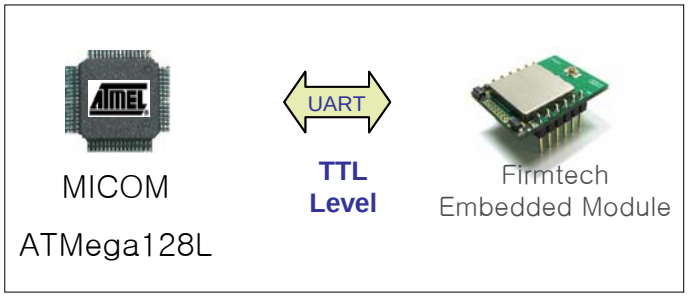
FBDx5xMC

< FBD755MC_gcc V0.2 >

1. 개요

- 본 문서는 MICOM으로 ㈜펌테크 제품인 Bluetooth Embedded Module을 제어하기 위한 하드웨어 및 소프트웨어 사용 방법과 요령을 소개하는 문서로서 사용자들이 구현하고자 하는 Target Application 구성을 보다 빠르고 쉽게 구성할 수 있도록 도움을 주는데 목적을 둡니다.

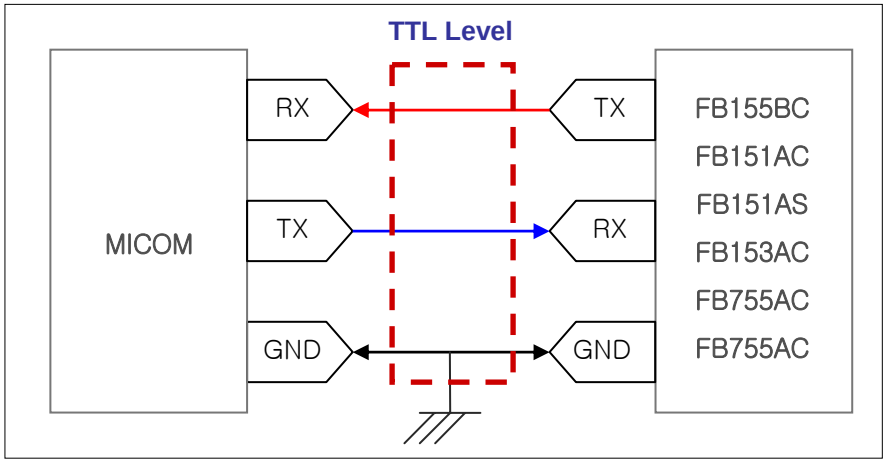
<1> 하드웨어 구성의 이해



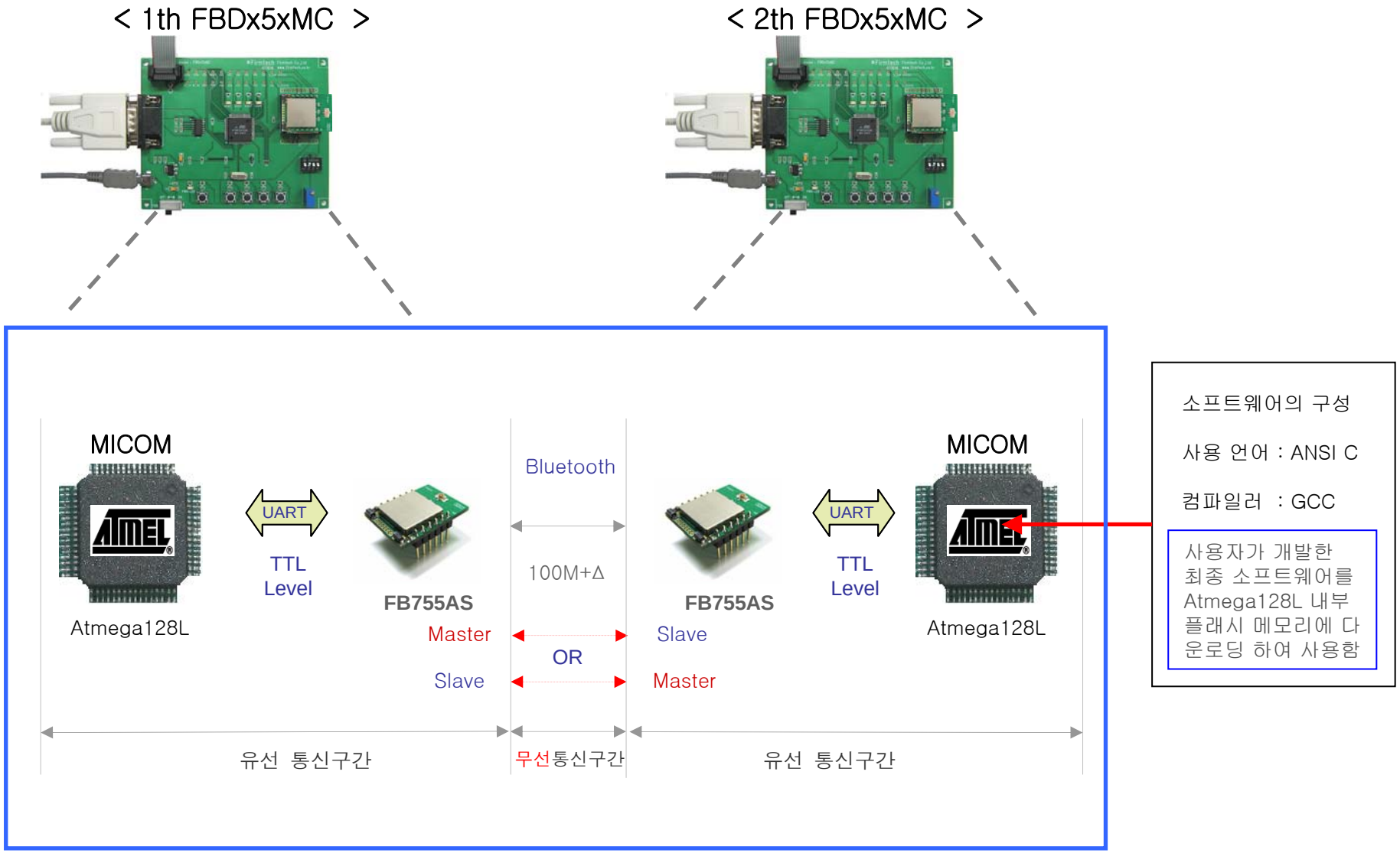
참고 : UART : Universal Asynchronous Receiver Transmitter

MICOM의 경우 현재 시중에 널리 사용되는 ATMega128L 사용

<2> MICOM 과 Firmtech Bluetooth Embedded Module 과의 인터페이스 방법 (기본 인터페이스)



2. 전체 구성도

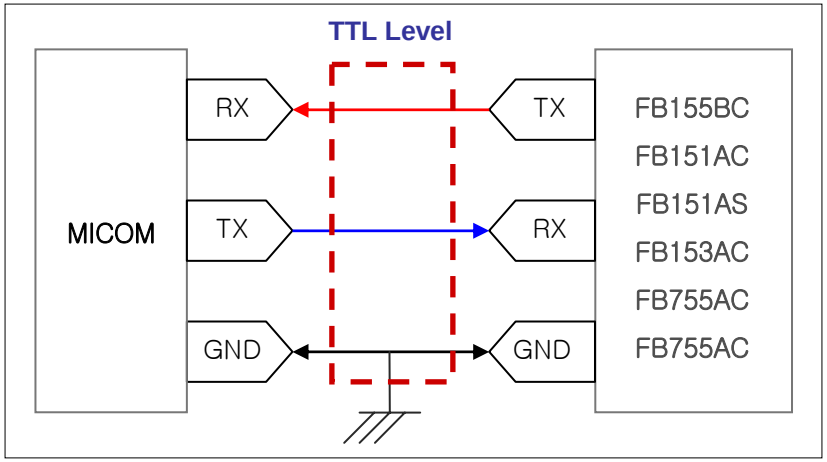

< 1th FBDx5xMC >
< 2th FBDx5xMC >
MICOM
Atmega128L
UART
TTL
Level
FB755AS
Master
Slave
Bluetooth
100M+Δ
OR
Slave
Master
FB755AS
UART
TTL
Level
MICOM
Atmega128L
유선 통신구간
무선통신구간
유선 통신구간
소프트웨어의 구성
사용 언어 : ANSI C
컴파일러 : GCC
사용자가 개발한
최종 소프트웨어를
Atmega128L 내부
플래시 메모리에 다
운로딩 하여 사용함

3. FB755AS 1:N 컨트롤을 위한 인터페이스

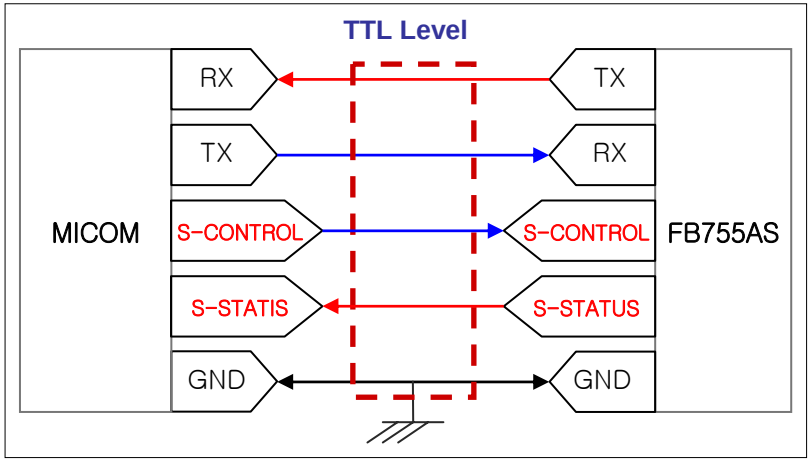
- MICOM과 ㈜펌테크의 Bluetooth Embedded Module을 “기본 인터페이스”로 구성한 경우, 1개의 Master와 1개의 Slave 연결 가능

- MICOM과 ㈜펌테크의 FB755AS Embedded Module을 “1:N 컨트롤을 위한 인터페이스”로 구성한 경우,1개의 Slave에 최대 7개의 Master 연결 가능

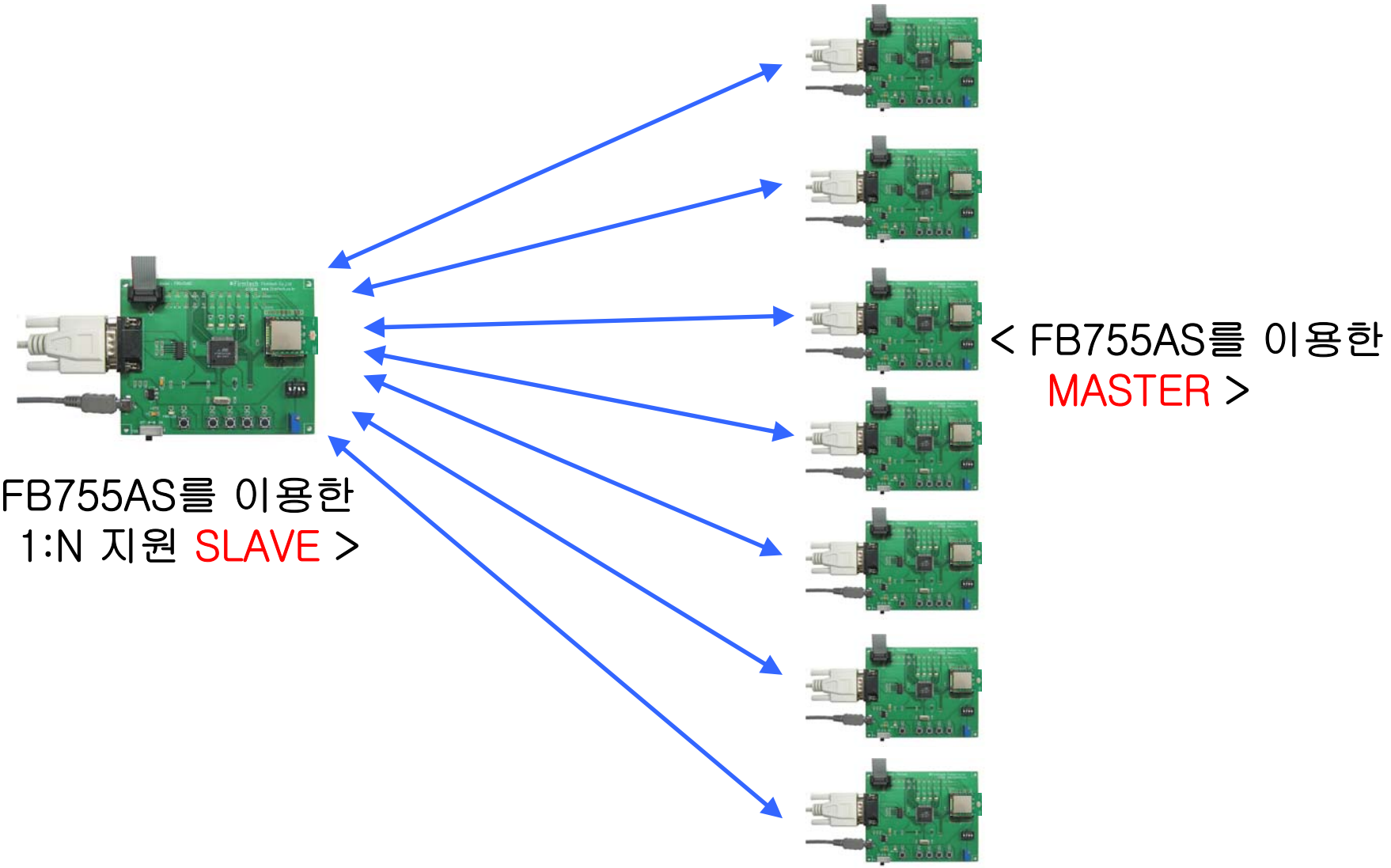
<1> MICOM 과 Bluetooth Embedded Module 과의 인터페이스 방법 (기본 인터페이스)



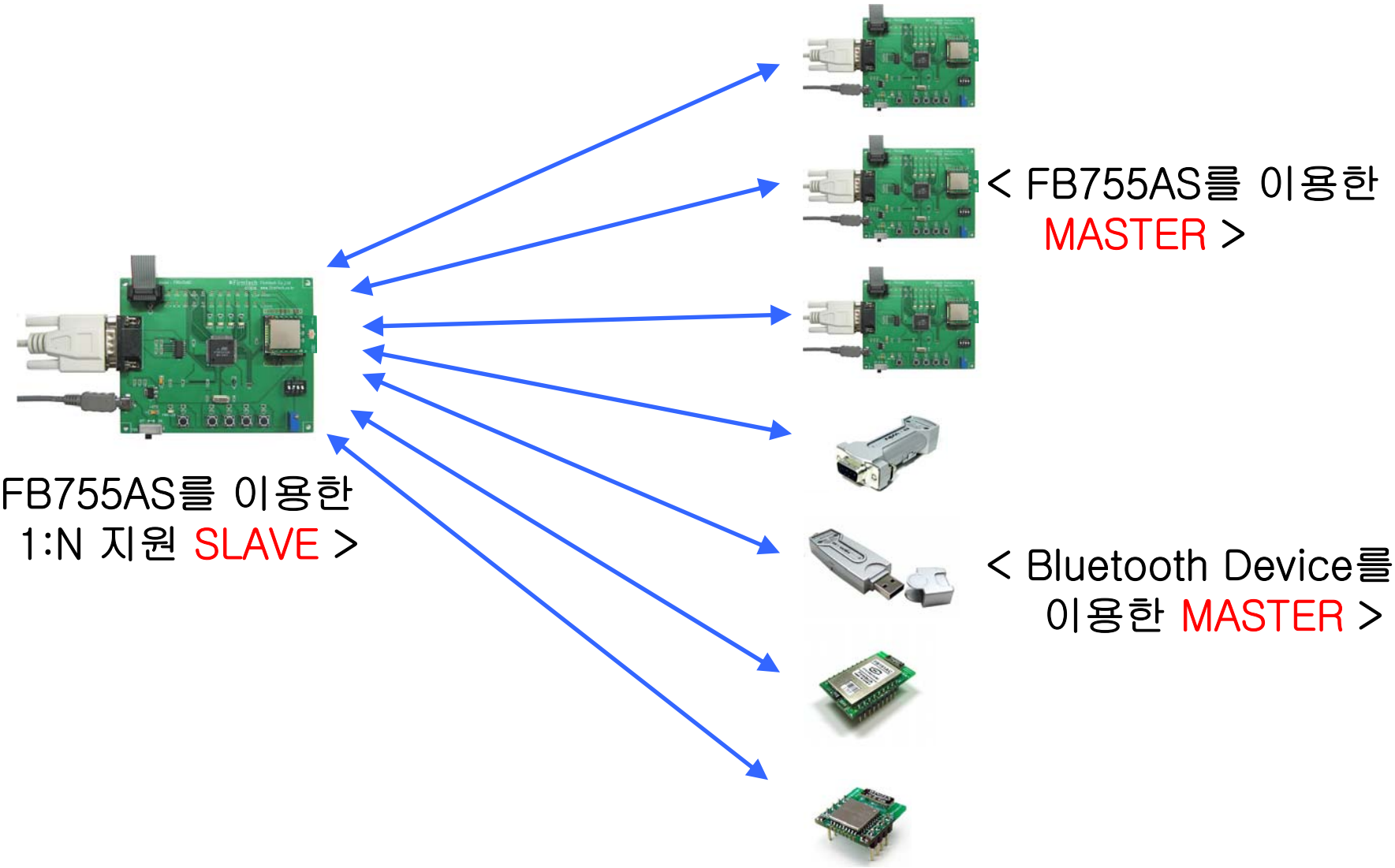
<2> MICOM 과 FB755AS 1:N 컨트롤을 위한 인터페이스 방법



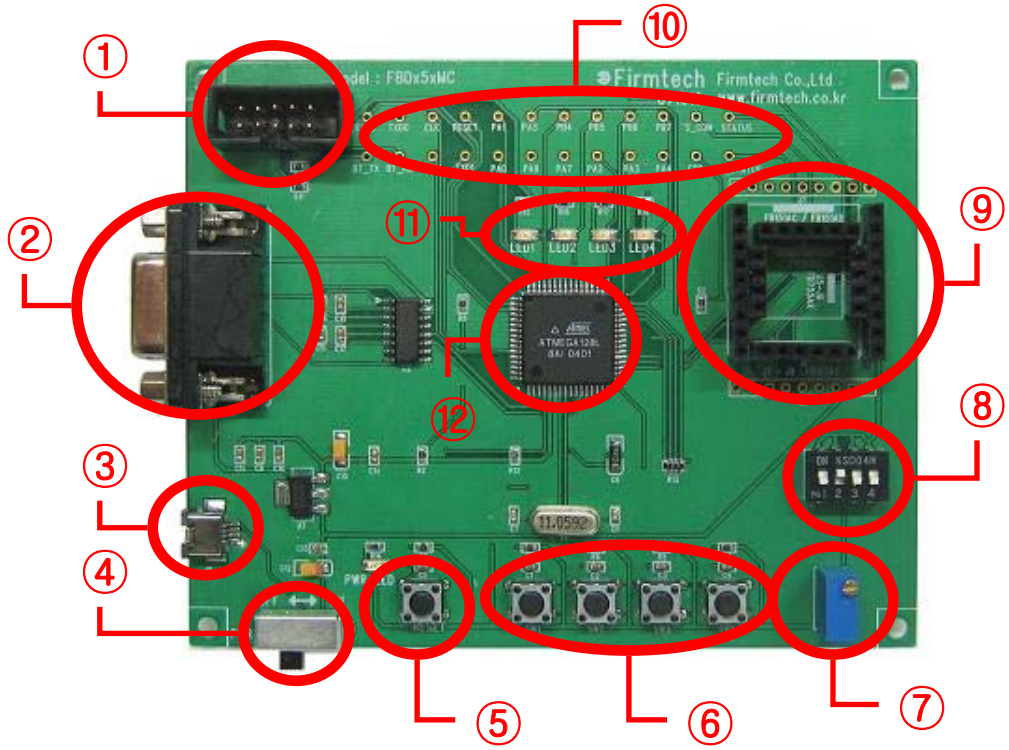
4. FB755AS를 이용한 1:N 실제 구성도 – FB755AS & FBDx5xMC 사용



