임베디드 시스템 설계 및 실험

8주차

7조
201424470 서민영
201424421 김시은
201424533 정종진
201424532 정재광

목차

1. 실험목표

1) Bluetooth 납땜 및 동작

2. 배경지식

1) Bluetooth 특징

- 컴퓨터, PDA, TV, 휴대전화 등 각종 기기들간 데이터를 무선으로 송수신할 수 있는 기술
- 목표 : 단거리, 저전력, 고 신뢰성, 저가의 무선통신 구현
- 사용 주파수 : 2,400 ~ 2,835 GHz, 79 Channel
- 전송 속도: 1Mbps ~ 3Mbps
- 네트워크 구성 : Master - Slave 형태의 주종 관계로 구성되며, 한 대의 Bluetooth 장치에 동시 접속이 가능한 최대 장치의 수는 7대이다.

2) Master & Slave

- Master : 검색 및 연결 요청을 하는 부분
- Slave : 검색 대기 및 연결 대기를 하는 부분
- Master가 주변의 Slave를 찾으면 Slave는 자신의 정보를 Master에게 송신하고, Slave의 정보가 Master와 일치하면 상호 연결이 이루어지며, 데이터 전송이 가능하게 된다.

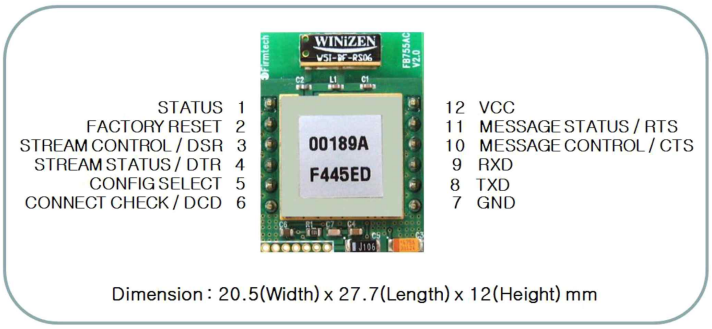
3) 프로파일 & SPP

- 프로파일 : 연결되었을 때 장치가 어떻게 동작하는지를 결정
- SSP(Serial Port Profile) : 두 장치가 많은 양의 데이터를 교환하는데 초점이 맞춰져있다. SPP를 사용하면 두 장치는 RX, TX라인이 유선으로 연결된 것처럼 데이터를 주고 받을 수 있다.

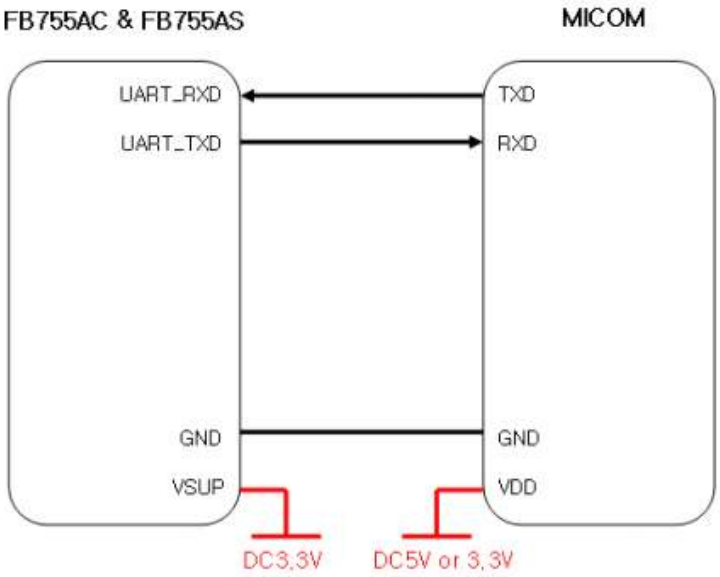
4) SSID & UUID

- SSID(Service Set Identifier) : WLAN을 통해 전송되는 모든 패킷에 존재하는 고유 식별자
- UUID(Universally Unique Identifier) : 전체 네트워크 상에서 서로 모르는 개체들을 식별하고 구별하기 위한 식별자

