임베디드 시스템 설계 및 실험 보고서

9주차 실험 결과 보고서 4조(강민진, 박기태, 이승윤, 하태승)

개요

실험목적

- 기판 납땜을 통한 보드와 모듈간 연결
- Bluetooth 모듈을 이용한 스마트폰 보드간 통신

세부목표

- 블루투스, 블루투스 프로파일, Identifier
- 블루투스 모듈과 AT 명령어 이해
- 납땜 방법 및 전선 배치

세부실험내용

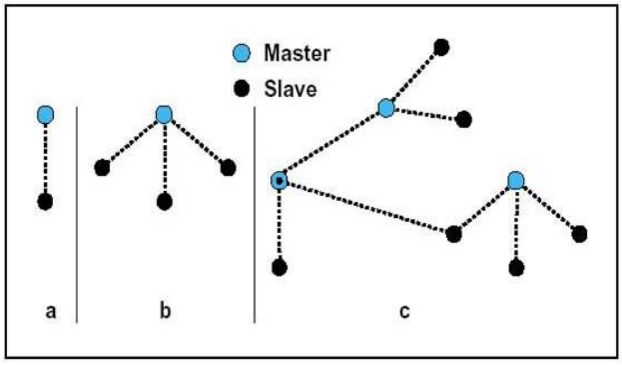
- 만능 기판 납땜
- PC의 putty 프로그램과 Bluetooth 모듈 간 통신이 가능하도록 펌웨어 작성
- Bluetooth의 CONFIG_SELECT 핀에 3v3 준 상태에서 보드를 켠 후 putty에 설정 메뉴가 뜨면 강의 자료 참고하여 설정 변경
- 안드로이드의 Serial Bluetooth Terminal 어플리케이션을 이용하여 PC의 putty와 통신
    1) PC의 putty 입력 > Bluetooth 모듈을 통해 스마트폰의 터미널에 출력
    2) 스마트폰의 터미널 입력 > PC의 putty에 출력

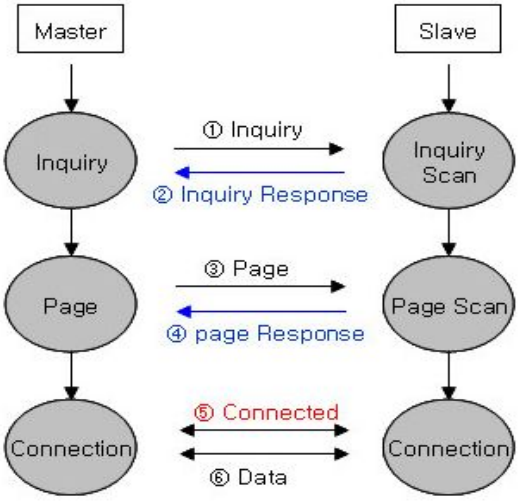
블루투스

- 유래 : 하랄드 블라톤 국왕의 이름 앞글자 ᚼ (H)와 ᛒ (B)를 결합한 모양에서 유래됨
- 단거리, 저전력, 높은 신뢰성, 저비용 무선통신
- 디바이스 간 무선통신으로 데이터를 송수신하는 기술

블루투스 작동원리

- 하나의 master 기기에 최대 7개의 slave 기기 연결 가능
- 동작 사이클
    1) master가 주변의 slave 에 검색
    2) slave는 연결요청을 받음
    3) 연결이 성공되면 master가 slave에게 데이터를 송신함
- 허가 없이 사용 가능한 2.4MHz ISM(Industrial, Scientific, Medical) 주파수 대역 사용
- 디바이스간 같은 주파수 대역을 사용하기 때문에 전파 간섭을 방지하기 위해 주파수 호핑 방식을 사용
  *주파수 호핑방식 : 많은 채널을 특정 패턴에 맞추어 빠르게 이동하면서, 데이터를 조금씩 전송하는 기법





- Master가 주변의 Slave를 찾으면(Inquire), Slave는 자신의 정보를 Master에게 송신한다(Inquiry Response).
- Slave의 정보가 Master와 일치하면 연결이 이루어 지고 데이터 전송이 가능하게 된다.
- Slave 끼리는 통신이 불가능하다.