5. FB755AS를 이용한 1:N 실제 구성도 – Bluetooth Device 사용



6. FBDx5xMC 제품 외형



< FBDx5xMC 제품 외형 >

1. AVR Loader 연결 커넥터
2. RS-232 시리얼 연결 커넥터
3. USB 전원 연결 커넥터
4. 전원 ON/OFF 스위치
5. 리셋 스위치 (ATMega128L 리셋 용)
6. 스위치 (데이터 입력 용)
7. 가변 저항 (데이터 입력 용)
8. 딥 스위치 (데이터 입력 용)
9. 임베디드 제품 연결 커넥터
10. 테스트 포인트 (디버거 용)
11. LED (데이터 출력 용)
12. ATMega128L (프로그램용 MICOM)

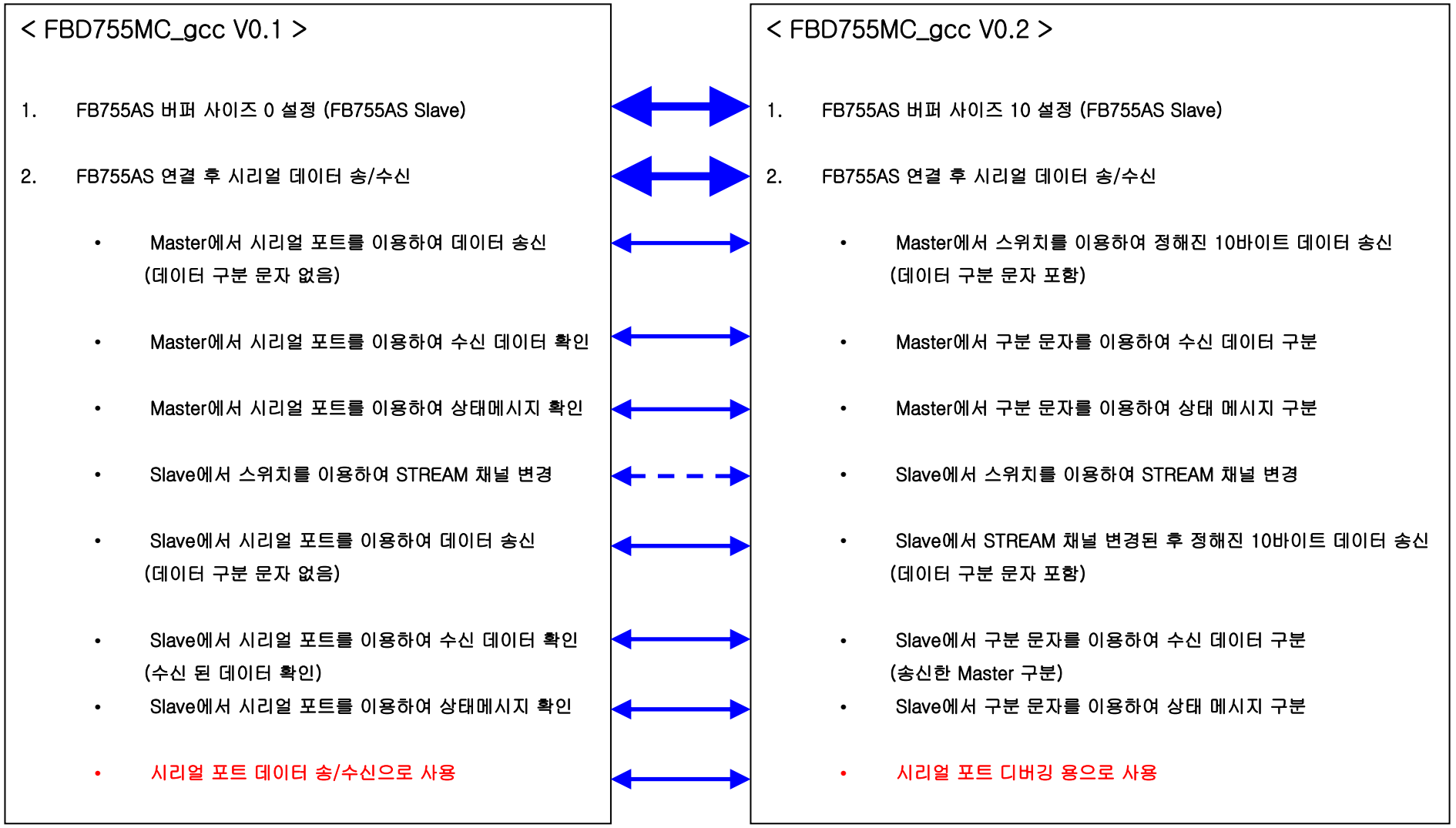
7. FBDx5xMC (FBD755MC_gcc) 제품의 기본 구성 품

Model NO.	Pictures	Q'ty	Description
FBDx5xMC		1	임베디드 블루투스 컨트롤 보드
FBA180SC		1	RS-232 시리얼 연결 케이블
FBA002PO		1	USB 전원 공급 케이블
CD		1	Source, Datasheet, Manual, Utility CD
AVR Loader		1	프로그램 다운로드 케이블
FB755AS		1	임베디드 블루투스 모듈 (1:N 지원)

8. FBD755MC_gcc V0.2 소개

- FBD755MC_gcc V0.2는 ㈜펌테크의 임베디드 모듈인 FB755AS(1:N 연결 지원)에서 제공되는 AT-Command (CNT MODE4)를 이용하여 1: N 모니터링 방식 (OP MODE1)의 데이터 송-수신을 MICOM으로 제어하는 프로그램 소스 명
- 기본적인 운영은 FBD755MC_gcc V0.1 기반으로 운영됨
- FB755AS의 버퍼 사이즈 10 설정(10byte의 데이터 수신 설정)
  - ✓ FB755AS(1:N 지원 Slave)의 버퍼 사이즈를 10으로 설정하는 것은, FB755AS와 연결되는 Master에서 데이터를 10 byte로 전송하는 것을 의미함
- Master에서 FB755AS(1:N 지원 Slave)로 데이터를 송신 할 때 "데이터 구분 문자"가 포함된 형태의 데이터 송신
  - ✓ 1번 Master에서 송신하는 데이터 형태 : 1ABCDEFGH1 (10 byte)
  - ✓ 2번 Master에서 송신하는 데이터 형태 : 2ABCDEFGH2 (10 byte)
  - ✓ 7번 Master에서 송신하는 데이터 형태 : 7ABCDEFGH7 (10 byte)
- Master에서 FB755AS(1:N 지원 Slave)로 데이터를 송신 할 때 "데이터 송신 간격"을 적용하여 송신
  - ✓ Master에서의 데이터 송신 간격은 FB755AS(1:N 지원 Slave)에 연결되는 Master 디바이스 수에 따라 달라짐
  - ✓ 연결되는 Master 디바이스 수가 1개씩 늘어나면 데이터 송신 간격은 400ms씩 늘어남
- FB755AS(1:N 지원 Slave)에서는 각 데이터의 구분 문자를 이용하여 수신 받은 데이터가 어떤 Master로부터 수신 받았는지를 구분함
- FB755AS(1:N 지원 Slave)는 Master에서 수신 받은 데이터와 FB755AS에서 출력하는 상태 메시지(CONNECT/DISCONNECT)를 구분함

9. FBD755MC_gcc V0.1 VS FBD755MC_gcc V0.2



10. CNT_MODE와 OP_MODE

- CNT_MODE (Master/Slave 연결 방법)
  - ✓ CNT_MODE1 : 저장되어 있는 Bluetooth Address를 이용하여 Master/Slave가 자동으로 연결.
    저장된 Address가 없는 경우, 주위의 PIN Code가 같은 Bluetooth Device와 자동으로 연결.
  - ✓ CNT_MODE2 : Master에서 주변의 Slave를 검색하고, Slave를 선택하여 Master/Slave 연결.
    사용자가 검색/선택에 대한 동작을 진행 해야 함.
  - ✓ CNT_MODE3 : CNT_MODE1과 동일
    저장되어 있는 Bluetooth Address를 사용자가 변경 가능함.
  - ✓ CNT_MODE4 : 정의되어 있는 AT-Command를 사용하여 Master/Slave 연결.
    AT-Command를 입력하지 않으면 Bluetooth Device는 아무런 동작을 하지 않음.
- OP_MODE (Master/Slave 데이터 송/수신 방법)
  - ✓ OP_MODE0 : 1 대 1 데이터 송/수신 방법.
  - ✓ OP_MODE1 : 1 대 N 모니터링 데이터 송/수신 방법.
    1개의 Slave는 N개의 Master로부터 데이터를 수집.
    필요 시 1개의 Slave는 N개의 Master중 1개를 선택하여 데이터 송신.
  - ✓ OP_MODE2 : 1 대 N 선별적 대 용량 데이터 송/수신 방법
    1개의 Slave는 N개의 Master중 1개를 선택 (원하는 경우에 N개의 Master중 1개 선택).
    1개의 Slave는 선택한 Master와 데이터 송/수신(이때는 1대1과 같음).

11. FBD755MC_gcc V0.2 Master 데이터 송신 간격

< compare_master_command( ) >

```
case STEP_8:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nCONNECT",&uart0_read_data[uart0_length-24],9))
    {
        step_master_bt++;
        status_sequence = AFTER_CONNECT;
//연결 디바이스 수가 1대 추가 되면 데이터 송신 간격이 400ms 추가됨(1대 : 400ms, 2대 : 800ms, 7대 : 2800ms)
        transfer_value = ((4*(CONNECT_DEV_NUM-0x30)))*34;
        DispStr1("[STEP_8 : COMMUNICATION START]");
    }
    else if(!memcmp("\r\nERROR\r\n",&uart0_read_data[uart0_length-9],9))
    {
        step_master_bt = STEP_7;
        status_target = TARGET_DETECT;
        DispStr1("[STEP_8 : ERROR MSG RECEIVED]");
        DispStr1_line("[       TRY TO RE-CONNECT ");
        for(i = 0;i < 12;i++)
                PutChar1(target_data[i]);
        DispStr1("]");
        wait_1ms(100);
        DispStr0("atd");
        for(i = 0;i<12;i++)
            PutChar0(target_data[i]);
        PutChar0(0x0d);
    }
    else
        execution_error();
    break;
```

< Master Device 데이터 송신 간격 계산 >

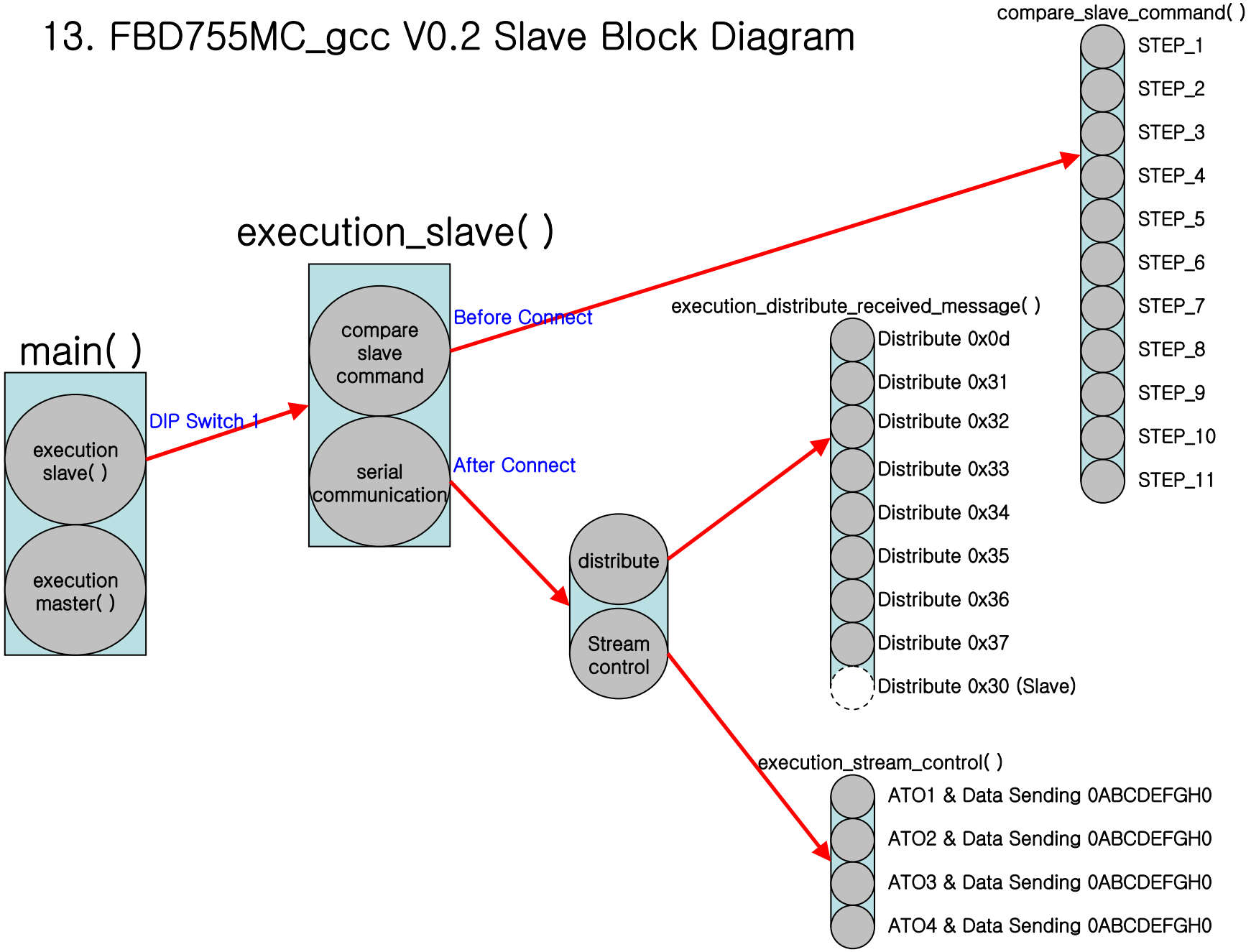
1. transfer_value에 송신 간격 값 저장
2. CONNECT_DEV_NUM : 연결될 디바이스 수
3. CONNECT_DEV_NUM – 0x30 : 1~7의 값 계산됨
4. (4*(1~7로 계산된 값))*34 : 1대당 400ms씩 추가하는 작업
5. (4*(1~7로 계산된 값))*34 : 4*34 = 약 400ms
6. transfer_value = 4*(연결된 디바이스 수)*34
   = 400ms*연결된 디바이스 수
7. Master Device의 송신 간격을 적용하지 않으면 데이터 송/수신 시 Data Loss 발생 우려 있음

12. FBD755MC_gcc V0.2 스텝 설명 요약

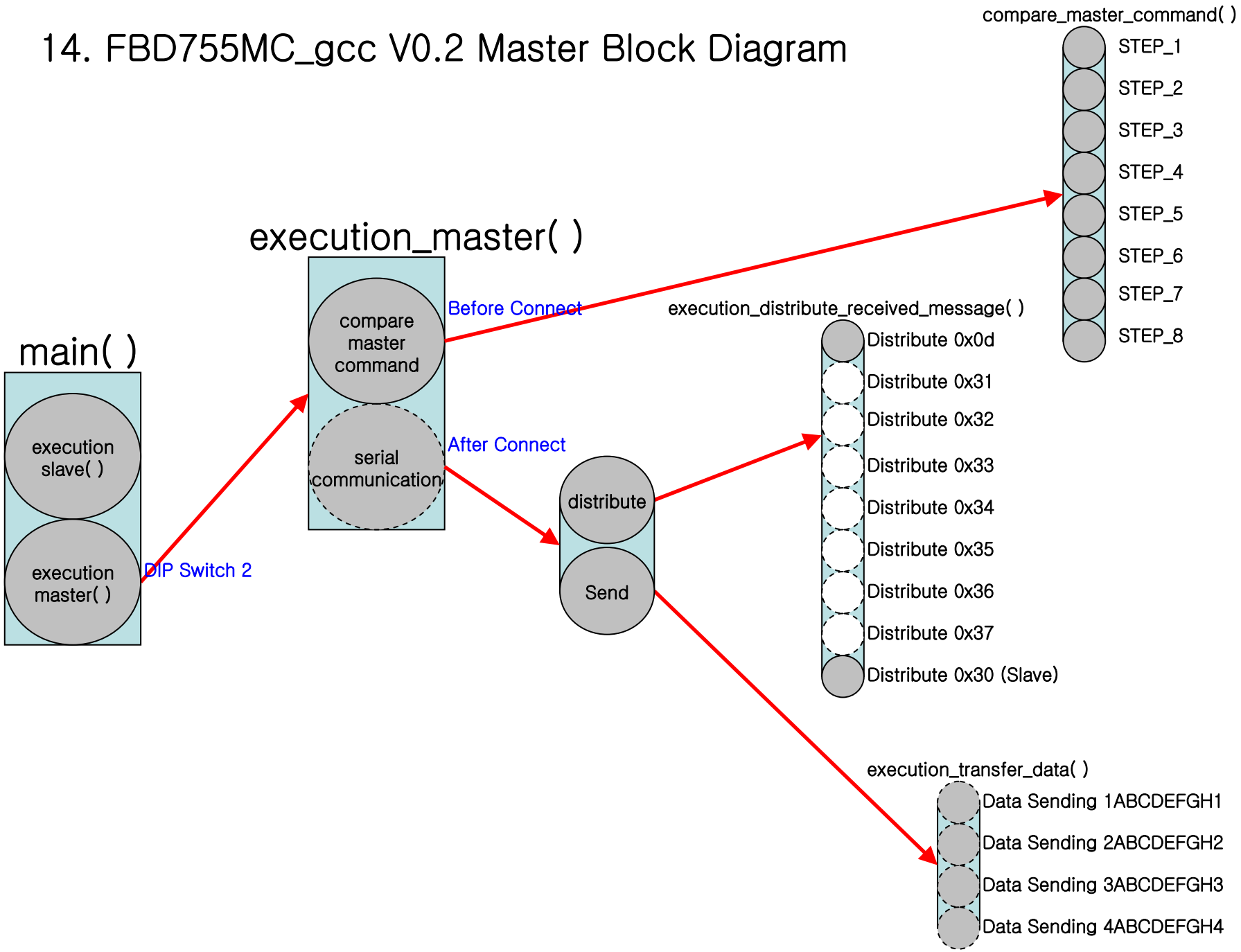
※ FBD755MC_gcc V0.1의 기본적인 사항은 FBD755MC_gcc V0.2 사용자 가이드에서 생략함

1. FB755AS를 Slave로 설정
   - OP_MODE1 설정
   - 연결 Device 수 설정
   - 버퍼 사이즈 10 설정
2. 설정된 수만큼의 Master Device가 연결되기를 기다림
3. 설정된 수만큼의 Master Device가 연결되면 데이터 송/수신 및 메시지 구분 시작
   - 수신 받은 데이터가 10바이트인 경우 동작
   - 구분 문자가 “0x0d”이면 상태 메시지 처리 (나머지 문자 수신 받은 후 처리)
     - ✓ 수신 데이터 형태 : “₩r₩nCONNECT 001122334455₩₩r₩n” (₩r 이 HEX 값으로 0x0d에 해당함)
     - ✓ 수신 데이터 형태 : “₩r₩nDISCONNECT 001122334455₩r₩n” (₩r 이 HEX 값으로 0x0d에 해당함)
   - 구분 문자가 “0x31”이면 1번 Master 데이터로 처리
     - ✓ 수신 데이터 형태 : “1ABCDEFGH1” (1 이 HEX 값으로 0x31에 해당함)
   - 구분 문자가 “0x32”이면 2번 Master 데이터로 처리
     - ✓ 수신 데이터 형태 : “2ABCDEFGH2” (2 가 HEX 값으로 0x32에 해당함)
   - 데이터의 송신은 Switch를 이용
   - Switch 1이 눌린 경우, STREAM 채널을 1번으로 변경하고, 1번 Master로 “0ABCDEFGH0” 데이터 송신(10byte)
   - Switch 3이 눌린 경우, STREAM 채널을 3번으로 변경하고, 3번 Master로 “0ABCDEFGH0” 데이터 송신
4. FBD755MC_gcc V0.2 인 경우 Master에서도 수신 데이터를 상태 메시지와 Slave에서 송신한 데이터로 구분함