5) FB755AC Module



번호	핀 이름	기능	입/출 방향	신호레벨
1	STATUS	STATUS LED	출력	TTL
2	FA SET	Factory Reset Go back default setting	입력	TTL Pull-up
3	STREAM_CONTROL UART_DSR	1:N – Stream Control 1:1 – UART Data Set Ready	입력	TTL
4	STREAM_STATUS UART_DTR	1:N – Stream Status 1:1 – UART Data Terminal Ready	출력	TTL
5	CONFIG_SELECT	Configuration Select	입력	TTL Pull-down
6	CONNECT_CHECK UART_DCD	1:N – Connect Check 1:1 - UART Data Carrier Detect	출력	TTL
7	GND	Ground		
8	UART_TXD	UART Transfer Data Data output	출력	TTL
9	UART_RXD	UART Received Data Data Input	입력	TTL
10	MESSAGE_CONTROL UART_CTS	1:N – Message Control 1:1 - UART Clear To Send	입력	TTL
11	MESSAGE_STATUS UART_RTS	1:N – Message Status 1:1 - UART Ready To Send	출력	TTL
12	VSUP	3V3 for RF circuit (Vcc)	입력	



구 분	설 정 값
Device Name	FB755vx.x.x
Pin Code (Pass key)	BTWIN
Uart (baud rate-data bit-parity bit-stop bit)	9600-8-N-1
ROLE	SLAVE
Connection Mode	MODE4 (AT command)
Operation Mode	MODE0 (1:1 통신)
Debug Char	0x02

3. 실험과정

1) 보드 연결

<그림 1 : Coretex M3/JTAG/DSTREAM을 연결한 모습>

1-1) Coretex M3/JTAG/DSTREAM 연결

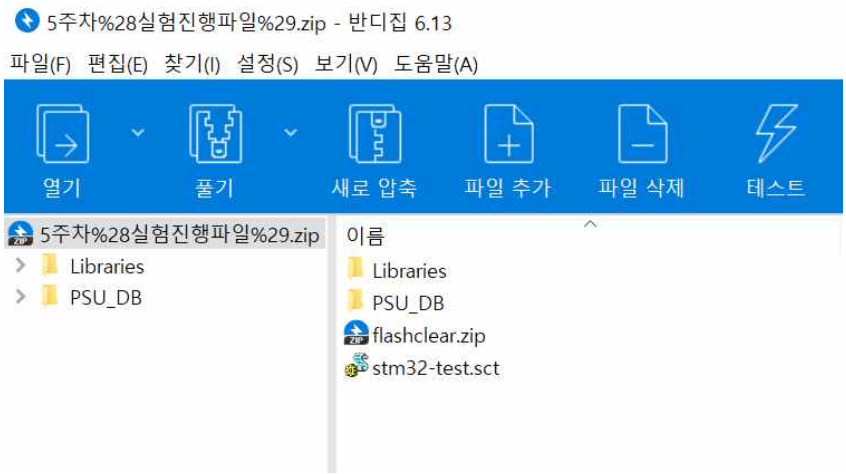
다음과 같은 순서로 보드를 연결한다. 이 때 연결 및 분리 순서를 제대로 지키지 않으면 장비가 망가질 수 있으니 주의해야한다.

2) DS-5 디바이스 데이터베이스 추가

2-1) 수업게시판에서 실습파일로 제공되는 PSU_DB, LIBRARIES, SCATTER FILE을 다운

① 보드와 DSTREAM JTAG 연결 ② 보드 전원선만 연결 (보드의 전원은 OFF 상태) ③ DSTREAM 전원 연결 및 ON ④ DSTREAM Status LED 점등 확인 ⑤ 보드 전원 ON ⑥ DSTREAM Target LED 점등 확인 ⑦ DS-5에서 'connect target'