블루투스 프로파일과 Identifier

블루투스 프로파일

블루투스 프로파일은 어플리케이션 관점에서 블루투스 기기의 기능별 성능을 정하는 사양을 의미한다. 다시 말해 블루투스 프로파일은 어떤 하나의 블루투스 기기와 다른 블루투스 기기가 서로 통신하는데 사용하는 특성을 규정한다.

하나의 블루투스 프로파일이 존재하는 것이 아니라 여러가지 다양한 프로파일이 존재하며 그 중 SPP는 RS232 시리얼 케이블 에뮬레이션을 위한 블루투스 기기에 사용되는 프로파일로 마치 RS232 시리얼 케이블이 연결된 상태에서와 같이 무선 블루투스 통신을 수행할 수 있다.

Identifier

고유식별자로 SSID와 UUID가 있다.

- SSID

SSID는 무선 랜을 통하여 클라이언트에 접속할 때 각각의 무선랜들을 구별하기 위한 고유식별자로 와이파이의 경우 각 와이파이 네트워크를 구별하기 위해 사용된다.

- UUID

UUID는 네트워크 상에서 서로 다른 개체들을 구별하기 위한 128비트 고유 식별자로 블루투스에서 서비스의 종류를 구분하기 위해 사용된다.

블루투스 모듈 FB755AC

- 최대 7대까지 Slave로 연결될 수 있으며, 최대 100m 거리 까지 인식 가능하다.
- AT 명령어를 이용하여 모듈을 제어할 수 있다.
- 노트북, 스마트폰, PDA 등의 장치와 연결 및 통신이 가능하다.
- 12개의 핀으로 구성되며 자세한 구성은 아래와 같다.

1 STATUS	12 VCC
2 FACTORY RESET	11 MESSAGE CONTROL
3 STREAM CONTROL / DSR	10 MESSAGE STATUS
4 STREAM STATUS / DTR	9 RXD
5 CONFIG SELECT	8 TXD
6 CONNECT CHECK / DCD	7 GND

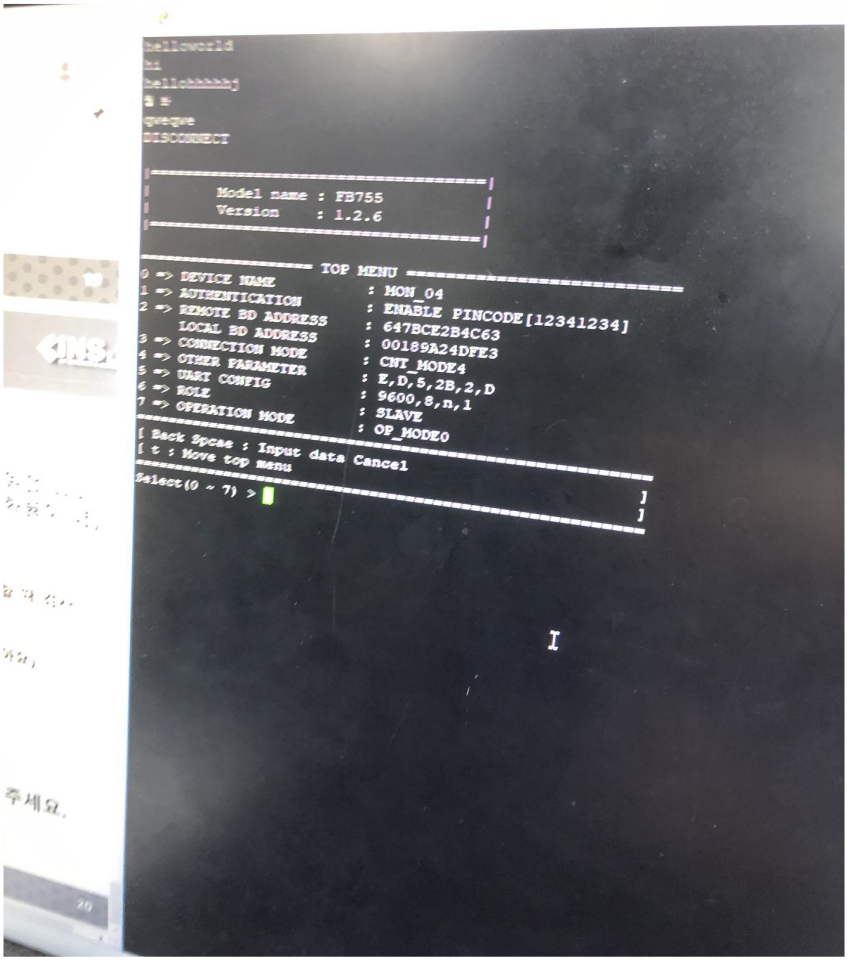
STATUS
: 상태를 표시하는 핀
CONFIG SELECT
: 블루투스 모듈 설정할때 HIGH를 입력한 채로 전원을 켜면 설정모드 진입한다.

