기본 개념 (배경 지식)

블루투스

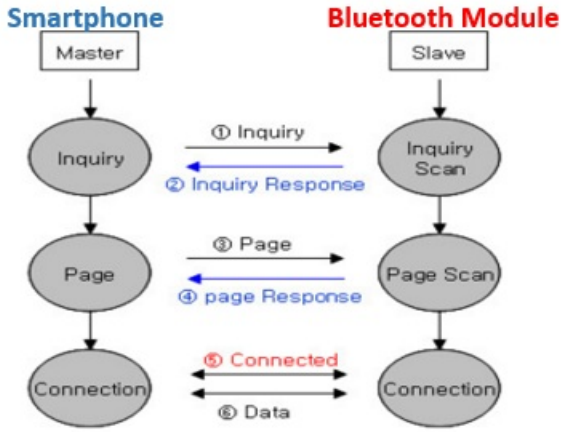
블루투스는 휴대폰 - 휴대폰 또는 휴대폰 - PC 간에 사진이나 벨소리 등의 파일을 전송하는 무선전송기술이다.

블루투스 통신 개요와 Master-Slave 구조

블루투스 통신 개요

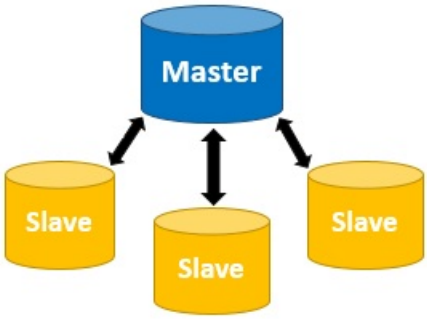
블루투스의 무선 시스템은 ISM 주파수 대역인 2400~2483.5MHz를 사용한다. 이 주파수 대역은 산업, 의료, 과학용으로 할당된 주파수로 따로 전파사용에 대한 허가를 받을 필요가 없어서 개인 무선 통신에 쓰인다. 데이터 전송 거리는 10m 정도로 최대 100m 까지 가능하다. 좁은 대역대에 많은 기기가 몰리다보니 충돌이나 간섭 가능성이 생기게 됐는데, 이를 피하기 위해 주파수 호핑기법(FHSS : Frequency Hop Spread Spectrum)을 사용하여 간섭과 페이딩을 방지한다. 주파수 호핑기법은 짧은 시간에 주파수를 이동하면서 패킷을 잘게잘게 보내는 방식이다. 이 호핑 방식을 매칭하는게 일반적인 우리가 알고 있는 블루투스 연결이라는 개념이다. 블루투스는 Master-Slave 형태로 구성되며, 한 개의 블루투스 장치에 7개가 동시 접속 가능하다.

블루투스 통신 구조




```
1. 블루투스는 기본적으로 Master와 Slave인 주종인 역할(ROLE)로 동작한다.

2. 통상적으로 Inquiry(검색) 및 Page(연결요청)을 하는 쪽을 Master라고 하며, Inquiry Scan(검색 대기) 및 Page Scan(연결 대기)를 하는 쪽을 Slave 라고 한다.

3. Master가 주변의 Slave를 찾으면(Inquiry), Slave는 자신의 정보를 Master에게 송신(Inquiry Response)한다.

4. Slave의 정보가 Master와 일치하면 상호 연결이 이루어 지며, 데이터 전송이 가능하게 한다.
```

이번 실험에서 Smartphone이 주변 bluetooth가능한 기기를 찾게 되니깐, Master는 Smartphone이 되고 slave는 Bluetooth 모듈이 된다.



< Master-Slave 구조 >

블루투스 연결을 위해서는 Master 가 생성하는 주파수호핑에 Slave 가 동기화시켜야 한다. master 와 slave 는 상황에 따라 역할을 바꿀 수 있고 slave 끼리는 통신이 불가능하다.

블루투스 프로파일과 SPP

블루투스 프로파일

프로파일이란 어떤 종류의 데이터를 보내는지 명확하게 정의하기 위한, 블루투스의 프로토콜이다. 프로파일의 종류로는 HSP, HID, SPP, A2DP 와 같은 여러가지가 있는데, 우리가 다룰 프로파일은 SPP 이다.