13. FBD755MC_gcc V0.2 Slave Block Diagram



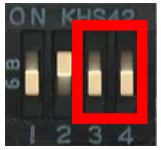
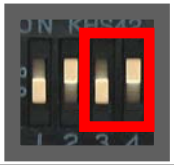
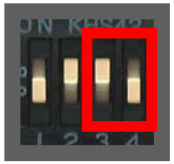
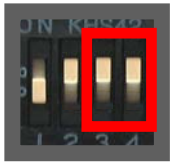
14. FBD755MC_gcc V0.2 Master Block Diagram



15. execution_distribute_received_message( ) Description

구분 문자	Description 1th	Description
0x0d	상태 메시지 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x0d인 경우, 상태 메시지로 판단 • FB755AS에서 발생될 수 있는 기본 상태 메시지 – ₩r₩nCONNECT 001122334455₩r₩n – ₩r₩nDISCONNECT 001122334455₩r₩n • 10바이트 단위를 기본으로 하기 때문에 나머지 바이트를 계산해서 읽어옴
0x30	0번 Slave Data 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x30인 경우, Slave에서 송신한 데이터로 판단 – Slave에서 Master로 데이터를 전송 시, “0ABCDEFGH0” 10 바이트 데이터 송신 – Slave Data의 구분 문자로 “0” 사용 • 10바이트 단위의 데이터 송/수신 기본 사항으로 정의
0x31	1번 Master Data 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x31인 경우, 1번 Master에서 송신한 데이터로 판단 – 1번 Master에서 Slave로 데이터를 전송 시, “1ABCDEFGH1” 10 바이트 데이터 송신 • 10바이트 단위의 데이터 송/수신 기본 사항으로 정의
0x32	2번 Master Data 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x32인 경우, 2번 Master에서 송신한 데이터로 판단 – 2번 Master에서 Slave로 데이터를 전송 시, “2ABCDEFGH2” 10 바이트 데이터 송신 • 10바이트 단위의 데이터 송/수신 기본 사항으로 정의
0x33	3번 Master Data 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x33인 경우, 3번 Master에서 송신한 데이터로 판단 – 3번 Master에서 Slave로 데이터를 전송 시, “3ABCDEFGH3” 10 바이트 데이터 송신 • 10바이트 단위의 데이터 송/수신 기본 사항으로 정의
0x34	4번 Master Data 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x34인 경우, 4번 Master에서 송신한 데이터로 판단 – 4번 Master에서 Slave로 데이터를 전송 시, “4ABCDEFGH4” 10 바이트 데이터 송신 • 10바이트 단위의 데이터 송/수신 기본 사항으로 정의
0x35	5번 Master Data 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x35인 경우, 5번 Master에서 송신한 데이터로 판단 – 5번 Master에서 Slave로 데이터를 전송 시, “5ABCDEFGH5” 10 바이트 데이터 송신 • 10바이트 단위의 데이터 송/수신 기본 사항으로 정의
0x36	6번 Master Data 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x36인 경우, 6번 Master에서 송신한 데이터로 판단 – 6번 Master에서 Slave로 데이터를 전송 시, “6ABCDEFGH6” 10 바이트 데이터 송신 • 10바이트 단위의 데이터 송/수신 기본 사항으로 정의
0x37	7번 Master Data 구분 문자	• 수신 받은 데이터의 첫 바이트가 0x37인 경우, 7번 Master에서 송신한 데이터로 판단 – 7번 Master에서 Slave로 데이터를 전송 시, “7ABCDEFGH7” 10 바이트 데이터 송신 • 10바이트 단위의 데이터 송/수신 기본 사항으로 정의

16. execution_transfer_data( ) Description

Master	딥 스위치	Description 1th	Description 2th
	DIP_2_2	데이터 송신 구분 문자를 “1”로 설정. Master 운영하면서 적용됨.	• 딥 스위치 3번 OFF, 4번 OFF된 경우 – 데이터 송신 시 구분 문자로 “1” 첨부 – Master에서 Slave로 데이터 송신 시, “1ABCDEFGH1” 송신
	DIP_2_4	데이터 송신 구분 문자를 “2”로 설정. Master 운영하면서 적용됨.	• 딥 스위치 3번 OFF, 4번 ON된 경우 – 데이터 송신 시 구분 문자로 “2” 첨부 – Master에서 Slave로 데이터 송신 시, “2ABCDEFGH2” 송신
	DIP_2_3	데이터 송신 구분 문자를 “3”로 설정. Master 운영하면서 적용됨.	• 딥 스위치 3번 ON, 4번 OFF된 경우 – 데이터 송신 시 구분 문자로 “3” 첨부 – Master에서 Slave로 데이터 송신 시, “3ABCDEFGH3” 송신
	DIP_2_3_4	데이터 송신 구분 문자를 “4”로 설정. Master 운영하면서 적용됨.	• 딥 스위치 3번 ON, 4번 ON된 경우 – 데이터 송신 시 구분 문자로 “4” 첨부 – Master에서 Slave로 데이터 송신 시, “4ABCDEFGH4” 송신

17. main( ) Source

```
int main(void)
{
    init_port();
    Init_UART1(BAUD_9600);
    Init_UART0(BAUD_9600);
    init_timer0();
    sei();

    read_avr_port = 0;
    lf_check_flag = LF_CHECK_NON;
    status_target = TARGET_NOT_DETECT;
    //FB755AX
    connect_count = 0x30;
    //
    wait_1ms(10);
    DispStr1("\r\n\r\nFBD755MC_gcc START");
    while(1)
    {
        read_avr_port = read_port_value(READ_DIP)&0x30;
        if(read_avr_port == DIP_1)
            execution_slave();
        else if(read_avr_port == DIP_2)
            execution_master();
        read_avr_port = read_port_value(READ_DIP)&0x80;
        if(read_avr_port == DIP_4)
            get_bluetooth_address();
    }
} ? end main ?
```

< 시스템 초기화 >

1. 포트 초기화
2. 통신 속도 초기화
3. 타이머 초기화

< Flag 초기화 >

1. 포트 저장 변수 초기화
2. LF Flag 초기화
3. 타겟 검색 Flag 초기화
4. 연결된 Device count = 0 (=0x30) 설정

< Master/Slave 설정 및 수행 >

1. 딥 스위치 Read
2. 1 번 딥 스위치가 ON 이면 슬레이브 루틴 진행
   - 딥 스위치 1번과 2번만 판단하기 위해 &0x30 (=0011 0000) 부분 추가
3. 2 번 딥 스위치가 ON 이면 마스터 루틴 진행
   - 딥 스위치 1번과 2번만 판단하기 위해 &0x30 (=0011 0000) 부분 추가
4. 4 번 딥 스위치가 ON이면 Address 읽기 루틴 진행
   - 딥 스위치 4번만 판단하기 위해 &0x80 (=1000 0000) 부분 추가

18. execution_slave( ) Source

```
void execution_slave(void)
{
    unsigned int i;

    status_sequence = BEFORE_CONNECT;
    step_slave_bt = STEP_1;
    old_switch_value = 0;
    stream_status = ERROR;
    init_uart0_read_data();
    write_port_value(WRITE_S_CONTROL_HIGH);
    while(1)
    {
        if(status_sequence == BEFORE_CONNECT)
        {
            uart0_length = Check_Rx_Buf0();
            if(uart0_length>0)
            {
                if(lf_check_flag == 2)
                {
                    uart0_length = Check_Rx_Buf0();
                    for(i = 0;i<uart0_length;i++)
                        uart0_read_data[i] = GetChar0();
                    compare_slave_command();
                }
            }
        }
        else if(status_sequence == AFTER_CONNECT)
        {
            lf_check_flag = LF_CHECK_NON;
            execution_distribute_received_message();
            execution_stream_control();
        }
    } ? end while 1 ?
} ? end execution_slave ?
```

< 슬레이브 루틴 진행 >

1. Bluetooth 연결 전 상태 설정
2. STEP_1 설정
3. 이전 스위치 값 0으로 설정
4. 스트림 상태 값 ERROR로 설정
5. 비교 데이터 저장 버퍼 초기화
6. S-CONTROL HIGH 출력

< Bluetooth 연결 전 진행 루틴 >

1. Bluetooth로부터 데이터 수신 시 동작
2. Bluetooth로부터 수신 받은 데이터에 0x0a가 2개인 경우 동작
3. Bluetooth로부터 수신 받은 데이터를 비교 데이터 저장 버퍼에 저장
4. Bluetooth 슬레이브 메시지 비교 루틴 진행

< Bluetooth 연결 후 진행 루틴 >

1. 수신 데이터 구분 루틴 진행
2. 스트림 제어 함수 호출 (데이터 송신 포함)

19. execution_master( ) Source

```
void execution_master(void)
{
    unsigned int i;

    //Write Slave Bluetooth Device Address
    memcpy(target_data,"00025b00a5a5",12);
    //
    status_sequence = BEFORE_CONNECT;
    step_master_bt = STEP_1;
    init_uart0_read_data();
    while(1)
    {
        if(status_sequence == BEFORE_CONNECT)
        {
            uart0_length = Check_Rx_Buf0();
            if(uart0_length>0)
            {
                if(lf_check_flag == 2)
                {
                    uart0_length = Check_Rx_Buf0();
                    for(i = 0;i<uart0_length;i++)
                        uart0_read_data[i] = GetChar0();
                    compare_master_command();
                }
            }
        }
        else if(status_sequence == AFTER_CONNECT)
        {
            lf_check_flag = LF_CHECK_NON;
            execution_distribute_received_message();
            execution_transfer_data();
        }
    } ? end while 1 ?
} ? end execution_master ?
```

< 슬레이브 어드레스 저장 >

1. 연결할 Bluetooth Slave Device의 어드레스를 입력
2. 1:N이 지원되는 FB755AS의 어드레스를 입력

< 마스터 루틴 진행 >

1. Bluetooth 연결 전 상태 설정
2. STEP_1 설정
3. 비교 데이터 저장 버퍼 초기화

< Bluetooth 연결 전 진행 루틴 >

1. Bluetooth로부터 데이터 수신 시 동작
2. Bluetooth로부터 수신 받은 데이터에 0x0a가 2개인 경우 동작
3. Bluetooth로부터 수신 받은 데이터를 비교 데이터 저장 버퍼에 저장
4. Bluetooth 마스터 메시지 비교 루틴 진행

< Bluetooth 연결 후 진행 루틴 >

1. 수신 데이터 구분 루틴 진행
2. 데이터 송신 함수 호출

20. compare_slave_command( ) Source – 1th

```
void compare_slave_command(void)
{
    unsigned int i;

    switch(step_slave_bt)
    {
        case STEP_1:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nBTWIN Slave mode start\r\n",&uart0_read_data[uart0_length-26],26))
            {
                step_slave_bt = STEP_3;
                DispStr1("[STEP_1 : START MSG RECEIVED]");
            }
            else if(!memcmp("\r\nBTWIN Master mode start\r\n",&uart0_read_data[uart0_length-27],27))
            {
                step_slave_bt++;
                DispStr1("[STEP_1 : START MSG RECEIVED]");
            }
            else
                execution_error();
            break;
        case STEP_2:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
            {
                step_slave_bt++;
                DispStr1("[STEP_2 : OK MSG RECEIVED]");
                DispStr1("[       CHANGE ROLE]");
                DispStr0("at+btrole=s\r");
            }
            else
                execution_error();
            break;
        case STEP_3:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
            {
                step_slave_bt++;
                DispStr1("[STEP_3 : OK MSG RECEIVED]");
                DispStr1("[       SET OPMODE 1]");
                wait_1ms(100);
                DispStr0("at+btopmode,1\r");
            }
            else
                execution_error();
            break;
```

< STEP_1 >

1. Bluetooth로부터 “₩r₩nBTWIN Slave mode start₩r₩n” 메시지를 수신한 경우 STEP_3 설정
2. Bluetooth로부터 “₩r₩nBTWIN Mater mode start₩r₩n” 메시지를 수신한 경우 STEP_2 설정
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

21. compare_slave_command( ) Source – 2th

```
void compare_slave_command(void)
{
    unsigned int i;

    switch(step_slave_bt)
    {
        case STEP_1:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nBTWIN Slave mode start\r\n",&uart0_read_data[uart0_length-26],26))
            {
                step_slave_bt = STEP_3;
                DispStr1("[STEP_1 : START MSG RECEIVED]");
            }
            else if(!memcmp("\r\nBTWIN Master mode start\r\n",&uart0_read_data[uart0_length-27],27))
            {
                step_slave_bt++;
                DispStr1("[STEP_1 : START MSG RECEIVED]");
            }
            else
                execution_error();
            break;
        case STEP_2:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
            {
                step_slave_bt++;
                DispStr1("[STEP_2 : OK MSG RECEIVED]");
                DispStr1("[       CHANGE ROLE]");
                DispStr0("at+btrole=s\r");
            }
            else
                execution_error();
            break;
        case STEP_3:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
            {
                step_slave_bt++;
                DispStr1("[STEP_3 : OK MSG RECEIVED]");
                DispStr1("[       SET OPMODE 1]");
                wait_1ms(100);
                DispStr0("at+btopmode,1\r");
            }
            else
                execution_error();
            break;
```

< STEP_2 : Bluetooth가 마스터로 시작된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지를 수신한 경우 STEP_3 설정
2. Bluetooth Device를 슬레이브로 변경
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_3 : Bluetooth가 슬레이브로 시작된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지를 수신한 경우 STEP_4 설정
2. Bluetooth Device OP_MODE1 설정
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

22. compare_slave_command( ) Source – 3th

```
case STEP_4:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
    {
        if(CONNECT_DEV_NUM > 0x37)
        {
            DispStr1("[ERROR, CHECK CONNECT_DEV_NUM]");
            DispStr1("[       MAX CONNECT_DEV_NUM : 7]");
            step_slave_bt = 50;
        }
        else
        {
            step_slave_bt++;
            DispStr1("[STEP_4 : OK MSG RECEIVED]");
            DispStr1_line("[       SET DEV NUM : ");
            PutChar1(CONNECT_DEV_NUM);
            DispStr1("]");
            wait_1ms(100);
            DispStr0("at+btdev=");
            PutChar0(CONNECT_DEV_NUM);
            PutChar0('\r');
        }
    } ? end if !memcmp("\r\nOK\r\n",... ?
    else
        execution_error();
    break;
case STEP_5:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
    {
        step_slave_bt++;
        DispStr1("[STEP_5 : OK MSG RECEIVED]");
        DispStr1("[       SET BUF SIZE : 10 BYTE]");
        wait_1ms(100);
        DispStr0("at+btbuff=10\r");
    }
    else
        execution_error();
    break;
```

< STEP_4 : OP_MODE1로 설정된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지 수신한 경우 동작
2. 연결 디바이스 수가 7보가 큰 경우 STEP을 50으로 설정 (일종의 ERROR 처리)
3. 연결 디바이스 수가 정상인 경우, STEP_5 설정
4. 연결 디바이스 수 설정 (CONNECT_DEV_NUM)
5. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_5 : 연결 디바이스 수가 설정된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지를 수신한 경우 STEP_6 설정
2. 데이터 송/수신 버퍼 사이즈 10으로 설정
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

23. compare_slave_command( ) Source – 4th

```
case STEP_6:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
    {
        step_slave_bt++;
        DispStr1("[STEP_6 : OK MSG RECEIVED]");
        DispStr1("[       RESET BLUETOOTH]");
        DispStr0("atz\r");
    }
    else
        execution_error();
    break;
case STEP_7:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nBTWIN Slave mode start\r\n",&uart0_read_data[uart0_length-26],26))
    {
        step_slave_bt++;
        DispStr1("[STEP_7 : START MSG RECEIVED]");
    }
    else
        execution_error();
    break;
case STEP_8:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
    {
        step_slave_bt++;
        DispStr1("[STEP_8 : OK MSG RECEIVED]");
        DispStr1("[       EXECUTON SCAN]");
        wait_1ms(100);
        DispStr0("at+btscan\r");
    }
    else
        execution_error();
    break;
```

< STEP_6 : 데이터 송/수신 버퍼 사이즈가 설정된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지 수신한 경우, STEP_7 설정
2. 스텝 별 설정 값 적용하기 위한 Device Reset
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_7 : Bluetooth가 슬레이브로 시작된 경우 >

1. Bluetooth로부터 “₩r₩nBTWIN Slave mode start₩r₩n”메시지를 수신한 경우 STEP_8 설정
2. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_8 : Bluetooth가 슬레이브로 시작된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지를 수신한 경우 STEP_9 설정
2. Bluetooth Device를 검색 대기 상태로 설정
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

24. compare_slave_command( ) Source – 5th

```
case STEP_9:
    lf_check_flag = LF_CHECK_NON;
    if(! memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
    {
        step_slave_bt++;
        DispStr1("[STEP_9 : OK MSG RECEIVED]");
        DispStr1("[       WAIT CONNECT MSG]");
    }
    else
        execution_error();
    break;
case STEP_10:
    lf_check_flag = LF_CHECK_NON;
    if(! memcmp("\r\nCONNECT",&uart0_read_data[uart0_length-24],9))
    {
        connect_count++;
        if(connect_count == CONNECT_DEV_NUM)
        {
            step_slave_bt++;
            status_sequence = AFTER_CONNECT;
            write_port_value(WRITE_S_CONTROL_LOW);
            DispStr1_line("[STEP_11 : ");
            for(i = 2;i < uart0_length-2;i++)
                PutChar1(uart0_read_data[i]);
            DispStr1("]");
            DispStr1("[       COMMUNICATION START]");
        }
        else
        {
            step_slave_bt = STEP_10;
            DispStr1_line("[STEP_10 : ");
            for(i = 2;i < uart0_length-2;i++)
                PutChar1(uart0_read_data[i]);
            DispStr1("]");
            DispStr1("[       WAIT NEXT CONNECT MSG]");
        }
    } ? end if ! memcmp("\r\nCONNECT"... ?
    else
        execution_error();
    break;
```

< STEP_9 : Bluetooth가 검색 대기 상태인 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지 수신한 경우, STEP_10 설정
2. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_10 : CONNECT 메시지 대기 >

1. Bluetooth로부터 “₩r₩nCONNECT”메시지를 수신한 경우 동작
2. 연결된 디바이스 수 카운트 (connect_count)
3. 연결된 디바이스 수가 설정한 카운트 수와 같은 경우,
   - STEP_11 설정 (STEP_11은 아무런 동작이 없고, 비교 루틴을 나가는 동작을 하기 위해 설정하는 것임)
   - Bluetooth Device를 연결 상태로 설정
   - S-CONTROL 포트를 LOW로 설정
   - 연결된 디바이스의 어드레스 출력
   - 데이터 송/수신 루틴 시작
4. 연결된 디바이스 수가 설정한 카운트 수와 다른 경우,
   - STEP_10 설정
   - 연결된 디바이스의 어드레스 출력
   - 다음 CONNECT 메시지대기 진행 (FB755AS는 SCAN동작을 1회 진행시키면 설정된 디바이스 수만큼 연결 될 때까지 SCAN 동작을 자동으로 수행)
5. 비교 데이터가 틀린 경우 execution_error()루틴 진행

25. main.h

```
//FB755AX
#define WRITE_S_CONTROL_HIGH   0x05
#define WRITE_S_CONTROL_LOW    0x06
//#define CONNECT_DEV_NUM      0x31
//#define CONNECT_DEV_NUM      0x32
//#define CONNECT_DEV_NUM      0x33
#define CONNECT_DEV_NUM        0x34
//#define CONNECT_DEV_NUM      0x35
//#define CONNECT_DEV_NUM      0x36
//#define CONNECT_DEV_NUM      0x37

//distribute data
#define DATA_OK          0x01
#define DATA_ERROR       0x02
#define STATUS_CON       0x03
#define STATUS_DISCIN    0x04
//
```

```
//FB755AX
unsigned char connect_count;
unsigned char old_switch_value;
unsigned char stream_status;
unsigned char s_status_value;
unsigned int loop_count_1,loop_count_2;

//distribute data
unsigned char message_value;

//transfer data
unsigned int transfer_interval;
unsigned int transfer_value;
//
```