<표 1 : 보드 연결 순서>



<그림 7 : 실습파일로 제공된 파일들>

2-3) Eclipse에 데이터베이스 추가

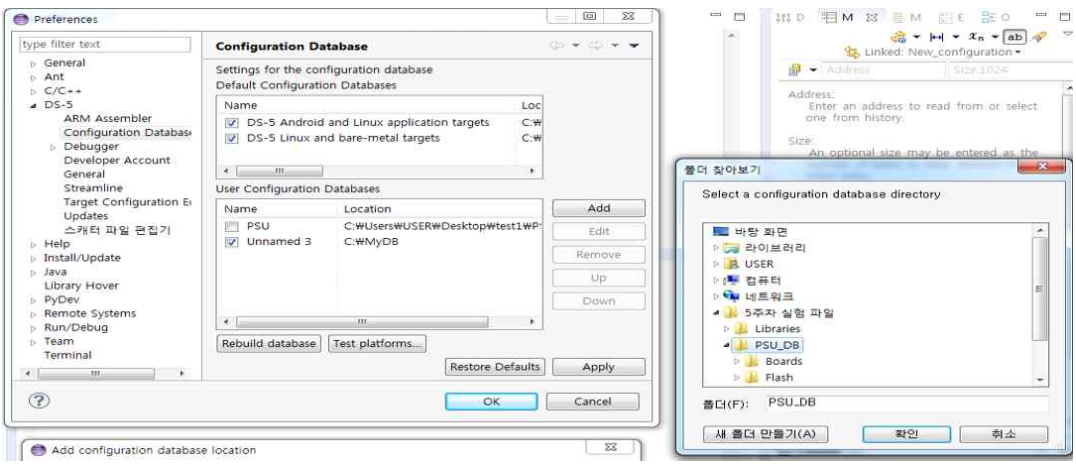
① 시작 → 모든 프로그램 → ARM DS-5 → Eclipse for DS-5 메뉴를 선택

② Windows → Preferences

③ DS-5 → Configuration Database 항목을 선택

④ Add 버튼을 클릭하여 사용자 데이터베이스의 디렉터리를 지정

⑤ Rebuild database 버튼을 클릭하여 데이터베이스 추가를 완료



<그림 8 : Eclipse에 데이터베이스 추가>

3) C Project 생성 및 환경설정

3-1) C Project 생성

① New Project → C project → Excutable → Empty Project를 선택해주고 Toolchains는 ARM Cmpiler 5(DS-5 built in)로 선택 후 프로젝트 생성

3-2) C Project Properties 설정

① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → ARM Linker 5 → Image Layout → Scatter file 설정

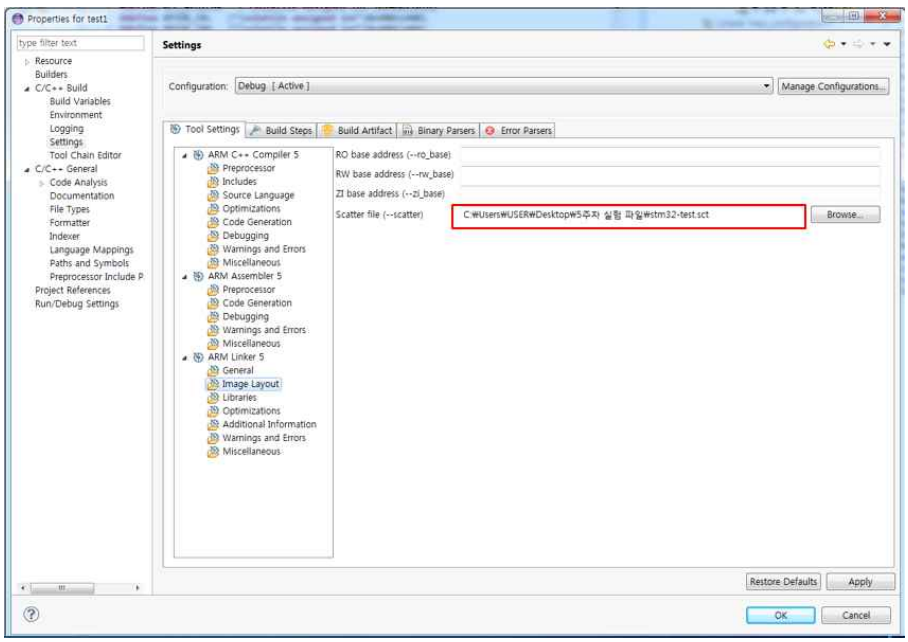
(Scatter file에서 RO/RW base address를 지정해주므로 설정해줄 필요가 없음)