SPP

SPP(Serial Port Profile)은 RS-232 통신을 무선으로 대체시키려고 할 때 쓴다. 시리얼 통신을 이용하여 데이터를 송/수신하는 프로파일로 SPP 를 이용하면 두 장치가 RX, TX 가 유선으로 연결된 것 처럼 동작한다.

블루투스 SSID와 UUID

SSID (Service Set Identifier)

SSID 는 Wi-Fi 네트워크 이름으로, 주변 장치에서 무선 라우터를 식별할 수 있도록 무선 라우터가 broadcast 하는 이름입니다.

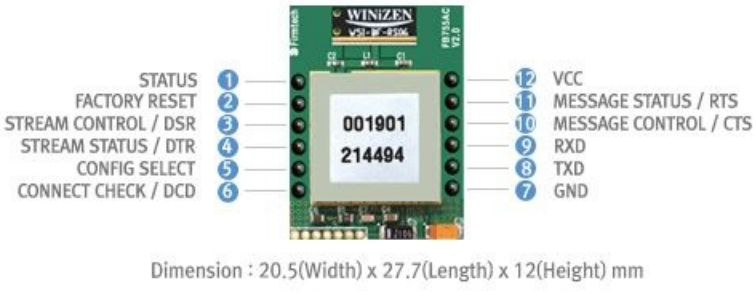
UUID (Universally Unique Identifier)

블루투스에는 Wi-Fi 의 대응되는 개념으로 UUID 가 있다.

블루투스의 UUID 란, 범용 고유 번호라고 불리며 128비트의 숫자들을 조합한다. 말 그대로 범용적으로 사용할 수 있는 고유의 ID를 사용하기 위하여 생성되며, 그렇기 때문에 128bits 의 Hex 조합은 Unique 하여야 한다. Bluetooth 에서는 device 에서 제공하는 service 를 검색하여 각 service 마다 UUID 를 부여하는 등 많은 부분에서 사용된다.

FB755AC

FB755AC 모듈



Dimension : 20.5(Width) x 27.7(Length) x 12(Height) mm

- Firmtech 에서 나온 제품으로 최대 7대 까지 slave 를 잡을 수 있으며, 100m 까지 인식
- 12 Pin Header Type 으로 되어있어 제품에 쉽게 적용 가능
- AT 명령어를 지원하며 AT 명령어를 이용하여 FB755AD 제어 가능
- Smartphone, Notebook, PDA 등의 스마트 장치들과 연결 및 통신 가능성이