< FB755AS의 연결 디바이스 수 설정을 위한 값 >

1. main.h 파일에 연결 디바이스 수 설정 값 Define
2. #define CONNECT_DEV_NUM 0x34
   - 연결 디바이스 수를 4개로 설정
   - 헥사 값 0x34는 10진수 캐릭터 값 4와 같음
3. 연결 디바이스 수 설정 값을 변경하기 위해서는 주석 처리된 부분을 변경
4. 연결 디바이스 수 설정 최대 값은 0x37 (10진수 캐릭터 값으로 7)
5. 수신 데이터 구분을 위한 값 정의
6. 데이터 송신에 사용할 변수 정의

26. execution_stream_control( ) Source

```
void execution_stream_control(void)
{
    read_avr_port = read_port_value(READ_SWITCH);
    if(read_avr_port == SWITCH_1||read_avr_port == SWITCH_2||
        read_avr_port == SWITCH_3||read_avr_port == SWITCH_4)
    {
        old_switch_value = read_avr_port;
        write_port_value(WRITE_S_CONTROL_HIGH);
        execution_check_s_status_high();
        if(s_status_value == OK)
        {
            switch(old_switch_value)
            {
                case SWITCH_1:
                    DispStr0("ato1\r");
                    DispStr1("[CHANGE STREAM CONNECT 1]");
                    break;
                case SWITCH_2:
                    DispStr0("ato2\r");
                    DispStr1("[CHANGE STREAM CONNECT 2]");
                    break;
                case SWITCH_3:
                    DispStr0("ato3\r");
                    DispStr1("[CHANGE STREAM CONNECT 3]");
                    break;
                case SWITCH_4:
                    DispStr0("ato4\r");
                    DispStr1("[CHANGE STREAM CONNECT 4]");
                    break;
            }
            execution_check_s_status_low();
            if(s_status_value == ERROR)
            {
                write_port_value(WRITE_S_CONTROL_LOW);
                stream_status = ERROR;
                DispStr1("[STREAM STATUS ERROR]");
            }
            else
            {
                write_port_value(WRITE_S_CONTROL_LOW);
                stream_status = OK;
                DispStr1("[STREAM STATUS OK]");
                DispStr1("SENDING DATA : 0ABCDEFGH0");
                DispStr0_line("0ABCDEFGH0");
            }
        } ? end if s_status_value==OK ?
        else
        {
            write_port_value(WRITE_S_CONTROL_LOW);
            stream_status = ERROR;
            DispStr1("[STREAM STATUS ERROR]");
        }
    } ? end if read_avr_port==SWITCH... ?
} ? end execution_stream_control ?
```

< 스위치를 읽어서 STREAM 채널 변경 진행 >

1. 스위치가 눌렸는지 검사
2. 스위치가 눌렸는지 검사한 값을 old_switch_value에 저장
3. 눌린 스위치가 SWITHC_1 / SWITCH_2 / SWITCH_3 / SWITCH_4인 경우, STREAM 채널 변경 적용
4. 스위치가 눌리지 않은 경우 이전 루틴으로 복귀

< STREAM 채널 변경 & 데이터 송신 >

1. 현재 눌린 스위치 저장 (old_switch_value)
2. S-CONTROL에 HIGH 출력
3. S-STATUS가 HIGH가 되는지 검사 (execution_check_s_status_high())
4. s_status_value가 OK인 경우 STREAM 변경을 위한 명령어 출력
   - 스위치 1 이 눌린 경우 “ato1₩r” Command FB755AS에 전달
   - 스위치 4 가 눌린 경우 “ato4₩r” Command FB755AS에 전달
5. S-STATUS가 LOW가 되는지 검사 (execution_check_s_status_low())
6. s_status_value가 ERROR인 경우
   - S-CONTROL LOW 설정
   - STREAM 채널 변경 실패 설정
7. s_status_value가 OK인 경우
   - S-CONTROL LOW 설정
   - STREAM 채널 변경 완료 설정
   - 데이터 송신 (“0ABCDEFGH0” 10바이트 송신)

27. execution_check_s_status_high( ) Source

```
void execution_check_s_status_high(void)
{
    loop_count_1 = 0;
    loop_count_2 = 0;
    s_status_value = OK;
    for(;;)
    {
        read_avr_port = read_port_value(READ_S_STATUS);
        if(read_avr_port == S_STATUS_HIGH)break;
        loop_count_1++;
        if(loop_count_1 > 1000)
        {
            loop_count_1 = 0;
            loop_count_2++;
        }
        if(loop_count_2 > 200)
        {
            loop_count_1 = 0;
            loop_count_2 = 0;
            s_status_value = ERROR;
            break;
        }
    }
} ? end execution_check_s_status_high ?
```

< FB755AS의 S-STATUS에서 HIGH가 출력되는지 조사하는 함수 >

1. MICOM에서 FB755AS의 S-CONTROL에 HIGH 신호를 출력하는 경우 FB755AS의 S-STATUS 포트가 HIGH로 변경됨
2. FB755AS의 STREAM 채널이 연결되지 않은 상태에서는 기본적으로 S-STATUS 포트에서 HIGH 신호 출력됨
3. loop_count_1과 loop_count_2를 사용하여 무한루프 방지
4. s_status_value를 OK로 설정
5. FOR문을 이용하여 FB755AS의 S-STATUS 포트 감시
6. S_STATUS_HIGH 인 경우 FOR문에서 빠져 나옴
   (s_status_value가 OK인 상태로 이전 루틴 복귀)
7. FOR문을 운영하는 동안 loop_count_1 증가
8. loop_count_1이 1000보다 큰 경우
   - loop_count_1을 0으로 설정
   - loop_count_2를 1 증가
9. loop_count_2가 200보다 큰 경우
   - loop_count_1을 0으로 설정
   - loop_count_2를 0으로 설정
   - s_status_value를 ERROR로 설정
   - FOR문을 빠져 나감
     (s_status_value가 ERROR인 상태로 이전 루틴 복귀)
10. loop_count_1과 loop_count_2를 이용하여 약 2초 동안 FB755AS의 S-STATUS 포트가 HIGH 로 변경하지 않으면 FOR문을 빠져 나감
    (s_status_value가 ERROR인 상태로 이전 루틴 복귀)

28. execution_check_s_status_low( ) Source

```
void execution_check_s_status_low(void)
{
    loop_count_1 = 0;
    loop_count_2 = 0;
    s_status_value = OK;
    for(;;)
    {
        read_avr_port = read_port_value(READ_S_STATUS);
        if(read_avr_port == S_STATUS_LOW)break;
        loop_count_1++;
        if(loop_count_1 > 1000)
        {
            loop_count_1 = 0;
            loop_count_2++;
        }
        if(loop_count_2 > 200)
        {
            loop_count_1 = 0;
            loop_count_2 = 0;
            s_status_value = ERROR;
            break;
        }
    }
} ? end execution_check_s_status_low ?
```

< FB755AS의 S-STATUS에서 LOW가 출력되는지 조사하는 함수 >

1. MICOM에서 FB755AS로 STREAM 채널 변경에 대한 명령어(ATO1)를 전송한 후, FB755AS가 STREAM 채널 변경을 정상적으로 수행하면 FB755AS의 S-STATUS포트가 LOW로 변경됨
2. FB755AS의 STREAM 채널이 연결되지 않은 상태이거나 변경되지 않는 경우는 기본적으로 S-STATUS 포트에서 HIGH 신호 출력됨
3. loop_count_1과 loop_count_2를 사용하여 무한루프 방지
4. s_status_value를 OK로 설정
5. FOR문을 이용하여 FB755AS의 S-STATUS 포트 감시
6. S_STATUS_LOW 인 경우 FOR문에서 빠져 나옴
   (s_status_value가 OK인 상태로 이전 루틴 복귀)
7. FOR문을 운영하는 동안 loop_count_1 증가
8. loop_count_1이 1000보다 큰 경우
   - loop_count_1을 0으로 설정
   - loop_count_2를 1 증가
9. loop_count_2가 200보다 큰 경우
   - loop_count_1을 0으로 설정
   - loop_count_2를 0으로 설정
   - s_status_value를 ERROR로 설정
   - FOR문을 빠져 나감
     (s_status_value가 ERROR인 상태로 이전 루틴 복귀)
10. loop_count_1과 loop_count_2를 이용하여 약 2초 동안 FB755AS의 S-STATUS 포트가 LOW 로 변경하지 않으면 FOR문을 빠져 나감
    (s_status_value가 ERROR인 상태로 이전 루틴 복귀)

29. execution_distribute_received_message( ) Source – 1th

```
void execution_distribute_received_message(void)
{
    unsigned int i;

    message_value = DATA_OK;
    init_uart0_read_data();
    uart0_length = 0;
    uart0_length = Check_Rx_Buf0();
    if(uart0_length>9)
    {
        uart0_read_data[0] = GetChar0();
        switch(uart0_read_data[0])
        {
            case 0x0d:
                message_value = STATUS_CON;
                execution_get_received_message();
                if(uart0_read_data[2] == 'C')
                {
                    for(i = 10;i<24;i++)uart0_read_data[i] = GetChar0();
                    if(uart0_read_data[23] == 0x0a)
                        DispStr1_line("STATUS MESSAGE : ");
                    else
                        DispStr1_line("CON ERROR : ");
                }
                else if(uart0_read_data[2] == 'D')
                {
                    message_value = STATUS_DISCIN;
                    for(i = 10;i<27;i++)uart0_read_data[i] = GetChar0();
                    if(uart0_read_data[26] == 0x0a)
                        DispStr1_line("STATUS MESSAGE : ");
                    else
                        DispStr1_line("DISCON ERROR : ");
                }
                break;
            case 0x30:
                execution_get_received_message();
                if(uart0_read_data[9] == 0x30)
                    DispStr1_line("SLAVE 0 DATA : ");
                else
                    DispStr1_line("SLAVE 0 ERROR : ");
                break;
```

< FB755AS에서 수신 받은 데이터를 “상태 메시지”와 “Master로부터 수신 받은 데이터”로 구분하는 함수 >

1. 데이터에 포함되어 있는 구분 문자를 이용하여 상태 메시지와 Master 로부터 수신 받은 데이터를 구분함
2. 상태 메시지 구분 문자로 0x0d (= ₩r) 사용
3. Master 데이터 구분 문자로 0x31 ~ 0x37 (= 1 ~ 7) 사용
4. FB755AS로부터 수신 받은 데이터가 9 바이트 이상인 경우 동작 (10바이트 데이터 송신으로 정의되어 있음)
5. 수신 받은 데이터의 첫 바이트를 읽어옴
6. 수신 받은 데이터의 첫 바이트가 0x0d 인 경우 상태 메시지로 구분
   - 상태메시지는 “CONNECT”와 “DISCONNECT”로 구분
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 3번째 데이터가 “C”인 경우 CONNECT 메시지로 구분
   - 3번째 데이터가 “D”인 경우 DISCONNECT 메시지로 구분
   - 상태 메시지로 구분한 후 나머지 바이트 읽어옴
   - 마지막 바이트가 0x0a (= ₩n)인 경우 상태 메시지로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리
7. 수신 받은 데이터의 첫 바이트가 0x30인 경우 Slave 데이터로 구분
   - 송/수신 데이터는 10바이트로 정의되어 있음
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 마지막 바이트가 0x30인 경우 Slave 데이터로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리

30. execution_distribute_received_message( ) Source – 2th

```
case 0x31:
    execution_get_received_message();
    if(uart0_read_data[9] == 0x31)
        DispStr1_line("MASTER 1 DATA : ");
    else
        DispStr1_line("MASTER 1 ERROR : ");
    break;
case 0x32:
    execution_get_received_message();
    if(uart0_read_data[9] == 0x32)
        DispStr1_line("MASTER 2 DATA : ");
    else
        DispStr1_line("MASTER 2 ERROR : ");
    break;
case 0x33:
    execution_get_received_message();
    if(uart0_read_data[9] == 0x33)
        DispStr1_line("MASTER 3 DATA : ");
    else
        DispStr1_line("MASTER 3 ERROR : ");
    break;
case 0x34:
    execution_get_received_message();
    if(uart0_read_data[9] == 0x34)
        DispStr1_line("MASTER 4 DATA : ");
    else
        DispStr1_line("MASTER 4 ERROR : ");
    break;
```

< FB755AS에서 수신 받은 데이터를 “상태 메시지”와 “Master로부터 수신 받은 데이터”로 구분하는 함수 >

1. 수신 받은 데이터의 첫 바이트가 0x31인 경우 1번 Master 데이터로 구분
   - 송/수신 데이터는 10바이트로 정의되어 있음
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 마지막 바이트가 0x31인 경우 1번 Master에서 송신한 데이터로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리
2. 수신 받은 데이터의 첫 바이트가 0x32인 경우 2번 Master 데이터로 구분
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 마지막 바이트가 0x32인 경우 2번 Master에서 송신한 데이터로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리
3. 수신 받은 데이터의 첫 바이트가 0x33인 경우 3번 Master 데이터로 구분
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 마지막 바이트가 0x33인 경우 3번 Master 에서 송신한 데이터로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리
4. 수신 받은 데이터의 첫 바이트가 0x34인 경우 4번 Master 데이터로 구분
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 마지막 바이트가 0x34인 경우 4번 Master에서 송신한 데이터로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리

31. execution_distribute_received_message( ) Source – 3th

```
    case 0x35:
        execution_get_received_message();
        if(uart0_read_data[9] == 0x35)
            DispStr1_line("MASTER 5 DATA : ");
        else
            DispStr1_line("MASTER 5 ERROR : ");
        break;
    case 0x36:
        execution_get_received_message();
        if(uart0_read_data[9] == 0x36)
            DispStr1_line("MASTER 6 DATA : ");
        else
            DispStr1_line("MASTER 6 ERROR : ");
        break;
    case 0x37:
        execution_get_received_message();
        if(uart0_read_data[9] == 0x37)
            DispStr1_line("MASTER 7 DATA : ");
        else
            DispStr1_line("MASTER 7 ERROR : ");
        break;
    default:
        message_value = DATA_ERROR;
        DispStr1_line("DATA ERROR : ");
        uart0_length = Check_Rx_Buf0();
        for(i = 0;i<uart0_length;i++)PutChar1(GetChar0());
        break;
} ? end switch uart0_read_data[0] ?
execution_put_received_message();
init_uart0_read_data();
```

< FB755AS에서 수신 받은 데이터를 “상태 메시지”와 “Master로부터 수신 받은 데이터”로 구분하는 함수 >

1. 수신 받은 데이터의 첫 바이트가 0x35인 경우 5번 Master 데이터로 구분
   - 송/수신 데이터는 10바이트로 정의되어 있음
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 마지막 바이트가 0x35인 경우 5번 Master에서 송신한 데이터로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리
2. 수신 받은 데이터의 첫 바이트가 0x36인 경우 6번 Master 데이터로 구분
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 마지막 바이트가 0x36인 경우 6번 Master에서 송신한 데이터로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리
3. 수신 받은 데이터의 첫 바이트가 0x37인 경우 7번 Master 데이터로 구분
   - 첫 바이트 조사 후, 9 바이트를 읽어옴
   - 마지막 바이트가 0x37인 경우 7번 Master 에서 송신한 데이터로 확인
   - 구분 문자가 틀린 경우 ERROR로 처리
4. 수신 받은 데이터의 첫 바이트가 정해진 구분 문자가 아닌 경우
   - ERROR로 처리
   - 수신 받은 데이터를 전부 출력
5. 구분에 사용했던 데이터를 구분 상태 값과 함께 시리얼로 출력

32. execution_get_received_message( ) & execution_put_received_message( ) Source

```
void execution_get_received_message(void)
{
    unsigned int i;

    if(message_value == STATUS_CON)
    {
        wait_1ms(150);
        for(i = 1;i<10;i++)uart0_read_data[i] = GetChar0();
    }
    else if(message_value == DATA_OK)
    {
        for(i = 1;i<10;i++)uart0_read_data[i] = GetChar0();
    }
}

void execution_put_received_message(void)
{
    unsigned int i;

    if(message_value == DATA_OK)
        for(i = 0;i < 10;i++)PutChar1(uart0_read_data[i]);
    else if(message_value == STATUS_CON)
        for(i = 2;i < 22;i++)PutChar1(uart0_read_data[i]);
    else if(message_value == STATUS_DISCON)
        for(i = 2;i < 25;i++)PutChar1(uart0_read_data[i]);
    DispStr1("");
}
```

< execution_get_received_message( ) >

1. 수신 받은 데이터의 첫 바이트 조사 후 나머지 바이트를 읽어오는 함수
2. 10바이트 기본 단위로 설정했기 때문에 10바이트를 읽어옴
3. 상태 메시지인 경우, 약간의 대기 시간을 기다린 후 읽어옴

< execution_put_received_message( ) >

1. 수신 받은 데이터의 구분이 끝난 이후, 데이터를 시리얼로 출력하는 함수
2. Master로부터 수신 받은 데이터인 경우 10바이트 기본 단위로 설정했기 때문에 10바이트를 출력함
3. 상태 메시지인 경우
   - CONNECT인 경우 : 데이터 구분에 사용한 2바이트(=₩r₩n)를 제외하고 나머지 20바이트를 출력함 (마지막 2바이트(=₩r₩n)도 제외)
   - DISCONNECT인 경우 : 데이터 구분에 사용한 2바이트(=₩r₩n)를 제외하고 나머지 23바이트를 출력함 (마지막 2바이트(=₩r₩n)도 제외)

33. compare_master_command( ) Source – 1th

```
void compare_master_command(void)
{
    unsigned int i;

    switch(step_master_bt)
    {
        case STEP_1:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nBTWIN Slave mode start\r\n",&uart0_read_data[uart0_length-26],26))
            {
                step_master_bt++;
                DispStr1("[STEP_1 : START MSG RECEIVED]");
            }
            else if(!memcmp("\r\nBTWIN Master mode start\r\n",&uart0_read_data[uart0_length-27],27))
            {
                step_master_bt = STEP_3;
                DispStr1("[STEP_1 : START MSG RECEIVED]");
            }
            else
                execution_error();
            break;
        case STEP_2:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
            {
                step_master_bt++;
                DispStr1("[STEP_2 : OK MSG RECEIVED]");
                DispStr1("[       CHANGE ROLE]");
                DispStr0("at+btrole=m\r");
            }
            else
                execution_error();
            break;
        case STEP_3:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
            {
                step_master_bt++;
                status_target = TARGET_DETECT;
                DispStr1("[STEP_3 : OK MSG RECEIVED]");
                DispStr1("[       SET OPMODE 0]");
                wait_1ms(100);
                DispStr0("at+btopmode,0\r");
            }
            else
                execution_error();
            break;
```

< STEP_1 >

1. Bluetooth로부터 “₩r₩nBTWIN Slave mode start₩r₩n” 메시지를 수신한 경우 STEP_2 설정
2. Bluetooth로부터 “₩r₩nBTWIN Mater mode start₩r₩n” 메시지를 수신한 경우 STEP_3 설정
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

34. compare_master_command( ) Source – 2th

```
void compare_master_command(void)
{
    unsigned int i;

    switch(step_master_bt)
    {
        case STEP_1:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nBTWIN Slave mode start\r\n",&uart0_read_data[uart0_length-26],26))
            {
                step_master_bt++;
                DispStr1("[STEP_1 : START MSG RECEIVED]");
            }
            else if(!memcmp("\r\nBTWIN Master mode start\r\n",&uart0_read_data[uart0_length-27],27))
            {
                step_master_bt = STEP_3;
                DispStr1("[STEP_1 : START MSG RECEIVED]");
            }
            else
                execution_error();
            break;
        case STEP_2:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
            {
                step_master_bt++;
                DispStr1("[STEP_2 : OK MSG RECEIVED]");
                DispStr1("[       CHANGE ROLE]");
                DispStr0("at+btrole=m\r");
            }
            else
                execution_error();
            break;
        case STEP_3:
            lf_check_flag = LF_CHECK_NON;
            if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
            {
                step_master_bt++;
                status_target = TARGET_DETECT;
                DispStr1("[STEP_3 : OK MSG RECEIVED]");
                DispStr1("[       SET OPMODE 0]");
                wait_1ms(100);
                DispStr0("at+btopmode,0\r");
            }
            else
                execution_error();
            break;
```

< STEP_2 : Bluetooth가 슬레이브로 시작된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지를 수신한 경우 STEP_3 설정
2. Bluetooth Device를 마스터로 변경
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_3 : Bluetooth가 마스터로 시작된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지를 수신한 경우 STEP_4 설정
2. Bluetooth Device OP_MODE0 설정
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