② C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Code Generation과 General의 Target CPU를 Cortex M3로 설정

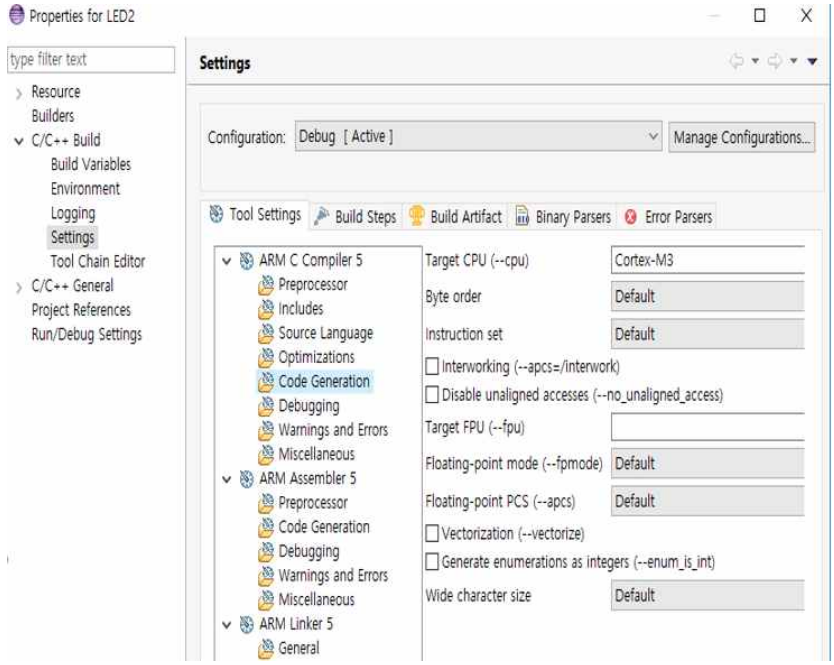
(entry point를 main으로 지정해주지 않아도 됨)

3-3) LIBRARIES 추가

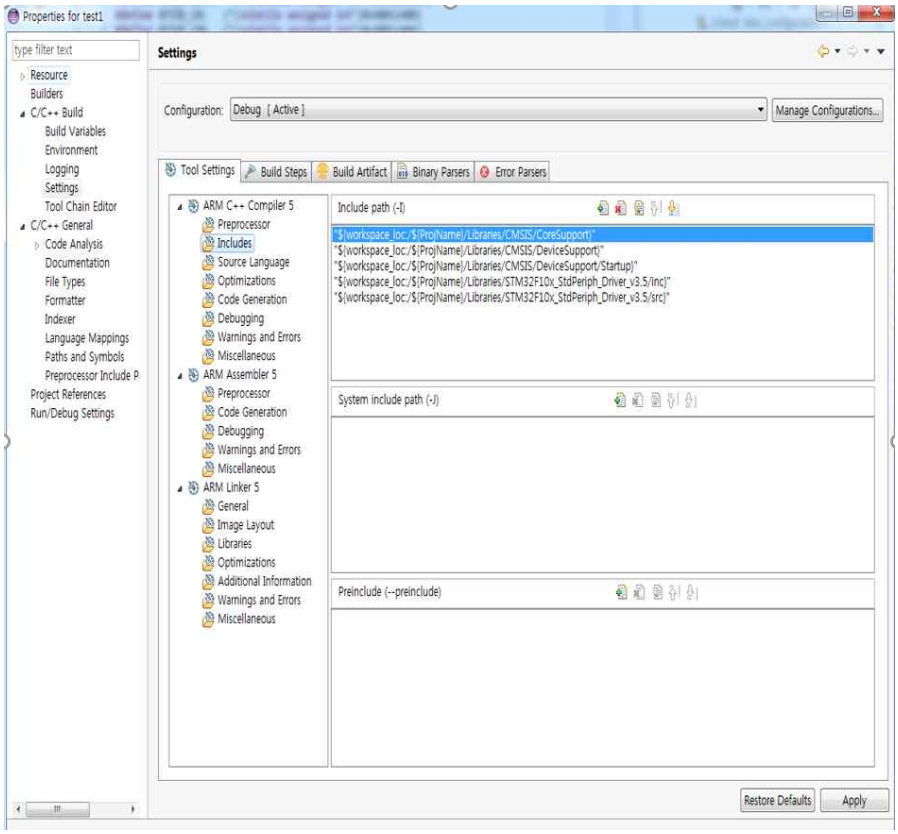
① C project 우클릭 후 Properties 선택 → C/C++ Build → Settings → Includes에서 제공되는 LIBRARIES 추가