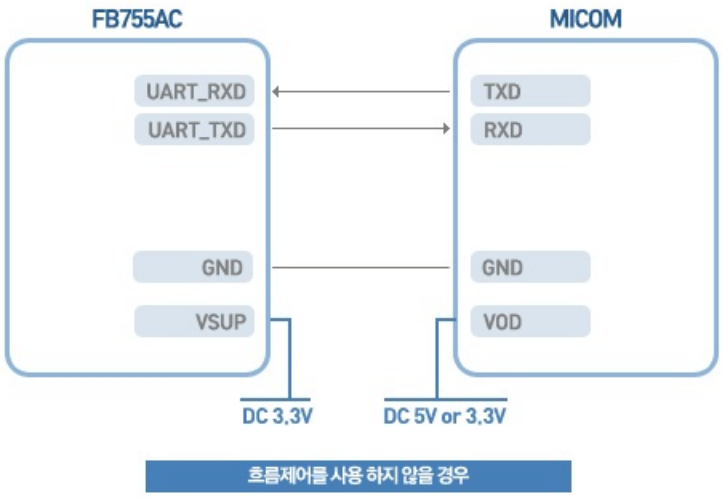
각 PIN 과 기능

핀번호	신호선	설 명	신호레벨	입/출력방향
1	STATUS	STATUS LED	TTL	출력
2	FA SET	FACTORY RESET	TTL	입력
3	STREAM CONTROL	1:N – Stream Control	TTL	입력
	DSR	1:1 – Data Set Ready		
4	STREAM STATUS	1:N – Stream Status	TTL	출력
	DTR	1:1 – Data Terminal Ready		
5	CONFIG SELECT	PC Configuration Select	TTL	입력
6	CONNECT CHECK	1:N – Connect Check	TTL	출력
	DCD	1:1 – Data Carrier Detect		
7	GND	Ground		
8	TXD	Transfer Data	TTL	출력
9	RXD	Received Data	TTL	입력
10	MESSAGE CONTROL	1:N – Message Control	TTL	출력
	CTS	1:1 – Clear To Send		
11	MESSAGE STATUS	1:1 – Message Status	TTL	출력
	RTS	1:1 – Ready To Send		
12	VCC	3.3V DC		입력

STATUS PIN은 블루투스 모듈의 상태를 모니터링 하기 위해 사용된다. 블루투스 연결을 대기 하거나 연결 시도 및 주변의 블루투스 장치를 검색 할 때는 LOW, HIGH를 반복하게 된다.

CTS/RTS/DSR/DTR PIN은 흐름제어를 사용하지 않을 시에는 연결하지 않아도 동작에 영향을 주지 않는다.

CONFIG_SELECT PIN을 사용하여 블루투스 설정을 할 수 있다.



위 그림은 시스템 연결도로, DCD PIN의 경우 1:N 통신시 마스터와 슬레이브의 connect check를 하고 1:1 통신 시에는 UART data carrier detect로 사용한다.

VCC와 GND PIN을 각각 보드에 연결하고 RXD PIN과 TXD PIN은 stm32보드와 반대로 연결하여야 한다.

AT 명령어

헤이즈 마이크로컴퓨터 사가 개발한 헤이즈 스마트 모뎀과 그 호환 모뎀을 제어하기위해 사용되는 명령어로 현재에는 사실상 거의 모든 모뎀의 표준 명령어이다.

본래는의 명칭은 헤이즈 명령어인데, 통상 AT로 명령어가 시작되기 때문에 AT명령어라고 불린다.

AT란 준비라는 뜻을 가진 Attention의 앞글자인 AT에서 따온 말로 헤이즈 명령어는 AT+Command와 같이 매우 간단한 명령어 구조를 가지고 있다.

즉 AT뒤에 원하는 명령어를 적어주기만 하면 되는데 한 명령어를 한 줄에 다 적어도 계속 이어 붙여 적을 수 있으므로 편리하다.

이 외에 헤이즈 모뎀명령어에는 '+++'과 'A/'가 추가적으로 있는데 이는 독자적으로 사용되는 명령어이다. 이 두 개를 제외한 명령어들은 거의 명령어 앞에 AT를 적어 사용한다.

또한 AT외에 at라고 사용하여도 되나 aT나 At와 같이 쓰면 인식하지 못하므로 이는 피해야하며 명령어는 한 줄에 40자까지 적을 수 있다.

3.14 AT+BTSCAN

FEATURE	BT의 검색대기(inquiry scan)와 연결대기(page scan)를 하도록 합니다.
APPLY PRODUCTS	FB100, FB200, FB755, FB155, FB155 HID, FB570
RESPONSE	∠OK∠ ∠CONNECT 123456789012∠ (다른 장치와 연결이 된 경우의 응답)
DESCRIPTION	블루투스 장치는 장치검색 작업을 할 때 검색대기를 하고 있는 블루투스 장치만 검색 할 수 있으며, 또한 연결대기를 하고 있는 장치로만 연결이 가능합니다. 이 명령을 사용하면 다른 블루투스 장치들이 BT 를 검색할 수 있고, 항상 연결도 가능합니다. 블루투스 장치로부터 연결이 이루어지면, "CONNECT 123456789012" 메시지가 표시되며, 연결이 종료되면 다시 검색대기와 연결대기를 수행합니다. 따라서 명령어 대기상태로 전환하려면 AT+BTCANCEL 명령을 사용하여야 합니다.