35. compare_master_command( ) Source – 3th

```
case STEP_4:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
    {
        step_master_bt++;
        DispStr1("[STEP_4 : OK MSG RECEIVED]");
        DispStr1("[       RESET BLUETOOTH]");
        DispStr0("atz\r");
    }
    else
        execution_error();
    break;
case STEP_5:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nBTWIN Master mode start\r\n",&uart0_read_data[uart0_length-27],27))
    {
        step_master_bt++;
        DispStr1("[STEP_5 : START MSG RECEIVED]");
    }
    else
        execution_error();
    break;
case STEP_6:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
    {
        step_master_bt++;
        status_target = TARGET_DETECT;
        DispStr1("[STEP_6 : OK MSG RECEIVED]");
        DispStr1_line("[       TRY TO CONNECT ");
        for(i = 0;i < 12;i++)
                PutChar1(target_data[i]);
        DispStr1("]");
        wait_1ms(100);
        DispStr0("atd");
        for(i = 0;i<12;i++)
            PutChar0(target_data[i]);
        PutChar0(0x0d);
    }
    else
        execution_error();
    break;
```

< STEP_4 : OP_MODE가 설정된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지를 수신한 경우 STEP_5 설정
2. 스텝 별 설정 값 적용하기 위한 Device Reset
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_5 : Bluetooth가 마스터로 시작된 경우 >

1. Bluetooth로부터 “₩r₩nBTWIN Master mode start₩r₩n”메시지를 수신한 경우 STEP_6 설정
2. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_6 : Bluetooth가 마스터로 시작된 경우 >

1. Bluetooth로부터 “₩r₩nOK”메시지를 수신한 경우 STEP_7 설정
2. 저장되어 있는 Slave Address를 이용하여 연결 시도
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

36. compare_master_command( ) Source – 4th

```
case STEP_7:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nOK\r\n",&uart0_read_data[uart0_length-6],6))
    {
        step_master_bt++;
        DispStr1("[STEP_7 : OK MSG RECEIVED]");
        DispStr1("[       WAIT CONNECT MSG]");
    }
    else
        execution_error();
    break;
case STEP_8:
    lf_check_flag = LF_CHECK_NON;
    if(!memcmp("\r\nCONNECT",&uart0_read_data[uart0_length-24],9))
    {
        step_master_bt++;
        status_sequence = AFTER_CONNECT;
//연결 디바이스 수가 1대 추가 되면 데이터 송신 간격이 400ms 추가됨(1대 : 400ms, 2대 : 800ms, 7대 : 2800ms)
        transfer_value = ((4*(CONNECT_DEV_NUM-0x30)))*34;
        DispStr1("[STEP_8 : COMMUNICATION START]");
    }
    else if(!memcmp("\r\nERROR\r\n",&uart0_read_data[uart0_length-9],9))
    {
        step_master_bt = STEP_7;
        status_target = TARGET_DETECT;
        DispStr1("[STEP_8 : ERROR MSG RECEIVED]");
        DispStr1_line("[       TRY TO RE-CONNECT ");
        for(i = 0;i < 12;i++)
                PutChar1(target_data[i]);
        DispStr1("]");
        wait_1ms(100);
        DispStr0("atd");
        for(i = 0;i<12;i++)
            PutChar0(target_data[i]);
        PutChar0(0x0d);
    }
    else
        execution_error();
    break;
```

< STEP_7 : 연결 시도가 진행된 경우 >

1. Bluetooth로부터 “₩r₩nOK₩r₩n”메시지를 수신한 경우 STEP_8 설정
2. 비교 데이터가 틀린 경우 execution_error()루틴 진행

< STEP_8 : 마스터와 슬레이브가 연결된 경우 >

1. Bluetooth로부터 “₩r₩nCONNECT”메시지를 수신한 경우 STEP_9 설정 (STEP_9는 아무런 동작이 없고, 비교 루틴을 나가는 동작을 하기 위해 설정하는 것임)
   - Bluetooth Device를 연결 상태로 설정
   - 데이터 송신 간격 계산 & 저장 (transfer_value)
2. Bluetooth로부터 “₩r₩nERROR₩r₩n”메시지를 수신한 경우 STEP_7 설정
   - 저장되어 있는 Slave Address를 이용하여 재연결 시도
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

37. execution_transfer_data( ) Source

```
void execution_transfer_data(void)
{
    read_avr_port = read_port_value(READ_SWITCH);
    if(read_avr_port == SWITCH_1||read_avr_port == SWITCH_2||
        read_avr_port == SWITCH_3||read_avr_port == SWITCH_4)
    {
        old_switch_value = read_avr_port;
    }
    if(old_switch_value == SWITCH_1)
    {
        if(transfer_interval > transfer_value)
        {
            transfer_interval = 0;
            read_avr_port = read_port_value(READ_DIP)&0xe0;
            if(read_avr_port == DIP_2_2)
            {
                DispStr0_line("1ABCDEFGH1");
                DispStr1("SENDING DATA : 1ABCDEFGH1");
            }
            else if(read_avr_port == DIP_2_4)
            {
                DispStr0_line("2ABCDEFGH2");
                DispStr1("SENDING DATA : 2ABCDEFGH2");
            }
            else if(read_avr_port == DIP_2_3)
            {
                DispStr0_line("3ABCDEFGH3");
                DispStr1("SENDING DATA : 3ABCDEFGH3");
            }
            else if(read_avr_port == DIP_2_3_4)
            {
                DispStr0_line("4ABCDEFGH4");
                DispStr1("SENDING DATA : 4ABCDEFGH4");
            }
        } ? end if transfer_interval>tra... ?
    } ? end if old_switch_value==SWI... ?
} ? end execution_transfer_data ?
```

< FBDx5xMC보드에서 FB755AS로 데이터를 입력하는 함수>

1. 스위치를 이용하여 데이터 입력을 시작함
   - 딥 스위치를 이용하여 데이터 구분 문자를 설정함
   - Master 보드의 Switch 1번이 눌린 경우, 구분 문자와 데이터가 FB755AS로 데이터 송신 간격 시간에 한번씩 입력되기 시작함
2. Master 보드의 Switch 1번 이외의 스위치를 누른 경우, 데이터 입력 정지
3. Switch 1이 눌린 경우
   - 데이터 송신 간격 값이 데이터 송신 설정 값보다 큰 경우 동작 (transfer_interval > transfer_value)
   - transfer_interval : timer0을 이용하여 증가 하는 변수
   - transfer_value : 연결 디바이스 수에 의해 설정되는 변수
   - 딥 스위치 값 읽기
   - 딥 스위치 3번 OFF, 4번 OFF인 경우 “1ABCDEFGH1” 데이터를 FB755AS로 입력
   - 딥 스위치 3번 OFF, 4번 ON인 경우 “2ABCDEFGH2” 데이터를 FB755AS로 입력
   - 딥 스위치 3번 ON, 4번 OFF인 경우 “3ABCDEFGH3” 데이터를 FB755AS로 입력
   - 딥 스위치 3번 ON, 4번 ON인 경우 “4ABCDEFGH4” 데이터를 FB755AS로 입력

38. Initialize Function – port, timer0

```
void init_port (void)
{
    DDRA  = 0x00;
    PORTA = 0x0f;

    DDRB  = 0xff;
    PORTB = 0xf0;

    DDRC  = 0x04;
    PORTC = 0x00;

    DDRD  = 0xff;
    PORTD = 0x00;

    DDRE  = 0xff;
    PORTE = 0x00;

    DDRF  = 0xff;
    PORTF = 0x00;
} ? end init_port ?
```

< ATMega128 포트 초기화 : 각 포트는 0 ~ 7 비트까지 8개의 포트로 구성 >

1. 사용하는 포트는 LED로 할당된 PORTB의 4~7 비트, 딥 스위치로 할당된 PORTA의 4~7 비트, 푸시 스위치로 할당된 PORTA의 0~3 비트, STREAM 채널 변경을 위해 할당된 (S-CONTROL / S-STATUS) PORTC의 1~2 비트
2. 딥 스위치 1번이 ON 되어 있으면 Slave로 동작, 딥 스위치 2번이 ON 되어 있으면 Master로 동작
3. Slave 동작 시 LED 1번 동작, Master 동작 시 LED 2번 동작
4. PORTB를 출력 포트로 설정 (DDRB = 0xFF)
5. LED는 Active Low (ATMega128에서 0 V 출력할 때 LED가 ON됨)
6. 초기값으로 PORTB의 상위 비트 (PORTB의 4~7 비트) 1로 설정 (PORTB = 0xF0 = 1111 0000)
7. PORTA를 입력 포트로 설정 (DDRA = 0x00)
8. 딥 스위치는 Active High (ATMega128에 3.3 V 또는 5 V 입력할 때 딥 스위치 인식)
9. 초기값으로 PORTA의 상위 비트 (PORTA의 4~7 비트) 0으로 설정 (PORTA = 0x0F = 0000 1111)
10. 푸시 스위치는 Active Low (ATMega128에 0 V 입력할 때 푸시 스위치 인식)
11. 초기값으로 PORTA의 하위 비트 (PORTA의 0~3 비트) 1로 설정 (PORTA = 0x0F = 0000 1111)
12. PORTC를 입/출력 포트로 설정 (DDRC = 0x04 = 0000 0100)
    - PORTC의 2 번째 포트 출력 설정 (PORTC의 2번째 포트를 S-CONTROL로 사용)
    - PORTC의 나머지 포트 입력 설정 (PORTC의 1번째 포트를 S-STATUS로 사용)

```
void init_timer0(void)
{

    TIMSK |= 1 << TOIE0;
    TCNT0 = 0;
    TCCR0 = 5;
    timer0_counter = 0;
}
```

< ATMega128 Timer0 초기화 >

1. ATMega128의 Timer0를 LED의 상태를 나타내는 간격으로 사용
2. Timer0의 인터럽트 허용 설정 (TIMSK |=1<<TOIE0)
3. Timer0의 카운터는 0부터 시작 (TCNT0 = 0)
4. Timer0의 분주 비 128 설정 (TCCR0 = 5 = 0x05 = 0000 0101)

39. Initialize Function – uart0, uart1

```
void Init_UART0(unsigned char baudrate)
{
    unsigned int i;

    UBRR0H    = 0;
    if(baudrate == BAUD_9600){UBRR0L = 71;}/* 9600 BAUD at 11.0592MHz*/
    else if(baudrate == BAUD_19200){UBRR0L = 35;}/* 19200 BAUD at 11.0592MHz*/
    else if(baudrate == BAUD_38400){UBRR0L = 17;}/* 38400 BAUD at 11.0592MHz*/
    else if(baudrate == BAUD_57600){UBRR0L = 11;}/* 57600 BAUD at 11.0592MHz*/
    else if(baudrate == BAUD_115200){UBRR0L = 5;} /* 115200 BAUD at 11.0592MHz*/
    else {UBRR0L = 71;};/* default 9600 BAUD at 11.0592MHz*/
    UCSR0B = (1<<RXCIE)|(1<<RXEN)|(1<<TXEN);
    p_rx0_wr = 0;
    p_rx0_rd = 0;
    for (i=0; i<UART0_BUF_SIZE; i++) rx0_buf[i] = 0;
}
```

< ATMega128 uart0 초기화 >

1. UBBR0L 설정에 따른 시리얼 데이터 통신 속도 설정
2. 수신 완료 인터럽트 허용 (RXCIE)
3. UART0 수신 부 동작 허용 (RXEN)
4. UART0 송신 부 동작 허용 (TXEN)
5. 수신 데이터 저장 버퍼 및 수신 데이터 포인터 초기값 설정

```
void Init_UART1(unsigned char baudrate)
{
    unsigned int i;

    UBRR1H    = 0;
    if(baudrate == BAUD_9600){UBRR1L = 71;}/* 9600 BAUD at 11.0592MHz*/
    else if(baudrate == BAUD_19200){UBRR1L = 35;}/* 19200 BAUD at 11.0592MHz*/
    else if(baudrate == BAUD_38400){UBRR1L = 17;}/* 38400 BAUD at 11.0592MHz*/
    else if(baudrate == BAUD_57600){UBRR1L = 11;}/* 57600 BAUD at 11.0592MHz*/
    else if(baudrate == BAUD_115200){UBRR1L = 5;} /* 115200 BAUD at 11.0592MHz*/
    else {UBRR1L = 71;};/* default 9600 BAUD at 11.0592MHz*/
    UCSR1B = (1<<RXCIE)|(1<<RXEN)|(1<<TXEN);
    p_rx1_wr = 0;
    p_rx1_rd = 0;
    for (i=0; i<UART1_BUF_SIZE; i++) rx1_buf[i] = 0;
}
```

< ATMega128 uart1 초기화 >

1. UBBR1L 설정에 따른 시리얼 데이터 통신 속도 설정
2. 수신 완료 인터럽트 허용 (RXCIE)
3. UART0 수신 부 동작 허용 (RXEN)
4. UART0 송신 부 동작 허용 (TXEN)
5. 수신 데이터 저장 버퍼 및 수신 데이터 포인터 초기값 설정

40. Uart0 Function

```
unsigned char PutChar0 (unsigned char c)
{
    while (!(UCSR0A & 0x20)) ;
    UDR0 = c;
    return 0;
}

void DispStr0 (char *s)
{
    while (*s != '\0') {
        PutChar0(*s++);
    }
}

unsigned int Check_Rx_Buf0(void)
{
    unsigned int len;

    if (p_rx0_wr == p_rx0_rd) {
        len = 0;
    }
    else {
        if (p_rx0_wr > p_rx0_rd) len = p_rx0_wr - p_rx0_rd;
        else len = UART0_BUF_SIZE + p_rx0_wr - p_rx0_rd;
    }
    return len;
}

unsigned char GetChar0 (void)
{
    unsigned char ch;

    ch = rx0_buf[p_rx0_rd];
    p_rx0_rd++;
    if (p_rx0_rd > UART0_BUF_SIZE-1) p_rx0_rd = 0;
    return ch;
}
```

< PutChar0( ) >

1. UCSR0A 레지스터의 5번째 비트가 1 이 되면 1 바이트 시리얼 데이터를 UART0 포트로 출력

< DispStr0( ) >

1. 문자열로 된 시리얼 데이터를 UART0 포트로 출력

< Check_Rx_Buf0( ) >

1. UART0로 입력된 데이터는 rx0_buf[] 버퍼에 저장
2. Check_Rx _Buf0() 함수는 rx0_buf[] 버퍼에 저장되어 있는 데이터의 개수를 체크하는 함수

< GetChar0( ) >

1. rx0_buf[]에 저장되어 있는 데이터를 1바이트씩 꺼내오는 함수

41. Uart1 Function

```
unsigned char PutChar1 (unsigned char c)
{
    while (!(UCSR1A & 0x20)) ;
    UDR1 = c;
    return 0;
}

void DispStr1 (char *s)
{
    while (*s != '\0') {
        PutChar1(*s++);
    }
    PutChar1(0x0a);
    PutChar1(0x0d);
}

void DispStr1_line (char *s)
{
    while (*s != '\0') {
        PutChar1(*s++);
    }
}

unsigned int Check_Rx_Buf1(void)
{
    unsigned int len;

    if (p_rx1_wr == p_rx1_rd) {
        len = 0;
    }
    else {
        if (p_rx1_wr > p_rx1_rd) len = p_rx1_wr - p_rx1_rd;
        else len = UART1_BUF_SIZE + p_rx1_wr - p_rx1_rd;
    }
    return len;
}

unsigned char GetChar1 (void)
{
    unsigned char ch;

    ch = rx1_buf[p_rx1_rd];
    p_rx1_rd++;
    if (p_rx1_rd > UART1_BUF_SIZE-1) p_rx1_rd = 0;
    return ch;
}
```

< PutChar1( ) >

1. UCSR1A 레지스터의 5번째 비트가 1 이 되면 1 바이트 시리얼 데이터를 UART1 포트로 출력

< DispStr1( ) >

1. 문자열로 된 시리얼 데이터를 UART1 포트로 출력
2. 마지막에 0x0a(Line Feed) 0x0d(Carriage return)를 출력

< DispStr1_line( ) >

1. 문자열로 된 시리얼 데이터를 UART1 포트로 출력
2. 마지막에 0x0a(Line Feed) 0x0d(Carriage return)를 출력하지 않음

< Check_Rx_Buf1( ) >

1. UART1로 입력된 데이터는 rx1_buf[] 버퍼에 저장
2. Check_Rx _Buf1() 함수는 rx1_buf[] 버퍼에 저장되어 있는 데이터의 개수를 체크하는 함수

< GetChar1( ) >

1. rx1_buf[]에 저장되어 있는 데이터를 1바이트씩 꺼내오는 함수

42. LED 처리 함수

```
void status_led(void)
{
    if(status_sequence == BEFORE_CONNECT)
    {
        if (timer0_counter < 100 )
        {
            if(read_avr_port == DIP_1)
                write_port_value(WRITE_LED_1_ON);
            else if(read_avr_port == DIP_2)
                write_port_value(WRITE_LED_2_ON);
        }
        else if (timer0_counter < 200 )
        {
            if(read_avr_port == DIP_1)
                write_port_value(WRITE_LED_1_OFF);
            else if(read_avr_port == DIP_2)
                write_port_value(WRITE_LED_2_OFF);
        }
    }
    else
    {
        if(read_avr_port == DIP_1)
            write_port_value(WRITE_LED_1_ON);
        else if(read_avr_port == DIP_2)
            write_port_value(WRITE_LED_2_ON);
    }
} ? end status_led ?
```

< status_led( ) >

1. LED의 상태를 나타내는 함수
2. timer0_counter는 timer0 인터럽트 벡터 상에서 카운트 됨
3. Bluetooth가 연결되기 전에는 LED 가 깜빡임
4. Bluetooth가 연결되면 LED가 ON 된 상태 유지
5. 딥 스위치 1번이 ON 된 상태에서는 1번 LED가 동작
6. 딥 스위치 2번이 ON 된 상태에서는 2번 LED가 동작
7. LED의 ON/OFF는 write_port_value( ) 함수 사용

43. Hex To Char 변경 / 시간 지연 함수

```
/*
****************************************
CONVERT HEX INTO CHAR
****************************************
*/
unsigned char Hex2Char(unsigned char c)
{
    if ((c == 0)||(c == 9)||(c > 0&&c < 9))
        return '0' + c;
    if (c >= 10 && c <= 15)
        return 'A' + c - 10;

    return c;
}

/*
****************************************
Function For Delay
****************************************
*/
void wait_1ms(unsigned int cnt)
{
    unsigned int i;

    for (i = 0; i < cnt; i++) wait_1us(1000);
}

void wait_1us(unsigned int cnt)
{
    unsigned int i;

    for (i = 0; i < cnt; i++) ;
}
```

< Hex2Char( ) >

1. 1 바이트의 HEX 값을 1 바이트의 Char로 변경
2. 하이퍼 터미널과 같은 시리얼 통신 프로그램에서 정상적으로 데이터가 표시 되기 위해서는 HEX 값을 Char 형태로 변경해야 함
3. HEX값 0x01은 Char 타입의 1(HEX값으로 0x31)로 변경됨
4. HEX값 0x0a는 Char 타입의 A(HEX값으로 0x41)로 변경됨

< 시간 지연 함수 >

1. wait_1ms(파라미터)
   - 1ms 단위로 설정된 파라미터까지 시간을 지연
2. wait_1us(파라미터)
   - 1us 단위로 서정된 파라미터까지 시간을 지연