<그림 9 : Scatter file 설정>



<그림 10 : Target CPU 설정>



<그림 11 : 제공되는 LIBRARIES 추가>

4) DS-5 Debugger 연결

4-1) Debug Configuration 설정

① Run → Debug Configuration 메뉴를 선택

② DS-5 Debugger 더블 클릭하여 새로운 하위 오브젝트 생성

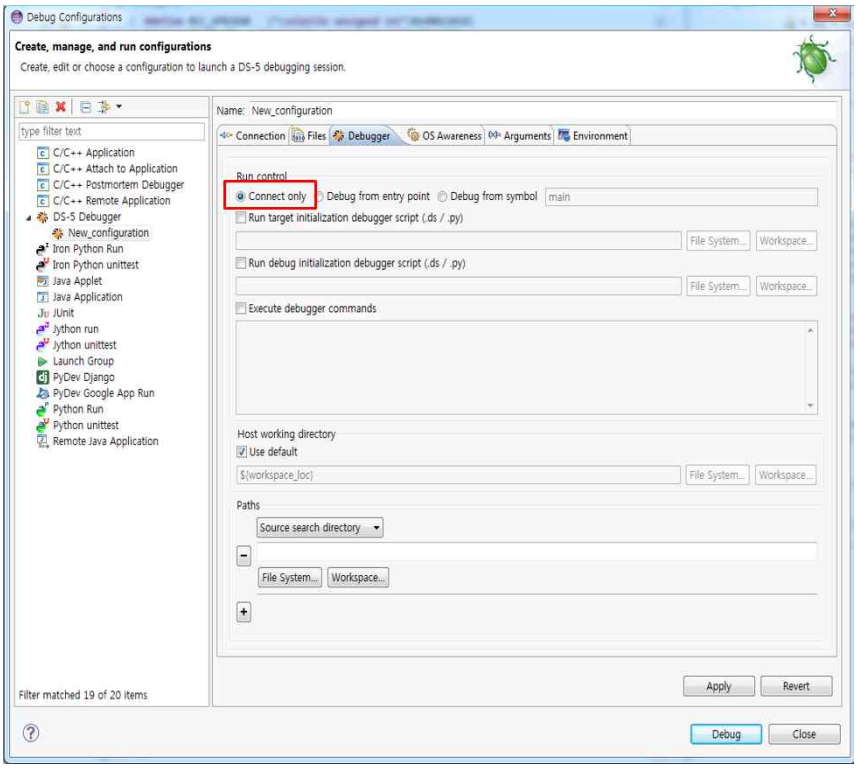
③ Name, Platform 등 Debug 환경설정을 변경

④ Browse 버튼을 클릭하여 DSTREAM 장비를 detection

4-2) Debug

① Debugger 탭에서 Connect only 체크

② Apply 클릭



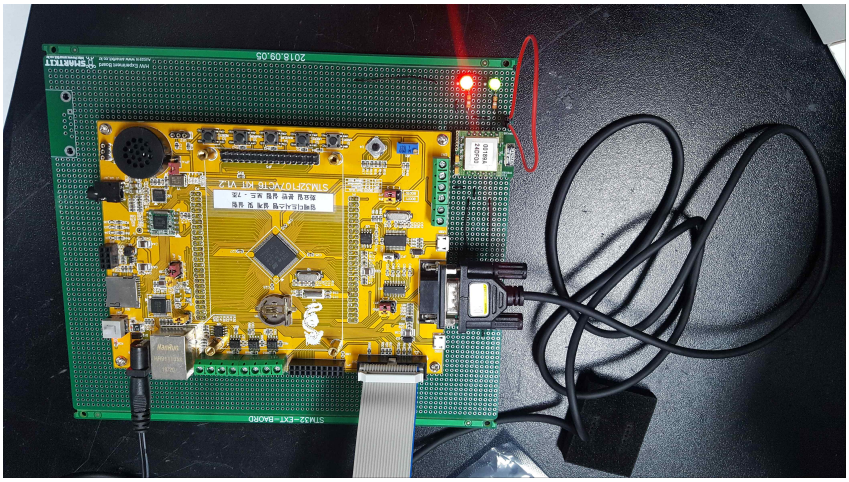
<그림 12 : Debug Configurations에서 Debugger탭의 설정화면>

5) C source code 작성

① GPIO, NVIC, USART1. USART2 코드 작성

② USART1, USART2 각각 IRQhandler 작성

6) 납땜을 통해 회로구성 및 보드와 연동



<그림 납땜을 통한 회로 구성>

7) C project 빌드 및 .axf 업로드

7-1) C project 빌드 및 Debug As