이번 실험에서는 이 AT명령어로 Bluetooth 모듈이 모바일에서 검색 및 연결을 대기할 수 있도록 만들어 주기위해 사용되며 AT+BTSCAN의 명령어를 사용한다.

실험 방법

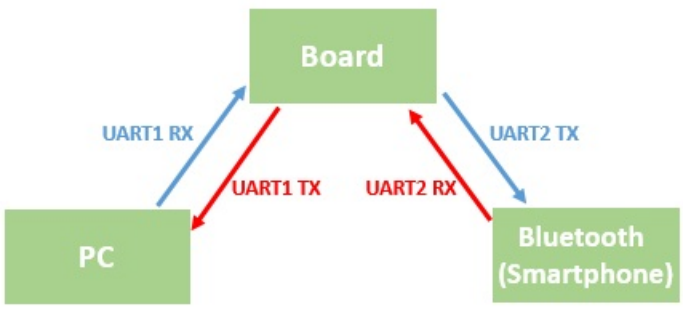
Android Studio

```
public class BluetoothChatService {
    //Debugging
    private static final String TAG = "BluetoothChatService";
    private static final boolean 0 = true;

    //Name for the SDP record when creating server socket
    private static final String NAME = "BluetoothChat";

    //Unique UUID for this application
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    //Member fields
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;
    private AcceptThread mAccepThread;
    private ConnectThread mConnectThread;
    private ConnectedThread mConnectedThread;
    private int mState;
```

먼저 스마트폰에서 Bluetooth chatting App 을 올리기 위해서 Sample Project 에서 BluetoothChat 을 선택하여 적절한 UUID 값으로 변경한다.

Sample Project 를 android studio 에서 import 한 뒤, 자신이 쓰는 스마트폰에 올린다.