44. 포트 읽기 / 포트 쓰기 함수

```
Function For PORT Read
*****************************************************************
*/
unsigned char read_port_value(unsigned char avr_port_num)
{
    unsigned char return_avr_port_value;

    switch(avr_port_num)
    {
        case READ_DIP:
            return_avr_port_value = PINA;
            return_avr_port_value = return_avr_port_value & 0xf0;
            break;
        case READ_SWITCH:
            return_avr_port_value = PINA;
            return_avr_port_value = return_avr_port_value & 0x0f;
            break;
        //FB755AX
        case READ_S_STATUS:
            return_avr_port_value = PINC;
            return_avr_port_value = return_avr_port_value & 0x02;
            break;
        //
        default:
            break;
    }
    return return_avr_port_value;
} ? end read_port_value ?
/*
*****************************************************************
Function For PORT Write
*****************************************************************
*/
void write_port_value(unsigned char avr_port_value)
{

    switch(avr_port_value)
    {
        case WRITE_LED_1_ON:
            cbi(PORTB,PB4);
            break;
        case WRITE_LED_1_OFF:
            sbi(PORTB,PB4);
            break;
        case WRITE_LED_2_ON:
            cbi(PORTB,PB5);
            break;
        case WRITE_LED_2_OFF:
            sbi(PORTB,PB5);
            break;
        //FB755AX
        case WRITE_S_CONTROL_HIGH:
            sbi(PORTC,PC2);
            break;
        case WRITE_S_CONTROL_LOW:
            cbi(PORTC,PC2);
            break;
        //
        default:
            break;
    } ? end switch avr_port_value ?
} ? end write_port_value ?
```

< read_port_value( ) >

1. ATMega128의 포트를 읽어오는 함수 (각 포트는 0 ~ 7 비트로 8개의 비트로 구성)
2. 딥 스위치 값으로 할당된 PORTA의 상위 비트 를 읽음 (...&0xF0)
3. 스위치 값으로 할당된 PORTA의 하위 비트 를 읽음 (...&0x0F)
4. S-STATUS 값으로 할당된 PORTC의 1번째 비트 읽음 (...&0x02)
5. 읽은 포트 값 리턴

< write_port_value( ) >

1. ATMega128의 포트 값을 설정하는 함수
2. LED로 할당된 PORTB의 값을 설정
3. S-CONTROL로 할당된 PORTC의 값 설정
4. cbi( ) (Low 출력) / sbi( ) (High 출력)를 이용하여 비트 별로 설정

45. 비교버퍼 초기화 / 에러 처리 함수

```
/*
*********************************************
* Function For Init uart0_read_data
*********************************************
*/
void init_uart0_read_data(void)
{
     unsigned int i;

     for(i = 0;i<50;i++)
          uart0_read_data[i] = 0;
}
/*
*********************************************
* Function For error
*********************************************
*/
void execution_error(void)
{
     unsigned int i;

     DispStr1_line("[ERROR ");
     DispStr1_line(" STEP : ");
     if(read_avr_port == DIP_1)
          PutChar1(Hex2Char(step_slave_bt));
     else if(read_avr_port == DIP_2)
          PutChar1(Hex2Char(step_master_bt));
     DispStr1_line(" UART_LEN : ");
     PutChar1(Hex2Char(uart0_length));
     DispStr1_line(" DATA : ");
     for(i = 0;i < uart0_length;i++)
          PutChar1(uart0_read_data[i]);
     if(read_avr_port == DIP_1)
          step_slave_bt = 50;
     else if(read_avr_port == DIP_2)
          step_master_bt = 50;

} ? end execution_error ?
```

< init_uart0_read_data( ) >

1. uart0_read_data[] 버퍼는 수신된 데이터의 비교 버퍼로 사용
2. uart0_read_data[] 버퍼의 데이터를 초기화 하는 함수

< execution_error( ) >

1. 수신 데이터 비교 중에 비교 데이터와 다른 경우 수행되는 함수
2. 시리얼로 “ERROR”를 출력하고, ERROR가 발생된 현재의 STEP 출력, Bluetooth로부터 수신 받은 데이터의 길이 출력, Bluetooth 로부터 수신 받은 데이터 출력
3. ERROR 처리 후, 스텝을 50으로 설정하여 더 이상 다른 스텝이 진행되지 않게 설정

46. Interrupt Vector

```
SIGNAL(SIG_UART0_RECV)
{
    rx0_buf[p_rx0_wr++] = UDR0;
    if(rx0_buf[p_rx0_wr-1] == LF_VALUE)if_check_flag++;
    if (p_rx0_wr > UART0_BUF_SIZE-1)p_rx0_wr = 0;
}
```

< uart0 interrupt vector >

1. Uart0로 데이터가 입력된 경우 수행되는 부분
2. Uart0으로 입력된 데이터는 UDR0 에 저장되어 있음
3. UDR0 에 저장되어 있는 1바이트의 데이터를 rx0_buf [] 버퍼에 저장
4. 입력된 데이터가 0x0a(Line Feed)인지 검사
5. rx0_buf[]의 저장 공간이 더 이상 없는 경우, rx0_buf[]의 처음부터 저장 (링 버퍼 구성)
6. Bluetooth로부터 수신된 데이터의 비교 시작을 0x0a가 2개 검출된 시점부터 적용

```
SIGNAL(SIG_UART1_RECV)
{
    rx1_buf[p_rx1_wr++] = UDR1;
    if (p_rx1_wr > UART1_BUF_SIZE-1)p_rx1_wr = 0;
}
```

< uart1 interrupt vector >

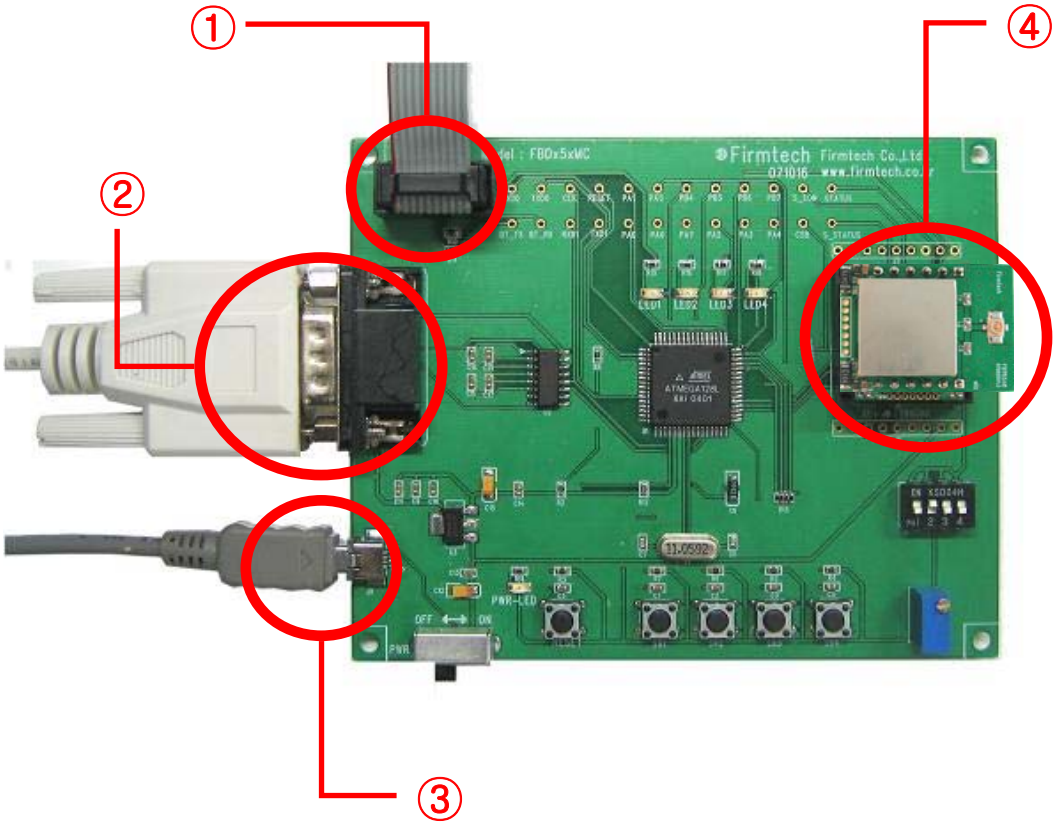
1. Uart1로 데이터가 입력된 경우 수행되는 부분
2. Uart1로 입력된 데이터는 UDR1 에 저장되어 있음
3. UDR1 에 저장되어 있는 1바이트의 데이터를 rx1_buf [] 버퍼에 저장
4. rx1_buf[]의 저장 공간이 더 이상 없는 경우, rx1_buf[]의 처음부터 저장 (링 버퍼 구성)

```
SIGNAL(SIG_OVERFLOW0)
{
    timer0_counter++;
    transfer_interval++;
    if (timer0_counter > 200 )
    {
        timer0_counter = 0;
    }
    if (transfer_interval > 60000 )
    {
        transfer_interval = 0;
    }
    status_led();
}
```

< timer0 interrupt vector >

1. timer0으로 설정된 시간이 되면 수행되는 부분
2. 설정된 시간마다 timer0_counter를 1씩 증가
3. timer0_counter가 200보다 커지면 timer0_counter를 0으로 설정
4. 설정된 시간마다 status_led( )함수를 이용하여 LED 상태 설정
5. 설정된 시간마다 transfer_interval을 1씩 증가
6. transfer_interval이 60000보다 커지면 transfer_interval을 0으로 설정

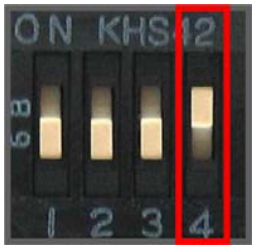
47. FBDx5xMC 보드의 올바른 연결



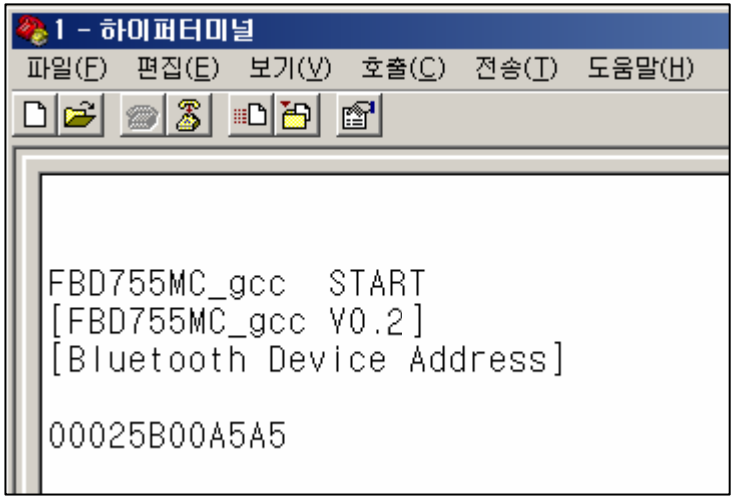
< 제품의 올바른 연결 >

1. PC와 FBDx5xMC를 AVR Loader로 연결
2. PC와 FBDx5xMC를 RS-232 Cable로 연결
3. PC와 FBDx5xMC를 USB Power Cable로 연결
4. FB755AS와 FBDx5xMC를 연결
   (장착 방향 유의)

48. Bluetooth Device Address (Slave Address) 확인 하는 방법



< 딥 스위치 4번 ON 상태 >



< 하이퍼 터미널에 출력된 Bluetooth Device Address >

< Bluetooth Device Local Address 확인 하는 방법>

1. FBDx5xMC 보드에 Bluetooth Device를 장착
   - 장착 방향에 유의
2. FBDx5xMC 보드와 PC를 RS-232 Cable로 연결
3. FBDx5xMC 보드에 Power Cable 연결
4. PC에서 하이퍼 터미널 실행
   - 통신속도 : 9600bps
   - 데이터 비트 : 8 비트
   - 패리티 : 없음
   - 정지 비트 : 1
   - 흐름제어 : 없음
5. FBDx5xMC 보드의 딥 스위치 4번 ON
6. FBDx5xMC 보드의 전원 ON
7. Bluetooth Device Address 하이퍼 터미널로 출력
   - 기본적으로 제공되는 소스를 이용하여 Bluetooth Device Address 확인 가능
   - HEX 파일 다운로드 없이 기본적으로 동작되는 FBDx5xMC보드에 Bluetooth Device 장착하여 Bluetooth Device Address 확인 가능

49. FBD755MC_gcc 프로그램 수정 (Slave Address 수정)

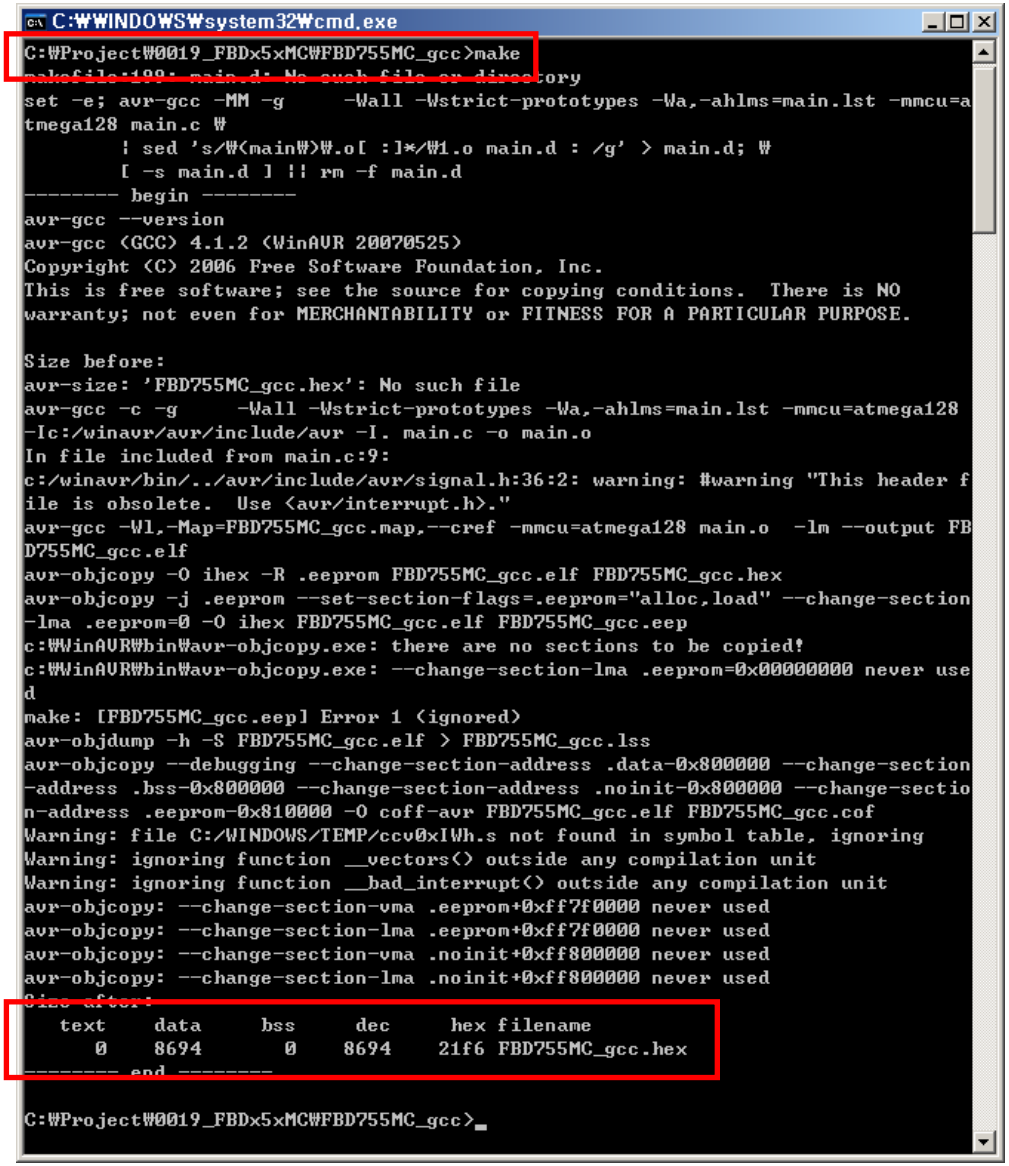
```
void execution_master(void)
{
    unsigned int i;

    //Write Slave Bluetooth Device Address
    memcpy(target_data,"00025b00a5a5",12);
    //
    status_sequence = BEFORE_CONNECT;
    step_master_bt = STEP_1;
    init_uart0_read_data();
    while(1)
    {
        if(status_sequence == BEFORE_CONNECT)
        {
            uart0_length = Check_Rx_Buf0();
            if(uart0_length>0)
            {
                if(lf_check_flag == 2)
                {
                    uart0_length = Check_Rx_Buf0();
                    for(i = 0;i<uart0_length;i++)
                        uart0_read_data[i] = GetChar0();
                    compare_master_command();
                }
            }
        }
        else if(status_sequence == AFTER_CONNECT)
        {
            lf_check_flag = LF_CHECK_NON;
            execution_distribute_received_message();
            execution_transfer_data();
        }
    } ? end while 1 ?
} ? end execution_master ?
```

< 슬레이브 어드레스 저장 >

1. execution_master( ) 소스에 확인된 Slave Bluetooth Device의 Address를 입력
2. Slave의 Address를 정확히 입력하지 않으면 Master / Slave의 연결이 원활하지 않음
3. 예제 소스의 00025b00a5a5 부분에 확인된 Slave Bluetooth Device의 Address 12바이트를 정확하게 입력
4. 수정 완료된 프로그램 컴파일 진행
5. 컴파일 진행 후 생성된 HEX 파일 FBDx5xMC 보드에 다운로드

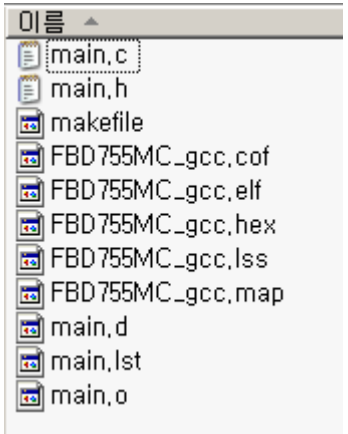
50. FBD755MC_gcc 프로그램 컴파일 (WINAVR 컴파일러 사용)



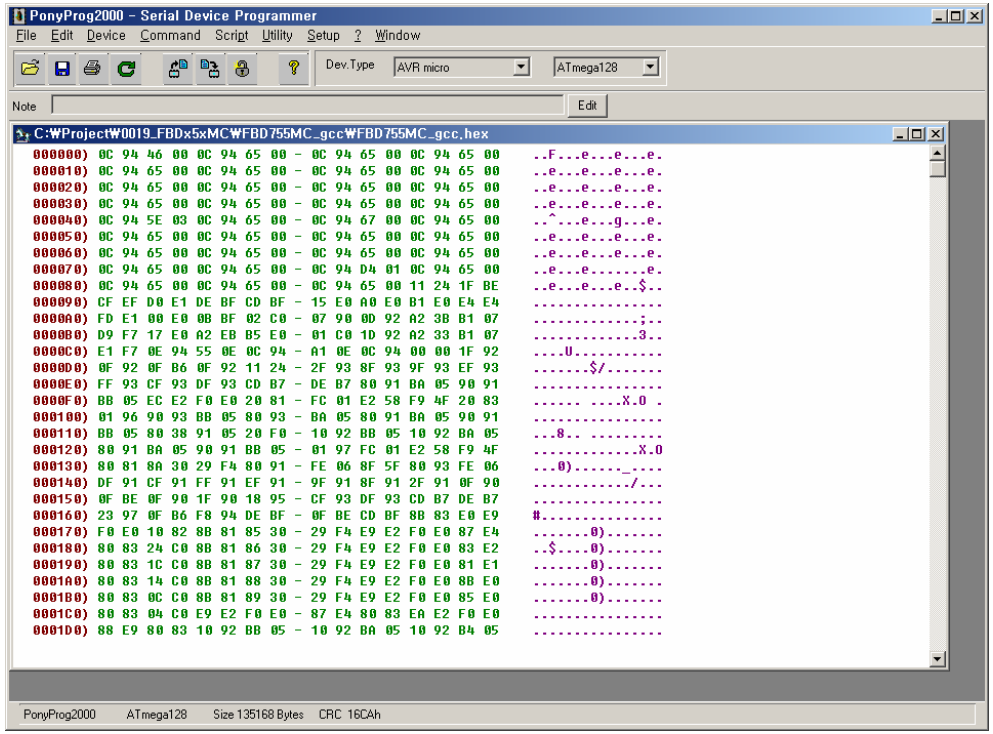
1. 사용 컴파일러
   - 사용하는 컴파일러는 WINAVR 사용
   - http://winavr.sourceforge.net 에서 다운로드 하여 설치
2. 컴파일에 사용하는 파일
   - main.c
   - main.h
   - makefile
3. 컴파일 방법
   - Window Command 창을 이용하여 컴파일에 사용할 파일이 있는 위치로 이동
   - Make 입력 후 Enter Key 입력
   - ERROR 없이 진행된 경우, HEX 파일과 기타 파일 생성