AT 명령어

AT 명령어는 블루투스를 포함하여 여러 모뎀 모듈을 제어하기 위해 사용되는 명령어이다. AT 명령어 세트를 이용하여 앞에서 살펴본 FB755AC 모듈을 제어한다.

아래는 실험 중 사용한 몇가지 AT 명령어이다.

- AT : 정상적으로 연결되어 있는지 확인을 위해 사용하는 명령이다. 정상적으로 연결된 경우 OK를 출력한다.
- AT+RST : 모듈을 리셋하는 명령이다.
- ATO : 명령어 대기 상태로 전환하는 명령이다.
- ATOn : master 를 명령어 대기 상태로 전환하는 명령이다.
- ATH : 연결 상태를 해제하는 명령이다.
- AT+BTINFO? : 현재 모듈의 상태를 출력하는 명령이다. 각 상태에 따라 STANDBY, PENDING, CONNECT가 출력된다.
- AT+BTNAME : 현재 장치의 이름을 출력하거나 변경(=변경할 이름)하는 명령이다.
- AT+BTROLE : 마스터 또는 슬레이브 상태로 변경하는 명령이다.
- AT+BTSCAN : BT를 검색 및 연결 대기 상태로 전환하는 명령이다.

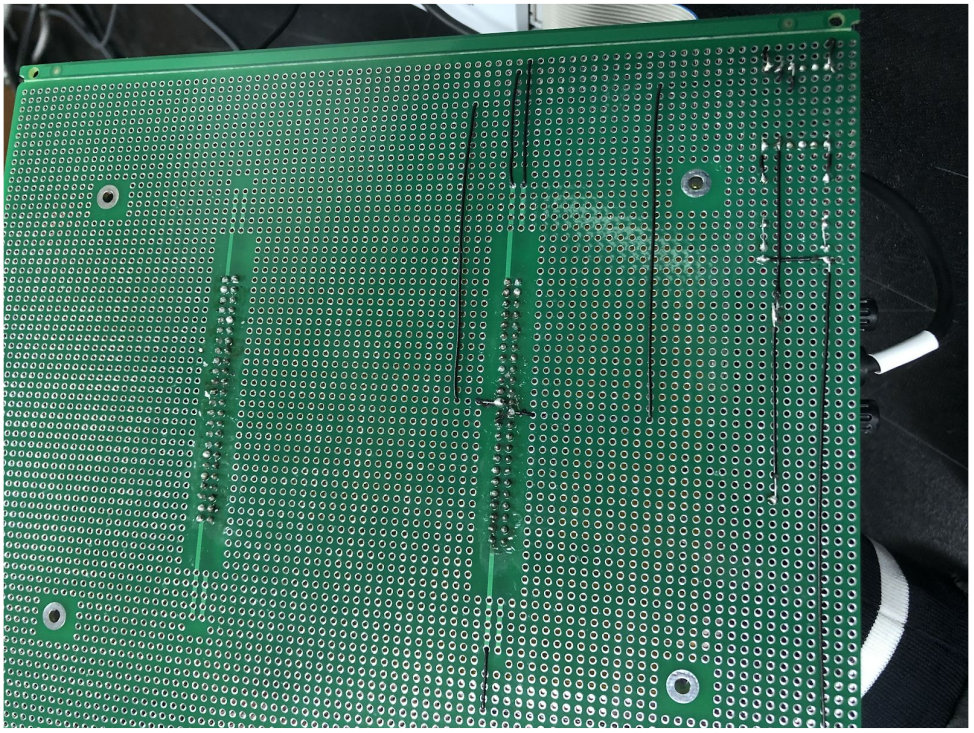


납땜

지난 실험에서는 보드를 기판에 부착하기 쉽게 하기 위한 보드 장착부를 납땜하였다. 이번 실험에서는 보드와 블루투스 모듈을 연결하는 회로를 구성하고 블루투스 모듈 장착부를 납땜하여 기판을 통해 보드와 블루투스 모듈이 연결될 수 있도록 하였다.

납땜 과정은 아래와 같다.

1) VCC, GND를 보드의 3.3V와 GND를 연결하였다.
2) 1번 핀(STATUS), 6번 핀 (DCD)는 저항과 LED에 연결하였다.
3) 8번 핀(TXD), 9번 핀(RXD)는 보드의 RX와 TX에 교차하여 연결하였다.
4) 5번 핀(CONFIG SELECT)는 3.3V에 연결하였다.



지난 시간과 달리 납땜만 진행한 것이 아니라 이번 실험에서는 전선도 함께 사용하였다. 기타 이번 실험에서 납땜시 주의해야 할 점은 다음과 같다.

- 납땜 할 때 전선이 끊어지거나 접촉 불량이 나는 것을 방지하기 위해서 전선을 최대한 팽팽하게 연결해야 한다.
- 다른 외부 영향을 받지 않고 전기적으로 잘 연결되도록 밀폐하여 납땜한다.
- 느슨하거나 사선으로 납땜하지 않고 팽팽하게 연결한다.
- 인두기를 이용하여 전선을 피복을 잘 녹여준 뒤에 납땜해주어야 전선을 통해 전류가 잘 전달될 수 있다.
- LED는 긴 쪽이 +(Vcc)로 방향을 잘 확인하여 납땜해야 한다.
- 인두기 사용 시 자리를 비울 때는 콘센트를 뽑고, 주변 환경을 살피며 안전하게 실습을 진행하였다.