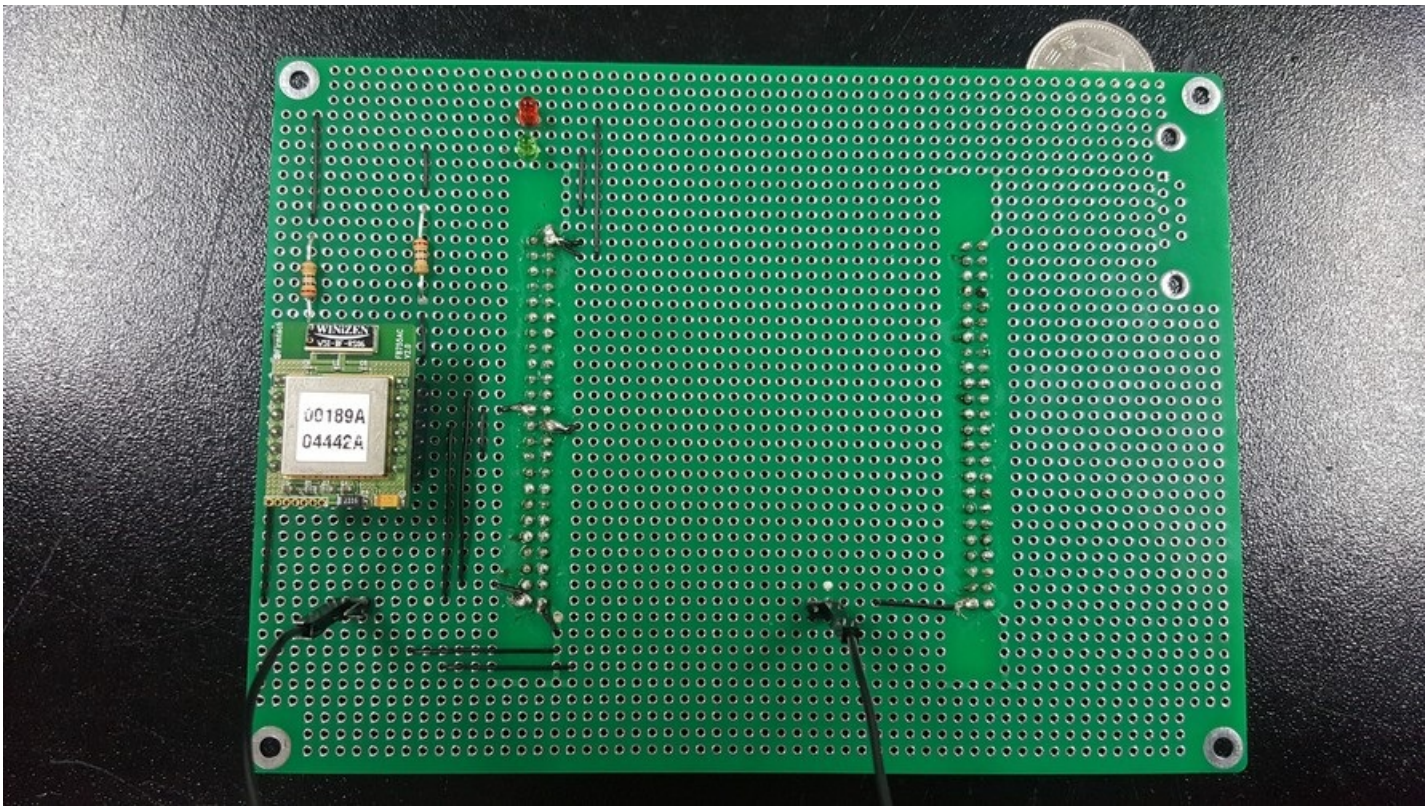
UART 세팅



PC 와 Board 는 UART 1, Board 와 Bluetooth(Smartphone) 은 UART 2 를 이용하여 연결한다.

PC 가 보낸 UART1 RX 를 BOARD 받아 UART2 TX 를 Bluetooth 에게 보낸다. 그리고 Bluetooth 에서 보낸 UART2 RX 를 BOARD 가 받아서 UART TX 를 PC 에게 보낸다.

개발보드와 FB755AC 연결(납땜) 방법



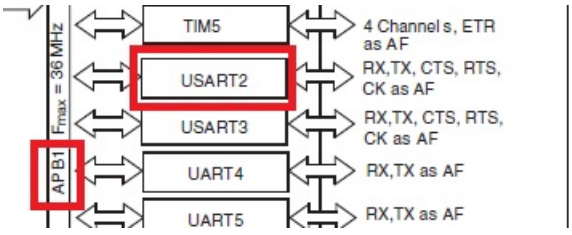

```
- 1, 6 번 : LED 와 연결
- 2, 3, 4, 10, 11 번 : 사용 X
 5 번 : 직접적으로 연결하지 않고 점프 선을 연결하여 값이 low/high 로 변경 가능 하도록 만듬
- 5, 12 번 : VCC(3.3V) 와 연결
- 8 번 : RX 인 PA3 과 연결
- 9 번 : TX 인 PA2 와 연결
- LED 의 (+)와 저항을 연결
```

USART 1 은 baud rate를 57600, USART 2 는 baud rate 를 9600 으로 설정한다.

5 번 port 가 high 가 되면 설정 모드, low 가 되면 설정 모드 해제를 뜻한다.

납땜 시, 선끼리 교차되지 않게 납땜한다.

실습 과정



datasheet 를 참고하면, USART 2 는 APB1 에 속해있다.