< make 실행 이전 파일>

< make 실행 이후 파일>



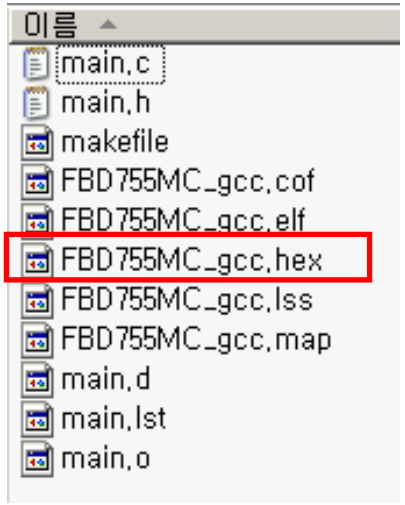
51. FBDx5xMC 보드에 프로그램 다운로드



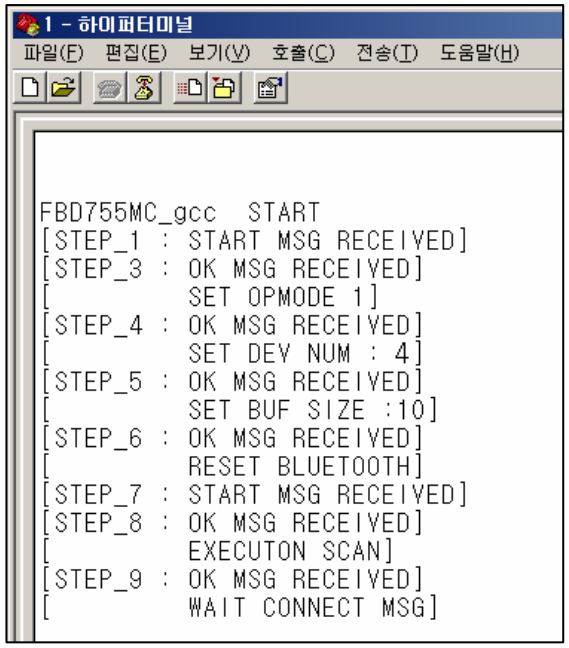
< HEX 파일 다운로드 프로그램 PonyProg>

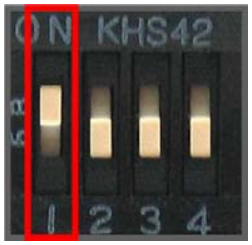
1. Slave Bluetooth Device Address를 확인하고, 확인한 Slave Bluetooth Device Address를 소스에 입력 후 컴파일 진행
   - WINAVR을 이용하여 컴파일 진행
2. 컴파일에 의해 생성된 HEX 파일을 FBDx5xMC 보드에 다운로드
   - 다운로드 프로그램은 PonyProg 사용
   - http://www.lancos.com/ppwin95.html 에서 다운로드 하여 설치
   - AVR Loader로 PC와 FBDx5xMC 보드 연결
   - FBDx5xMC 보드에 생성된 HEX 파일을 다운로드

< 다운로드 할 HEX 파일>



52. FBDx5xMC 보드의 Slave 설정 & 최초 동작 화면



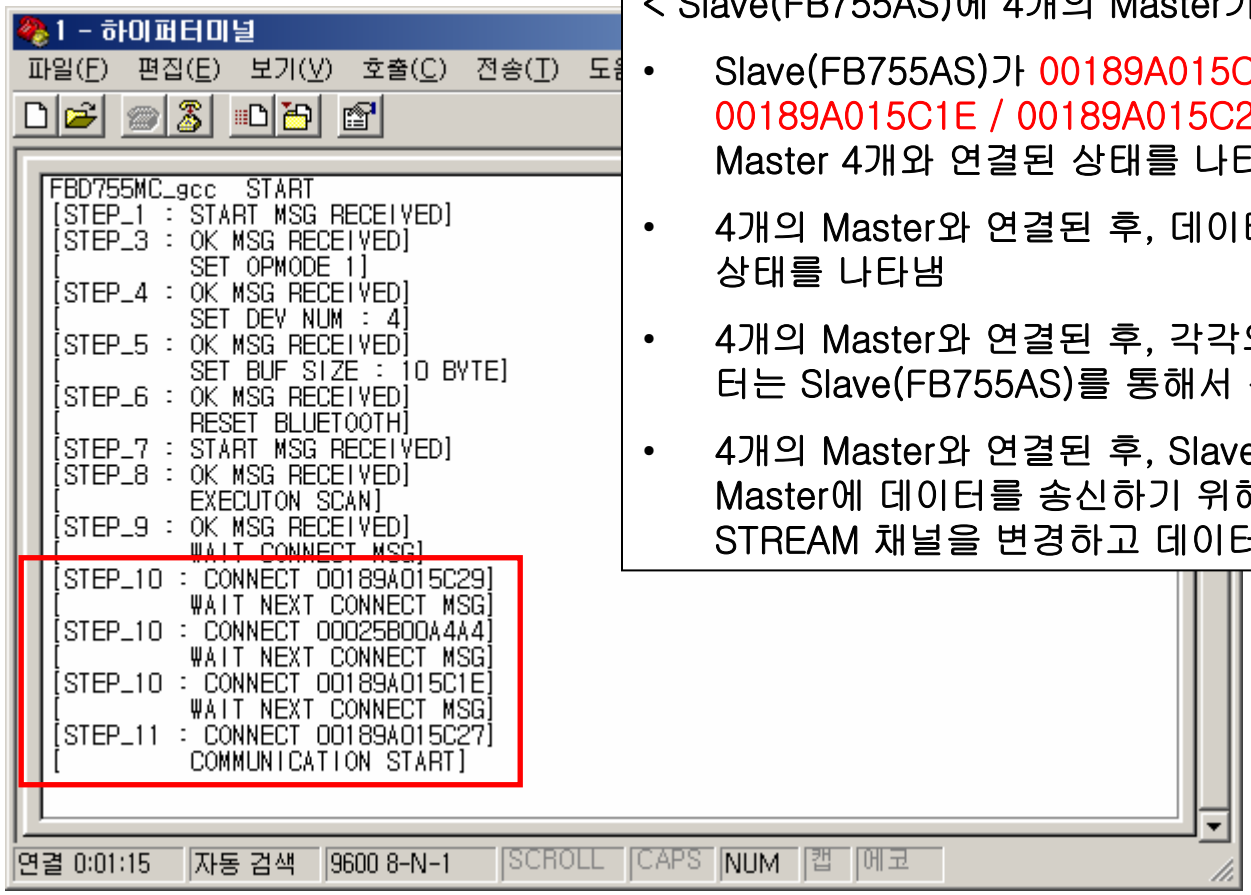


< 딥 스위치 1번 ON 상태 >

< Slave 설정 & 동작 화면 >

1. FBDx5xMC 보드를 Slave로 동작 시키기 위해 딥 스위치 1번 ON
2. FBDx5xMC 보드에 Bluetooth Device를 장착 후 전원 ON
3. Slave로 동작되는 경우, 1번 LED가 깜빡임
4. 하이퍼 터미널에 각 스텝 별 메시지 출력
   - STEP_1 : BTWIN Slave mode start 메시지 수신
   - STEP_3 : OK 메시지 수신, at+btopmode,1 명령 전달
   - STEP_4 : OK 메시지 수신, at+btdev=4 명령 전달
   - STEP_5 : OK 메시지 수신, at+btbuff=10 명령 전달
   - STEP_6 : OK 메시지 수신, atz 명령 전달
   - STEP_7 : BTWIN Slave mode start 메시지 수신
   - STEP_8 : OK 메시지 수신, at+btscan 명령 전달
   - STEP_9 : OK 메시지수신, CONNECT 메시지수신 대기
     - ✓ 1회의 SCAN 명령을 전달하면, FB755AS에 설정한 연결될 디바이스의 수만큼 자동으로 SCAN 작업 진행
     - ✓ FB755AS에 설정한 연결될 디바이스의 수만큼 Master가 전부 연결된 경우, 1번 LED ON된 상태 유지
     - ✓ FB755AS에 설정한 연결될 디바이스의 수만큼 Master가 전부 연결된 경우, Master에서 송신한 데이터가 FB755AS에 수신됨(CNT_MODE4)
     - ✓ FB755AS에 설정한 연결될 디바이스의 수만큼 Master가 전부 연결된 경우, Switch를 이용하여 STREAM 채널을 변경하고 FB755AS에서 Master로 데이터 송신 가능(OP_MODE1)

53. Slave(FB755AS)에 설정한 “연결될 디바이스 수”만큼 연결된 화면

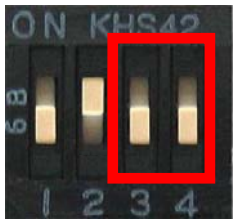


< Slave(FB755AS)에 4개의 Master가 연결된 화면 >

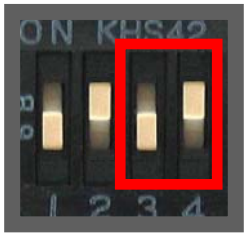
- Slave(FB755AS)가 00189A015C29 / 00025B00A4A4 / 00189A015C1E / 00189A015C27 어드레스에 해당하는 Master 4개와 연결된 상태를 나타냄
- 4개의 Master와 연결된 후, 데이터 송/수신 루틴 진행 가능한 상태를 나타냄
- 4개의 Master와 연결된 후, 각각의 Master에서 송신하는 데이터는 Slave(FB755AS)를 통해서 구분 문자와 데이터로 출력됨
- 4개의 Master와 연결된 후, Slave(FB755AS)에서 각각의 Master에 데이터를 송신하기 위해서는 Switch를 이용하여 STREAM 채널을 변경하고 데이터 송신

54. FBDx5xMC 보드의 Master 구분 문자 설정 사항

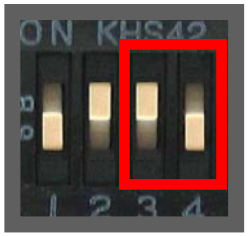
FBDx5xMC 보드에 있는 2개의 딥 스위치를 Master 데이터의 구분 문자 설정에 사용 : 4개의 구분 문자 사용.
7개의 구분 문자를 사용하기 위해서는 사용자가 정의 하여 사용해야 함.



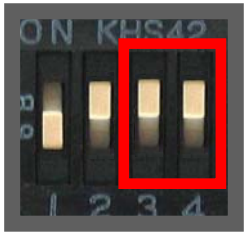
- 2번 딥 스위치 ON : Master 설정
- 3번 딥 스위치 OFF, 4번 딥 스위치 OFF : Master 구분 문자 “1” 설정 (= 00)



- 2번 딥 스위치 ON : Master 설정
- 3번 딥 스위치 OFF, 4번 딥 스위치 ON : Master 구분 문자 “2” 설정 (= 01)

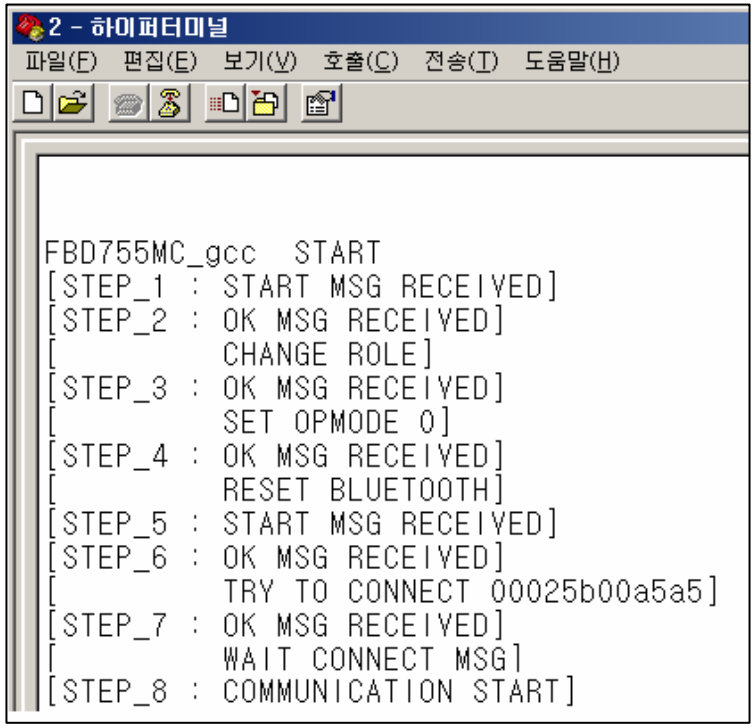


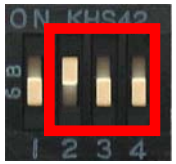
- 2번 딥 스위치 ON : Master 설정
- 3번 딥 스위치 ON, 4번 딥 스위치 OFF : Master 구분 문자 “3” 설정 (= 10)



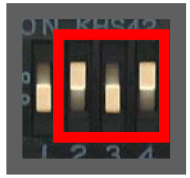
- 2번 딥 스위치 ON : Master 설정
- 3번 딥 스위치 ON, 4번 딥 스위치 ON : Master 구분 문자 “4” 설정 (= 11)

55. FBDx5xMC 보드의 Master 설정 & 동작 화면

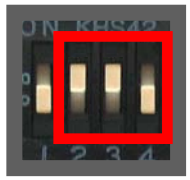




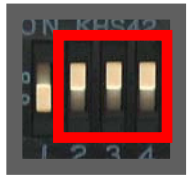
< Master, 구분 문자 “1”설정>



< Master, 구분 문자 “2”설정>



< Master, 구분 문자 “3”설정>



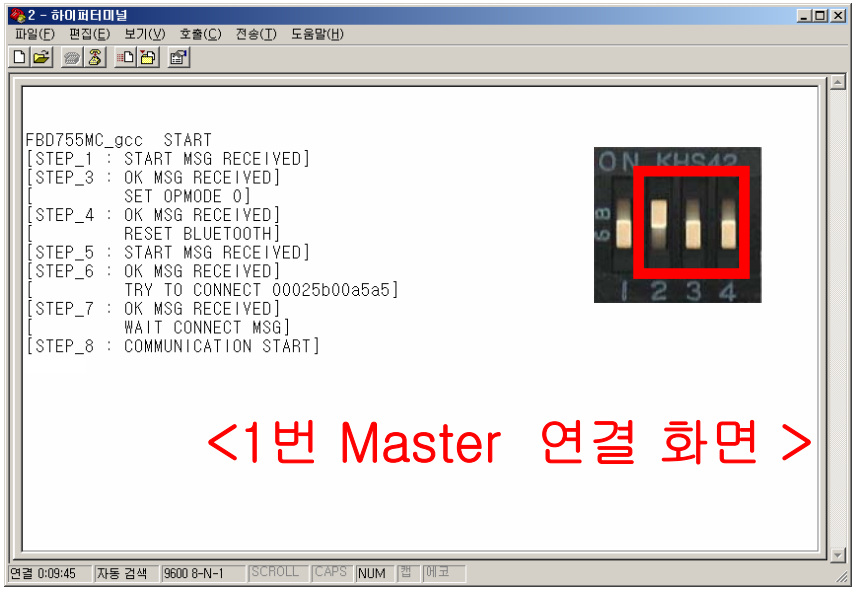
< Master, 구분 문자 “4”설정>

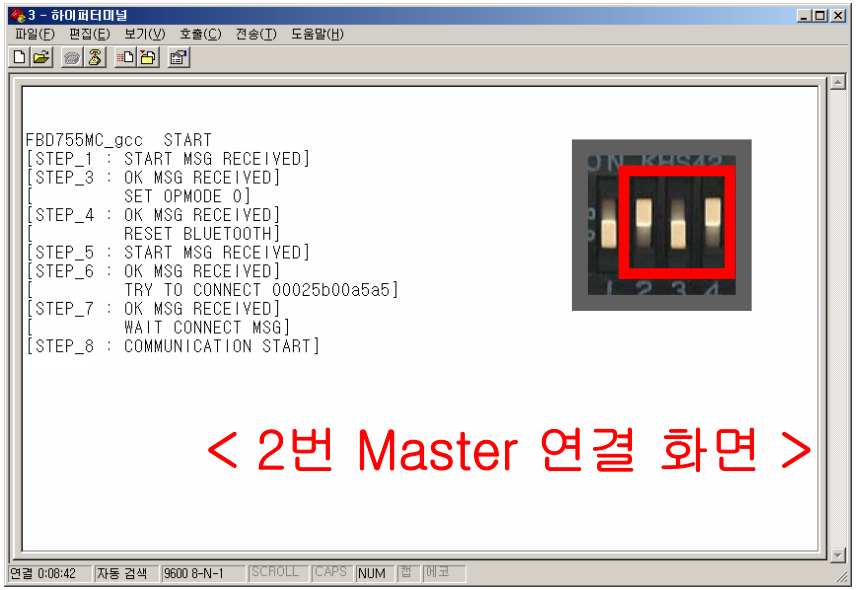
< Master 설정 & 동작 화면>

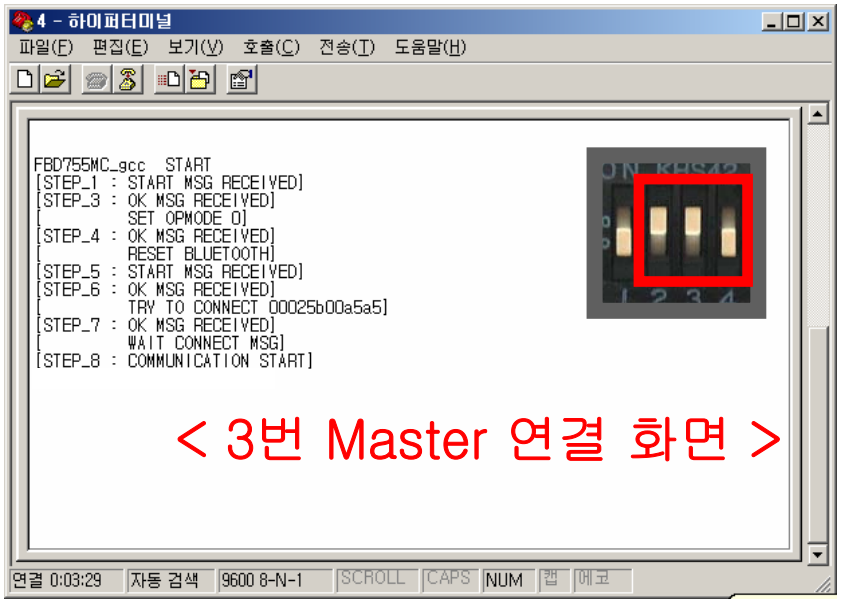
* Master가 실행되기 이전에 Slave가 SCAN 동작을 진행 하고 있어야 함

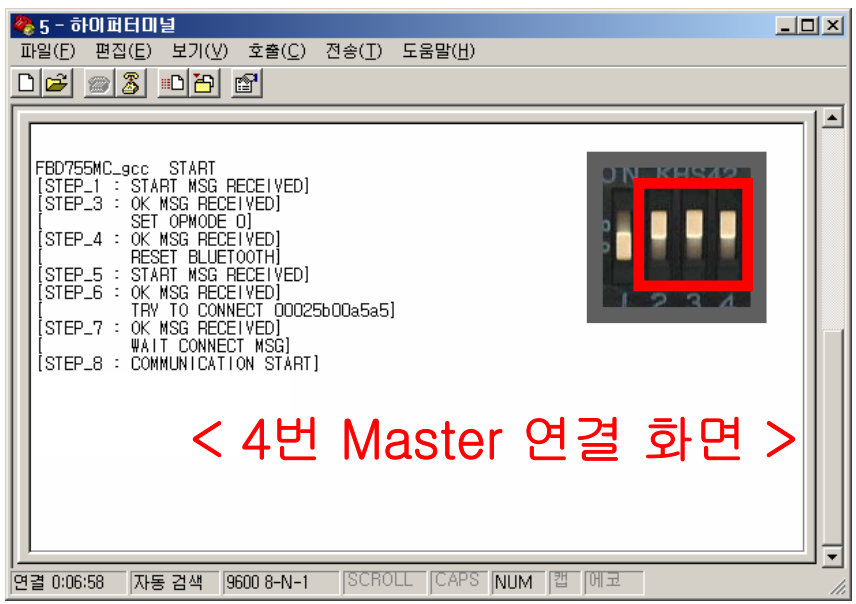
1. Master로 동작 시키기 위해 딥 스위치 2번 ON
2. Master 구분 문자 설정을 위해 딥 스위치 3번, 4번 설정
3. FBDx5xMC 보드에 Bluetooth Device를 장착 후 전원 ON
4. Master로 동작되는 경우, 2번 LED가 깜빡임
5. 하이퍼 터미널에 각 스텝 별 메시지 출력
   - STEP_1 : BTWIN Slave mode start 메시지 수신
   - STEP_2 : OK 메시지 수신, at+btrole=m 명령 전달
   - STEP_3 : OK 메시지 수신, at+btopmode,0 명령 전달
   - STEP_4 : OK 메시지 수신, atz 명령 전달
   - STEP_5 : BTWIN Master mode start 메시지 수신
   - STEP_6 : OK 메시지 수신, atd 저장어드레스 명령 전달
   - STEP_7 : OK 메시지 수신, CONNECT 메시지 수신 대기
   - STEP_8 : CONNECT 메시지 수신, 1번 LED ON 상태 유지, 데이터 송/수신 루틴 진행