① C Project 빌드

(.axf 파일이 생성된 것을 확인할 수 있음)

② C Project 우클릭 → Debug → Debug As

7-2) flashclear.axf 및 생성된 .axf 업로드

① command 라인에 다음과 같은 명령어를 입력해서 flashclear.axf를 업로드
flash load "flashclear.axf 파일경로"

② disconnect한 다음 보드를 껐다가 켬

③ command 라인에 다음과 같은 명령어를 입력해서 생성된 .axf를 업로드
flash load "생성된 .axf 파일경로"

④ disconnect한 다음 보드를 껐다가 켬
(flash load 후에는 반드시 diconnect를 하고 보드를 껐다가 켜야 함)

7-3) 오실로스코프에 나타나는 파형 확인

① 소스코드에서 작성한 주파수와 일치하는 지 확인한다.

② 제대로 동작하지 않으면 5) C source code 작성으로 돌아감

8) 보드 연결 해체

앞서 보드 연결과 마찬가지로, 보드 연결 해체 시에도 순서를 제대로 지키지 않으면 보드가 망가질 수 있으므로 유의해야한다. 보드 연결 해체 순서는 다음과 같다.

① DS-5에서 'disconnect target'
② 보드 전원 OFF
③ DSTREAM 전원 해제 및 OFF
④ 보드 전원선 분리
⑤ DSTREAM과 보드 JTAG 분리