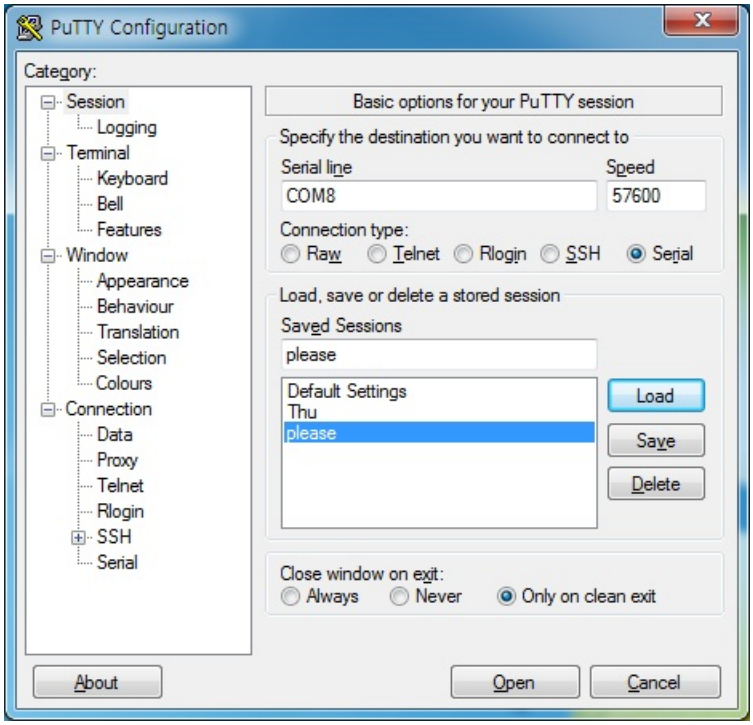
J2	16	25	PA2	I/O	-	PA2	USART2_TX[7]/ TIM5_CH3/ADC12_IN2/ TIM2_CH3 [7]/ ETH_MII_MDIO/ ETH_RMII_MDIO	-
K2	17	26	PA3	I/O	-	PA3	USART2_RX[7]/ TIM5_CH4/ADC12_IN3 / TIM2_CH4[7]/ ETH_MII_COL	-

USART 2 의 TX, RX 의 PIN 은 각각 PA2, PA3 이다.

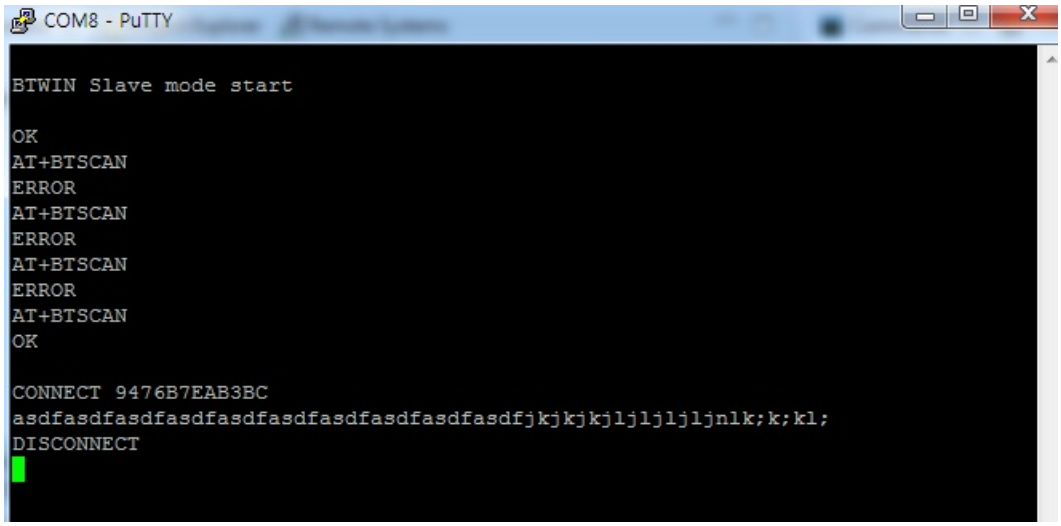
이를 참고하여 코드를 USART 2 에 대한 코드를 짜면 다음과 같다.

```
//USAR2 init

void USART2_Init(void)
{
  USART_InitTypeDef usart2_init_struct;
  GPIO_InitTypeDef gpioa_init_struct;
  NVIC_InitTypeDef NVIC_InitStructure;

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

  // tx, rx 설정
  gpioa_init_struct.GPIO_Pin = GPIO_Pin_2;
  gpioa_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
  gpioa_init_struct.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &gpioa_init_struct);

  gpioa_init_struct.GPIO_Pin = GPIO_Pin_3;
  gpioa_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
  gpioa_init_struct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &gpioa_init_struct);

  usart2_init_struct.USART_BaudRate = 9600;
  usart2_init_struct.USART_WordLength = USART_WordLength_8b;
  usart2_init_struct.USART_StopBits = USART_StopBits_1;
  usart2_init_struct.USART_Parity = USART_Parity_No;
  usart2_init_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  usart2_init_struct.USART_HardwareFlowControl = USART_HardwareFlowControl_None
;
  USART_Init(USART2, &usart2_init_struct);
  USART_Cmd(USART2, ENABLE);
  USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);

  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x01;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}


//USART 2 Handler
void USART2_IRQHandler(void) {
        unsigned char d;
        while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET)
                ;

        d = (unsigned char) USART_ReceiveData(USART2);
        USART_SendData(USART1, d);
        USART_ClearITPendingBit(USART2, USART_IT_RXNE);
}
```