소스코드

```
#include "stm32f10x.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_rcc.h"

#include "misc.h"

/* function prototype */
void RCC_Configure(void);
void GPIO_Configure(void);
void EXTI_Configure(void);
void USART1_Init(void);
void NVIC_Configure(void);

void EXTI15_10_IRQHandler(void);

void Delay(void);

void sendDataUART1(uint16_t data);

char myFlag = 0;
char wordFlag = 0;
char toggleFlag = 0;

void RCC_Configure(void)
{
  GPIO_InitTypeDef GPIO_LED;
      // TODO: Enable the APB2 peripheral clock using the function
'RCC_APB2PeriphClockCmd'

        /* UART TX/RX port clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

      /* USART1 clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
// USART2 clock enable
        /* USART2 clock enable */
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

      /* Alternate Function IO clock enable */
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
}

void GPIO_Configure(void)
{
```

```
    GPIO_InitTypeDef GPIO_InitStructure;

      // TODO: Initialize the GPIO pins using the structure
'GPIO_InitTypeDef' and the function 'GPIO_Init'

    /* UART1 pin setting */
    //TX
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //RX
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; // *Floating*?
    GPIO_Init(GPIOA, &GPIO_InitStructure);
// USART2 TX, RX 설정
    /* USART2 */
    //TX
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //RX
    GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; // *Floating*?
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void USART1_Init(void)
{
    USART_InitTypeDef USART1_InitStructure;

    // Enable the USART1 peripheral
    USART_Cmd(USART1, ENABLE);

    USART1_InitStructure.USART_BaudRate = 9600;
    USART1_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART1_InitStructure.USART_StopBits = USART_StopBits_1;
    USART1_InitStructure.USART_Parity = USART_Parity_No;
    USART1_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART1_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;

    USART_Init(USART1, &USART1_InitStructure);
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
```

```
// USART2 설정
    USART_InitTypeDef USART2_InitStructure;

    // Enable the USART2 peripheral
    USART_Cmd(USART2, ENABLE);


    USART2_InitStructure.USART_BaudRate = 9600;
    USART2_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART2_InitStructure.USART_StopBits = USART_StopBits_1;
    USART2_InitStructure.USART_Parity = USART_Parity_No;