<표 2 : 보드 연결 해체 순서>

4. 작성한 소스코드 및 실험결과

1) 작성한 소스코드

전체 소스코드

```
//flash load "C:\Users\Team07\week08\team07\flashclear\flashclear.axf"
//flash load "C:\Users\Team07\week08\team07\Debug\team07.axf"
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_usart.h"
#include "core_cm3.h"
#include "misc.h"

int flag = 0;
char c = '1';

void GPIOA_Init() {
   GPIO_InitTypeDef GPIOA_Bluetooth;
   GPIO_InitTypeDef GPIO_USART_TX, GPIO_USART_RX;

   //GPIO_USART_RX
   GPIO_USART_RX.GPIO_Pin = GPIO_Pin_10;
   GPIO_USART_RX.GPIO_Mode = GPIO_Mode_IN_FLOATING; // Input pull down
   GPIO_USART_RX.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOA, &GPIO_USART_RX);

   //GPIO_USART_TX
   GPIO_USART_TX.GPIO_Pin = GPIO_Pin_9;
   GPIO_USART_TX.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_USART_TX.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOA, &GPIO_USART_TX);

   GPIOA_Bluetooth.GPIO_Pin = GPIO_Pin_2;
   GPIOA_Bluetooth.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIOA_Bluetooth.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOA, &GPIOA_Bluetooth);

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


void USART1_Init() {
   USART_InitTypeDef uart1;

   uart1.USART_BaudRate = 9600;
   uart1.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
   uart1.USART_Mode = USART_Mode_Rx|USART_Mode_Tx;
   uart1.USART_Parity = USART_Parity_No;
   uart1.USART_WordLength = USART_WordLength_8b;
```

```
    uart1.USART_StopBits = USART_StopBits_1;

    USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
    USART_Init(USART1, &uart1);
    USART_Cmd(USART1, ENABLE);
}

void USART2_Init() {
    USART_InitTypeDef usart2;

    usart2.USART_BaudRate = 9600;
    usart2.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    usart2.USART_Mode = USART_Mode_Rx|USART_Mode_Tx;
    usart2.USART_Parity = USART_Parity_No;
    usart2.USART_WordLength = USART_WordLength_8b;
    usart2.USART_StopBits = USART_StopBits_1;

    USART_ITConfig(USART2, USART_IT_RXNE,ENABLE);
    USART_Init(USART2, &usart2);
    USART_Cmd(USART2, ENABLE);
}

void NVIC_SET() {
    NVIC_InitTypeDef NVIC_Pin_2, NVIC_Pin_9_5, NVIC_Pin_15_10, USART1_NVIC,
USART2_NVIC;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);


    // USART1
    USART1_NVIC.NVIC_IRQChannel = USART1_IRQn;
    USART1_NVIC.NVIC_IRQChannelPreemptionPriority = 0;
    USART1_NVIC.NVIC_IRQChannelSubPriority = 3;
    USART1_NVIC.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&USART1_NVIC);

    // USART2
    USART2_NVIC.NVIC_IRQChannel = USART2_IRQn;
    USART2_NVIC.NVIC_IRQChannelPreemptionPriority = 0;
    USART2_NVIC.NVIC_IRQChannelSubPriority = 4;
    USART2_NVIC.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&USART2_NVIC);
}

void EXTI2_IRQHandler(void) {
    if(EXTI_GetITStatus(EXTI_Line2) != RESET) {
        flag = 1;
    }

    EXTI_ClearITPendingBit(EXTI_Line2);
    EXTI_ClearITPendingBit(EXTI_Line5);
    EXTI_ClearITPendingBit(EXTI_Line11);
    EXTI_ClearITPendingBit(EXTI_Line12);

}
```