지난 실험의 interrupt 코드를 활용하여 UART 2 에 대한 설정, NVIC 레지스터 설정, RX, TX PIN 설정을 한다.



코드 작성 후, Putty 를 다음과 같이 설정한 뒤, 보드 전원 인가 시, 터미널(Putty) 에 Configuration 하는 화면이 뜬다.



Configuration 화면을 확인한 뒤, AT 명령어를 이용하여 제대로 통신이 되는지 확인한다.

그 후, Configuration 모드를 해제하여 chatting APP 을 실행하여 자신의 Bluetooth 모듈이 검색되면 연결한 후 채팅 가능한지 확인한다.

결론

소스 코드는 전 실험과 같이 USART를 사용했기 때문에 그리 어렵지 않았지만 만능 기판에 블루투스 모듈을 납땜하는 과정에서 오랜 시간이 소요되었다.

블루투스 모듈의 각 핀의 동작을 이해할 수 있었고 USART를 사용하여 새로운 모듈을 보드에 추가하는 방법을 알 수 있는 실험이었다.

이번 실험을 통해 텀 프로젝트에서 사용할 모듈을 어떻게 보드에 연결하고 동작을 제어할 것인지 생각해 볼 수 있었다.

전체 소스

```
#include "stm32f10x.h"

void Delay(int d){
  int i=0;
  for(i=0; i<1000000; i++){
    ;
  }
}
```

```
9
10 void send_com(char buf[]) {
11         char *s = buf;
12         while (*s) {
13                 while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET)
14                         ;
15                 USART_SendData(USART1, *s++);
16         }
17 }
18
19 void send_phone(char buf[]) {
20         char *s = buf;
21         while (*s) {
22                 while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET)
23                         ;
24                 USART_SendData(USART2, *s++);
25         }
26 }
27
28 void USART1_Init(void);
29 void USART2_Init(void);
30 void GPIOD_Init(void);
31 void led_toggle(void);
32 void EXTI11_Config(void);
33
34 //GPIOD_INIT -> USART_init -> EXTI11_Config
35 // USART1_IRQHandler 는 컴퓨터로부터 받는 인터럽트 아스키 보냈을 대 횟수만큼 led
36 반짝(USART1_INTTERUPT )
37 // EXTI15_10_IRQHandler 는 버튼 인터럽트.
38
39 int main(void)
40 {
41   GPIOD_Init();
42         EXTI11_Config();
43         SystemInit();
44         USART1_Init();
45         USART2_Init();
46
47   while(1)
48   {
49   }
50 }
51
52 void USART1_Init(void)
53 {
54   USART_InitTypeDef usart1_init_struct;
55   GPIO_InitTypeDef gpioa_init_struct;
56   NVIC_InitTypeDef NVIC_InitStructure;
57
58   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_AFIO |
59                  RCC_APB2Periph_GPIOA, ENABLE);
60
61   gpioa_init_struct.GPIO_Pin = GPIO_Pin_9;
62   gpioa_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
63   gpioa_init_struct.GPIO_Mode = GPIO_Mode_AF_PP;
64   GPIO_Init(GPIOA, &gpioa_init_struct);
65   gpioa_init_struct.GPIO_Pin = GPIO_Pin_10;
66   gpioa_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
67   gpioa_init_struct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
68   GPIO_Init(GPIOA, &gpioa_init_struct);
69
70   usart1_init_struct.USART_BaudRate = 57600;
71   usart1_init_struct.USART_WordLength = USART_WordLength_8b;
72   usart1_init_struct.USART_StopBits = USART_StopBits_1;
73   usart1_init_struct.USART_Parity = USART_Parity_No;
74   usart1_init_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
75   usart1_init_struct.USART_HardwareFlowControl = USART_HardwareFlowControl_Non
76 e;
77   USART_Init(USART1, &usart1_init_struct);
78   USART_Cmd(USART1, ENABLE);
79   USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
80
81   NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
82   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;
83   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x01;
84   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
85   NVIC_Init(&NVIC_InitStructure);
86 }
87
88 void USART2_Init(void)
89 {
90   USART_InitTypeDef usart2_init_struct;
91   GPIO_InitTypeDef gpioa_init_struct;
92   NVIC_InitTypeDef NVIC_InitStructure;
93