    USART2_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART2_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;

    USART_Init(USART2, &USART2_InitStructure);
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);
}

void NVIC_Configure(void) {

    NVIC_InitTypeDef NVIC_InitStructure;
// PriorityGroup을 2로 설정
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    NVIC_Init(&NVIC_InitStructure);

    // UART1
      // 'NVIC_EnableIRQ' is only required for USART setting
    NVIC_EnableIRQ(USART1_IRQn);
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // UART2
      // 'NVIC_EnableIRQ' is only required for USART setting
    NVIC_EnableIRQ(USART2_IRQn);
    NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // TODO
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; // TODO
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

void USART1_IRQHandler() {
      uint16_t word;
    if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET){
      // the most recent received data by the USART1 peripheral
        word = USART_ReceiveData(USART1);
```

```
        USART_SendData(USART2, word);
        // clear 'Read data register not empty' flag
      USART_ClearITPendingBit(USART1,USART_IT_RXNE);
    }
}
// USART1과 마찬가지로 USART2 IRQ Handler 설정
void USART2_IRQHandler() {
      uint16_t word;
    if(USART_GetITStatus(USART2,USART_IT_RXNE)!=RESET){
      // the most recent received data by the USART1 peripheral
        word = USART_ReceiveData(USART2);
        USART_SendData(USART1, word);
        // clear 'Read data register not empty' flag
      USART_ClearITPendingBit(USART2,USART_IT_RXNE);
    }
}

void Delay(void) {
      int i;

      for (i = 0; i < 2000000; i++) {}
}



int main(void)
{
    SystemInit();

    RCC_Configure();

    GPIO_Configure();

    USART1_Init();

    NVIC_Configure();

    while(1);

    return 0;
}
```

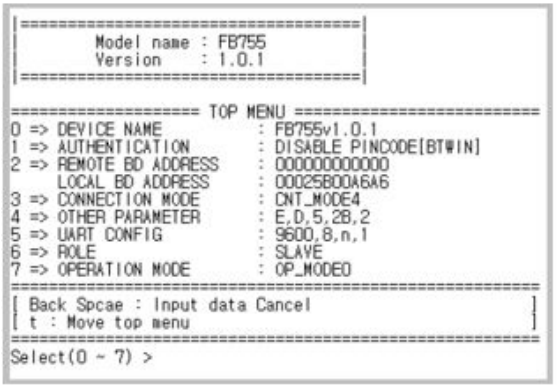
8주차의 코드는 PC(putty)와 Cortex-M3 보드가 USART1을 통하여 통신을 할 수 있는 코드였다. 이번 실험은 Bluetooth 모듈과 Cortex-M3 보드가 블루투스로 통신을 하도록 USART2 에 대한 코드를 추가하였다. 기존의 USART1과 똑같이 함수를 만들어 주었다.

주의할 점으로는 USART2의 TX와 RX의 port에 대한 clock 인가 정도이다. USART2의 TX, RX도 USART1과 마찬가지로 A포트를 사용하고 있으며 2, 3번 핀을 사용해서 설정하였다.

동작 과정

1) 위와 같이 블루투스(USART2)로 통신하도록 코드를 작성한다.

Configuration Menu



- 0. Device Name: 최대 12문자
- 1. Pin code: 두 BT device의 pin code가 동일해야 함.
- 2. Device Address: 마지막에 연결된 BT device의 MAC address. Default는 000000000000
- 3. Connection Mode: 기본값 mode4 (명령어 대기상태)
- 4. Parameter
- 5. UART Config: baud rate, data bit, parity bit, stop bit
- 6. Role : (Slave, Master)
- 7. Operation Mode : (OP_MODE0~2)