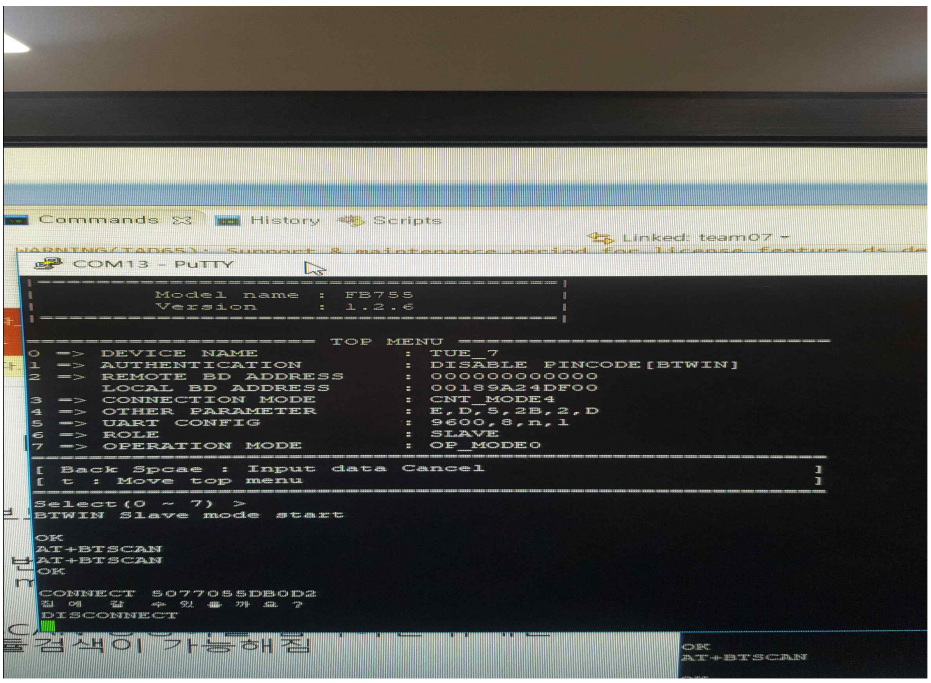
```
void EXTI9_5_IRQHandler(void) {
    if(EXTI_GetITStatus(EXTI_Line5) != RESET) {
        flag = 2;
    }
    EXTI_ClearITPendingBit(EXTI_Line2);
    EXTI_ClearITPendingBit(EXTI_Line5);
    EXTI_ClearITPendingBit(EXTI_Line11);
    EXTI_ClearITPendingBit(EXTI_Line12);
}

void EXTI15_10_IRQHandler(void) {
    if(EXTI_GetITStatus(EXTI_Line11) != RESET) {
        USART_SendData(USART1, 'G');
    }
    if(EXTI_GetITStatus(EXTI_Line12) != RESET) {
        USART_SendData(USART1, 'H');
    }
    EXTI_ClearITPendingBit(EXTI_Line2);
    EXTI_ClearITPendingBit(EXTI_Line5);
    EXTI_ClearITPendingBit(EXTI_Line11);
    EXTI_ClearITPendingBit(EXTI_Line12);
}

void USART1_IRQHandler(void)
{
    char d;
    if(USART_GetITStatus(USART1, USART_IT_RXNE)!= RESET){
        d = (char) USART_ReceiveData(USART1);

        while(!USART_GetFlagStatus(USART1, USART_FLAG_TXE));
        USART_SendData(USART1, d);

        while(!USART_GetFlagStatus(USART2, USART_FLAG_TXE));
        USART_SendData(USART2, d);

        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }
}

void USART2_IRQHandler(void) {
    char d;
    if(USART_GetITStatus(USART2, USART_IT_RXNE)!= RESET){
        d = (char) USART_ReceiveData(USART2);

        while(!USART_GetFlagStatus(USART1, USART_FLAG_TXE));
        USART_SendData(USART1, d);

        USART_ClearITPendingBit(USART2, USART_IT_RXNE);
    }
}


int main() {
```

```
    SystemInit();

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOA |
RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD | RCC_APB2Periph_USART1,
ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    GPIOA_Init();
    USART1_Init();
    USART2_Init();
    NVIC_SET();

}
```

2) 실험결과

2-1) PC에서 Putty를 통해 블루투스 모듈로부터 Configuration 화면이 로드되는 것을 확인하였다.

2-2) AT+BTSCAN 명령어를 통해 스마트폰의 입력이 putty를 통해 출력되는 것을 확인하였다.



<그림 Configuration화면과 블루투스통신을 통한 출력>

5. 결론 및 느낀점

이번 실험에서는 납땜을 잘하는 것이 중요했다. 소스코드를 잘 작성하는 것도 중요했지만, 납땜을 꼼꼼히 하지 않았거나 혹은 보드, LED, 블루투스 모듈 등 실험에 사용한 장비들이 제대로 작동했는지 등 하드웨어적인 요소에 의해서 실험의 성공여부가 많이 좌지우지 되었다. 하드웨어 실험에서는 실험 전에 사용할 장비들이 제대로 작동하는 지 꼼꼼히 체크하고 실험 중간 중간에도 계속 점검해주는 것이 필요한 것 같다. 하드웨어 실험에서는 급하게 하려면 할수록 오히려 처음부터 다시 해야 되는 경우도 종종 있어서 귀찮더라도 차분히 실험을 진행하는 것이 중요하다는 것을 깨달았다.

블루투스 모듈을 직접 사용해봄으로써, 블루투스 모듈의 동작 원리 및 과정을 이해할 수 있었다. 또한, 블루투스 통신을 위한 설정 값들을 조절하면서, 각 설정 값들이 가지는 의미에 대해서 배울 수 있었다.