56. Master가 연결된 화면

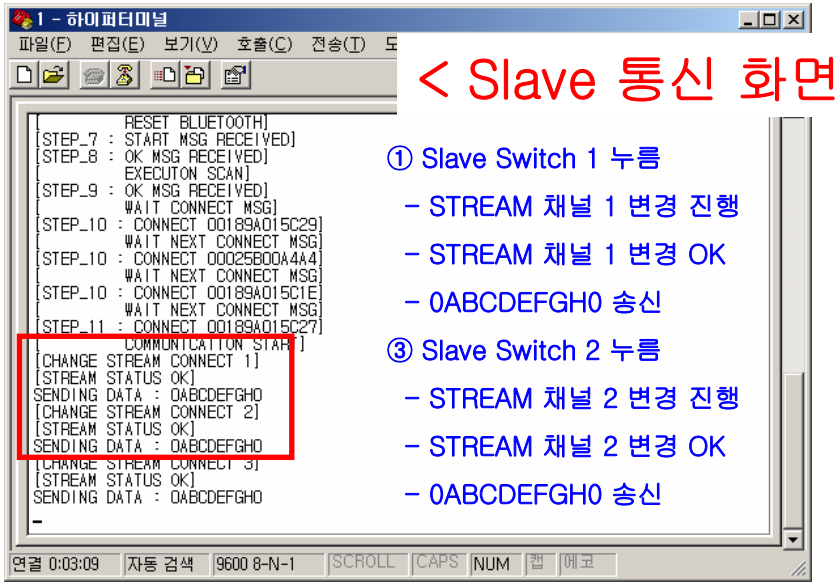








57. Slave에서 Master로 데이터를 송신한 화면



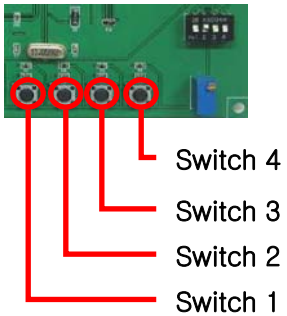
< Slave 통신 화면 >

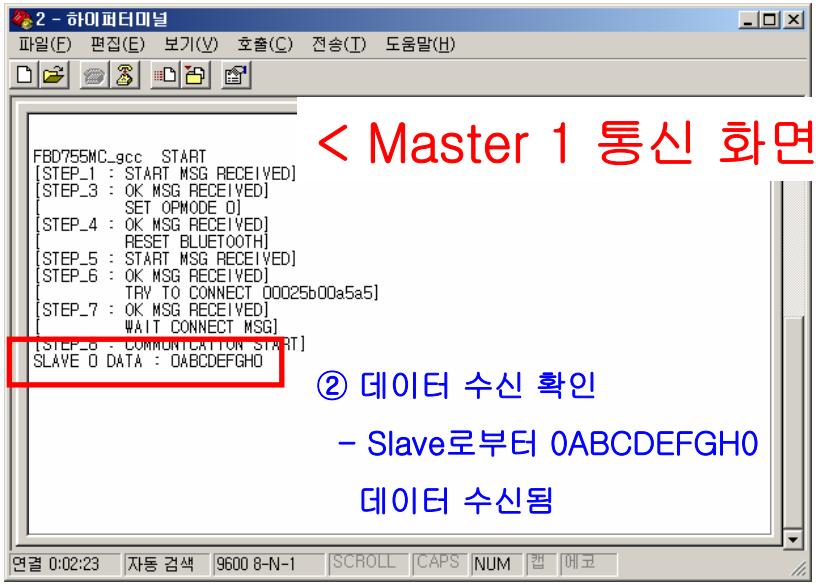
① Slave Switch 1 누름
- STREAM 채널 1 변경 진행
- STREAM 채널 1 변경 OK
- 0ABCDEFGH0 송신

③ Slave Switch 2 누름
- STREAM 채널 2 변경 진행
- STREAM 채널 2 변경 OK
- 0ABCDEFGH0 송신

< 데이터 송수신 테스트 절차 : Slave -> Master >

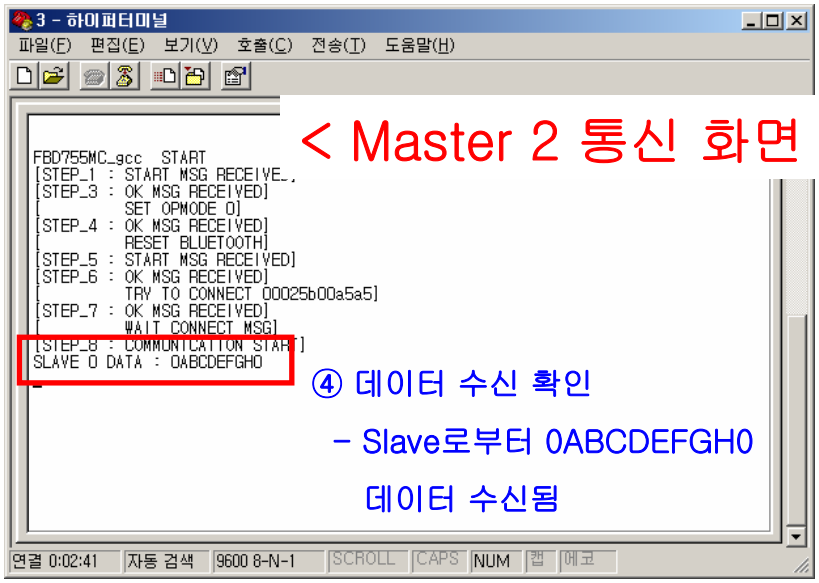
1. Slave의 Switch 1번 누름
   - STREAM 채널 1 변경 진행
   - STREAM 채널 변경 후 데이터 송신
2. 1번 Master에서 데이터 수신되는 것 확인
   - 0ABCDEFGH0 수신 확인
3. Slave의 Switch 2번 누름
   - STREAM 채널 2 변경 진행
   - STREAM 채널 변경 후 데이터 송신
4. 2번 Master에서 데이터수신 되는 것 확인
   - 0ABCDEFGH0 수신 확인





< Master 1 통신 화면 >

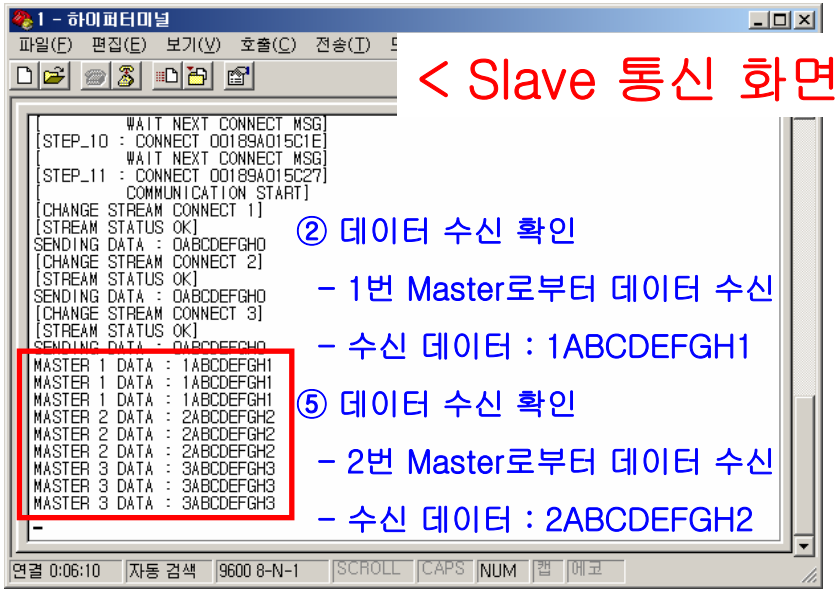
② 데이터 수신 확인
- Slave로부터 0ABCDEFGH0 데이터 수신됨



< Master 2 통신 화면 >

④ 데이터 수신 확인
- Slave로부터 0ABCDEFGH0 데이터 수신됨

58. Master에서 Slave로 데이터를 송신한 화면



< Slave 통신 화면 >

② 데이터 수신 확인

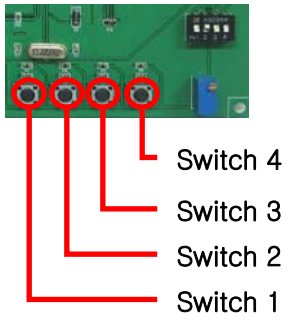
- 1번 Master로부터 데이터 수신
- 수신 데이터 : 1ABCDEFGH1

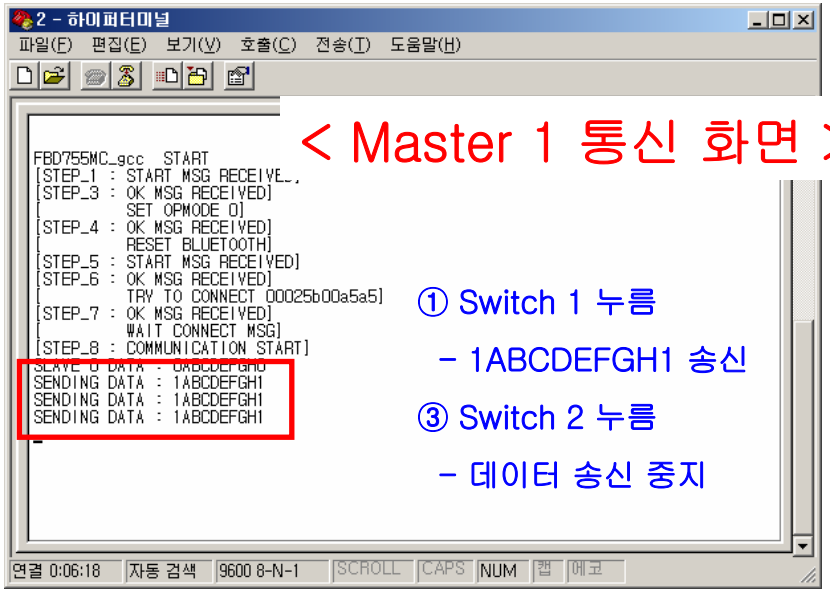
⑤ 데이터 수신 확인

- 2번 Master로부터 데이터 수신
- 수신 데이터 : 2ABCDEFGH2

< 데이터 송수신 테스트 절차 : Master -> Slave >

1. 1번 Master의 Switch 1 누름
   - Slave로 데이터 송신
   - 송신 형태 : 1ABCDEFGH1
2. Slave에서 데이터 수신되는 것 확인
   - 1ABCDEFGH1 수신 확인
3. 1번 Master의 Switch 1 이외의 Switch 누름
   - 데이터 송신 중지
4. 2번 Master의 Switch 1 누름
   - Slave로 데이터 송신
   - 송신 형태 : 2ABCDEFGH2
5. Slave에서 데이터수신 되는 것 확인
   - 2ABCDEFGH2 수신 확인
6. 2번 Master의 Switch 1 이외의 Switch 누름
   - 데이터 송신 중지





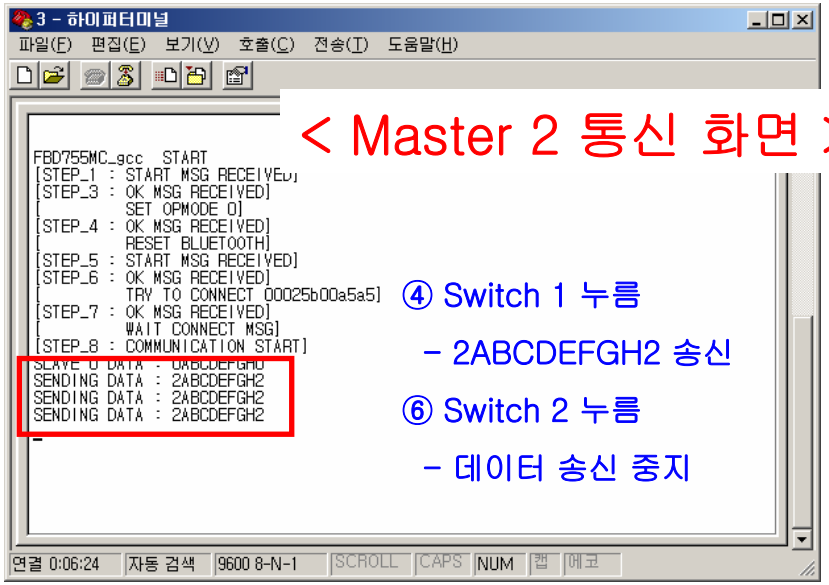
< Master 1 통신 화면 >

① Switch 1 누름

- 1ABCDEFGH1 송신

③ Switch 2 누름

- 데이터 송신 중지



< Master 2 통신 화면 >

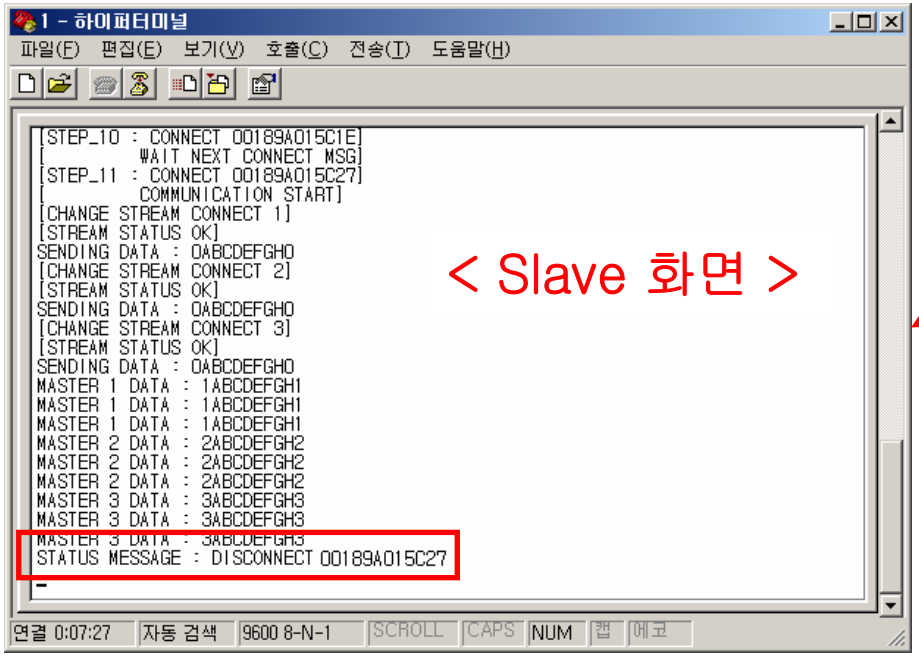
④ Switch 1 누름

- 2ABCDEFGH2 송신

⑥ Switch 2 누름

- 데이터 송신 중지

59. FB755AS로부터 상태 메시지를 수신 받은 Slave 화면

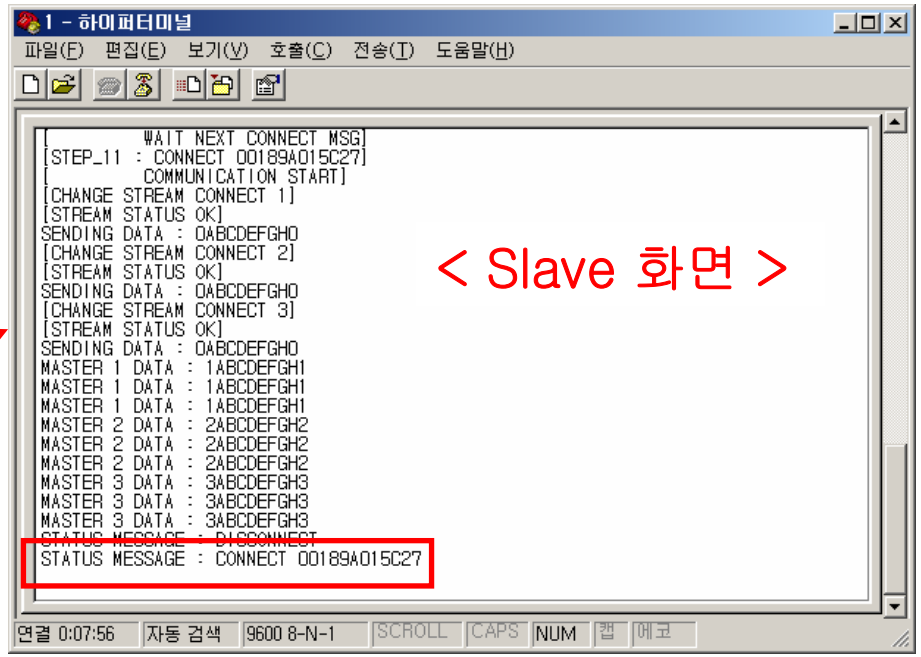


< 상태 메시지를 구분 하는 화면 : DISCONNECT >

1. Master중 1개의 전원을 OFF
2. 잠시 후 Status Message 출력
   - 구분된 메시지 : DISCONNECT

< 상태 메시지를 구분 하는 화면 : CONNECT >

1. 전원을 OFF 했던 Master의 전원 ON
2. 잠시 후 Status Message 출력
   - 구분된 메시지 : CONNECT



※ FBDx5xMC(FBD755MC_gcc) 사용시 주의 사항

1. Slave Bluetooth Device Address의 정확한 입력
   - Master Bluetooth Device가 Slave Bluetooth Device와 연결하기 위해서 Slave Bluetooth Device의 Address 필요
   - Slave Bluetooth Device의 정확한 Address를 execution_master() Source에 정확히 입력 후 컴파일 진행
   - Slave Bluetooth의 Address가 정확하지 않으면 연결이 되지 않음
2. AVR Loader의 연결
   - 사용자의 PC를 최초에 ON 하고, PonyProg 프로그램을 실행시키지 않은 상태에서, PC와 FBDx5xMC 보드 사이에 AVR Loader를 연결한여 FBDx5xMC 보드를 동작시키면 정상적으로 동작 되지 않음
   - FBDx5xMC 보드를 정상 동작 시키기 위해서, PC와 FBDx5xMC 보드 사이에 AVR Loader가 연결되어 있는 경우에는 PonyProg를 실행시켜야 함
   - FBDx5xMC 보드를 정상 동작 시키기 위해서, HEX 파일을 다운로드 시키는 경우를 제외하고는 PC와 FBDx5xMC 보드 사이에 AVR Loader를 연결하지 말아야 함
3. Bluetooth Device의 올바른 방향 장착
   - FBDx5xMC 보드에 장착하는 Bluetooth Device의 장착 방향 유의
   - Bluetooth Device를 잘못 장착 한 경우 FBDx5xMC 보드 뿐만 아니라 Bluetooth Device에도 무리가 발생할 수 있음
4. Bluetooth Device의 설정 상태
   - FBDx5xMC 보드는 장착되는 Bluetooth Device의 설정 상태가 공장 초기 설정 값을 기반으로 동작됨
   - 기본 설정 값 : Connect Mode 4 (AT-Command Mode), Status Message Enable, Baud rate 9600bps
5. Bluetooth Device 동작 순서
   - Slave Bluetooth Device가 Scan 동작을 진행한 이후에, Master Bluetooth Device가 연결을 진행 해야 함