2) CONFIG SELECT를 점프선으로 3.3V를 입력한 상태에서 부팅하면 Putty에 Configuration 설정 창 이 표시되고 각 항목에 대한 설정을 할 수 있다.
   - **Device name**은 MON_04로 설정한다.
   - **pincode**를 설정한다. (Bluetooth 연결 비밀번호)
   - **CNT_MODE4** SLAVE 설정한다.
   - **UART CONFIG** (9600, 8, n, 1)로 설정한다. (순서대로 Baud rate, Data bit, Parity bit, Stop bit)
3) 앞 과정에서 설정 모드로 부팅하기 위해 사용하였던 CONFIG SELECT에 전원 공급을 중단한(선을 뽑은) 뒤 보드를 재부팅 했을 때, **BTWIN Slave mode start**라는 출력이 나오면 **AT+BTSCAN**으로 BT가 검색 및 연결 대기 상태가 되도록 전환하는 AT 명령어를 입력한 후 스마트폰에서 블루투스 모듈을 검색, 등록(페어링)한다.
4) 스마트폰에 Serial Bluetooth Terminal 어플리케이션을 설치한다.
5) 스마트폰과 블루투스가 연결되면 연결되었다는 메시지(**CONNECT**)가 터미널에 출력이 된다.
6) 이제 각 장치에서 입력한 내용을 대응되는 장치에서 확인할 수 있다.

터미널을 통하여 통신할 시 간단한 팁 및 고려할 사항 :

- Putty 커맨드 창에 입력 시 입력한 글자가 보이지 않는 문제가 발생하였는데 Putty 설정 창에서 이를 보이도록 설정하여서 수월하게 AT 명령어 관련 실습을 진행할 수 있었다.
- 명령어 입력시 Buffer를 사용하여 전달하기 때문에 BackSpace도 버퍼에 함께 저장된다. 버퍼를 삭제하기 위해서 처음부터 다시 입력해야하는 번거로움이 있다.

결론

블루투스는 짧은 거리에서 여러 기기간 데이터를 주고 받을 수 있는 통신 방법이다. 저전력, 저가로 무선 통신 환경을 구축할 수 있고 다른 블루투스 기기와 통신하기 위해 블루투스 프로파일을 사용한다. 마치 유선 케이블로 연결된 것과 같이 사용하기 위해 SPP 프로파일을 이용할 수 있으며 블루투스 서비스 종류를 구분하기 위해 UUID를 고유 식별자로 사용한다.

블루투스 모듈을 직접 사용하여 스마트폰과 통신하는 과정에서 블루투스 모듈의 동작 원리 및 과정을 이해할 수 있었다. 또한, 블루투스 통신뿐만 아니라 거의 모든 모뎀에 대응되는 AT 명령어를 다뤄 설정 값들을 변경하여 각 설정 값들이 가지는 의미에 대해서 배울 수 있었다.

이번 실험은 USART 통신을 사용하여, 블루투스 모듈과 통신하는 방법에 대한 실험이었다. 보드를 중심으로, 컴퓨터(putty)와는 USART1 통신, 블루투스 모듈과는 USART2를 통해 통신하였다.

만능기판에 납땜을 할 때 주의사항이 있었는데, USART1의 Tx는 USART2의 Rx, USART1의 Rx는 USART2의 Tx와 연결해야 한다는 점이다. 이는 각 장치의 TX(Transmitter)가 대응되는 장치의 RX(Receiver) 와 연결되어야 입력이 상대방 장치의 출력으로 나타날 수 있기 때문이다. 마찬가지로 블루투스 모듈에서 값을 받아오는 방법 또한 동일한 방식으로 TX와 RX를 교차하여 연결했다.

이번 실험에서는 납땜이 중요했다. 물론 코드를 작성하는 것도 중요했지만, 납땜을 잘못한 경우, 즉 하드웨어적인 요소가 잘못된 경우 원인을 파악하기가 힘들기 때문에 납땜을 제대로 수행한 것이 이번 실습을 제 시간에 마무리하는데 중요한 요소였다.

텀 프로젝트에 필수적인 통신요소로 블루투스 모듈을 사용하는데, 이 모듈을 이번 9주차 실습을 통해 동작 및 동작방식을 학습한 것이 이후 텀프로젝트 진행 시에 도움이 될 것 같다.