94   RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
95
```

```
  // tx, rx 설정
  gpioa_init_struct.GPIO_Pin = GPIO_Pin_2;
  gpioa_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
  gpioa_init_struct.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &gpioa_init_struct);

  gpioa_init_struct.GPIO_Pin = GPIO_Pin_3;
  gpioa_init_struct.GPIO_Speed = GPIO_Speed_50MHz;
  gpioa_init_struct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &gpioa_init_struct);

  usart2_init_struct.USART_BaudRate = 9600;
  usart2_init_struct.USART_WordLength = USART_WordLength_8b;
  usart2_init_struct.USART_StopBits = USART_StopBits_1;
  usart2_init_struct.USART_Parity = USART_Parity_No;
  usart2_init_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  usart2_init_struct.USART_HardwareFlowControl = USART_HardwareFlowControl_Non
e;
  USART_Init(USART2, &usart2_init_struct);
  USART_Cmd(USART2, ENABLE);
  USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);

  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x01;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);




}

void USART1_IRQHandler(void) {
        unsigned char d;
        while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET)
                ;

        d = (unsigned char) USART_ReceiveData(USART1);
        USART_SendData(USART2, d);
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);

//        unsigned char buf[2];
//        int i = 0;
//        buf[1] = 0;
//
//        if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
//                USART_ClearITPendingBit(USART1, USART_IT_RXNE);
//                buf[0] = (unsigned char) USART_ReceiveData(USART1);
//                send_phone(buf);
//        }
}

void USART2_IRQHandler(void) {
        unsigned char d;
        while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET)
                ;

        d = (unsigned char) USART_ReceiveData(USART2);
        USART_SendData(USART1, d);
        USART_ClearITPendingBit(USART2, USART_IT_RXNE);

//        unsigned char buf[2];
//        int i = 0;
//        buf[1] = 0;
//
//        if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET) {
//                USART_ClearITPendingBit(USART2, USART_IT_RXNE);
//                buf[0] = (unsigned char) USART_ReceiveData(USART2);
//                send_com(buf);
//        }
}

void GPIOD_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOD, &GPIO_InitStructure);

}

void EXTI11_Config(void)
{
```

```
        GPIO_InitTypeDef GPIO_InitStructure;
        EXTI_InitTypeDef EXTI_InitStructure;
        NVIC_InitTypeDef NVIC_InitStructure;
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);


  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

  GPIO_EXTILineConfig(GPIO_PortSourceGPIOD, GPIO_PinSource11);


  EXTI_InitStructure.EXTI_Line = EXTI_Line11;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
  EXTI_Init(&EXTI_InitStructure);

  NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x01;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}

//void led_toggle(void)
//{
//  GPIO_SetBits(GPIOD, GPIO_Pin_2);
//  Delay(1000);zzzzz
//  GPIO_ResetBits(GPIOD, GPIO_Pin_2);
//  Delay(1000);
//}

void EXTI15_10_IRQHandler(void) {
        unsigned char buf[] = "Wed_team07";
        if (EXTI_GetITStatus(EXTI_Line11) != RESET) {
                send_com(buf);
        }
        EXTI_ClearITPendingBit(EXTI_Line11);
}
```