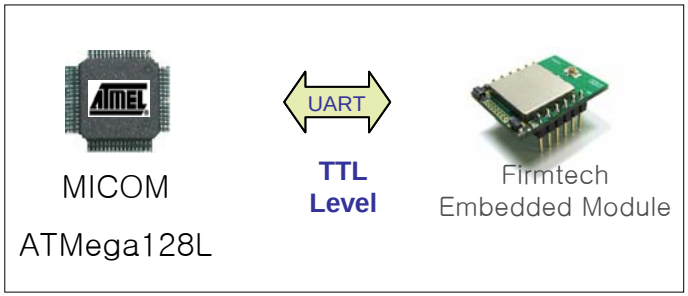
FBDx5xMC

< FBD755MC_gcc V0.1 >

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.

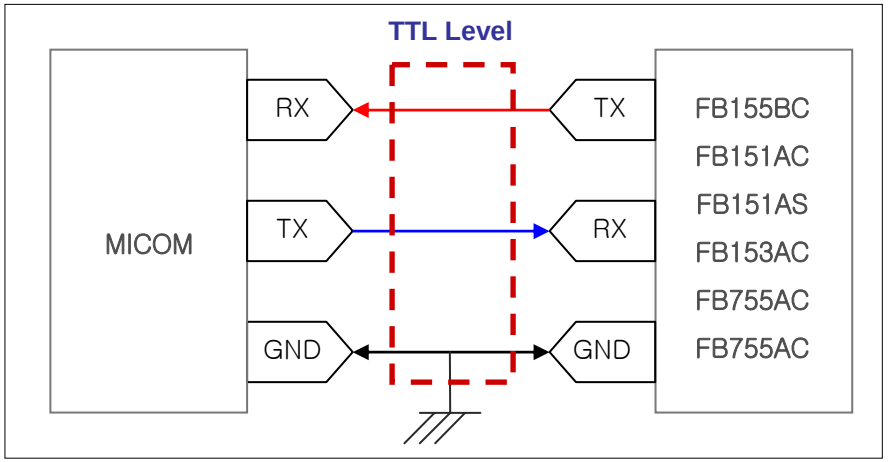
<1> 하드웨어 구성의 이해



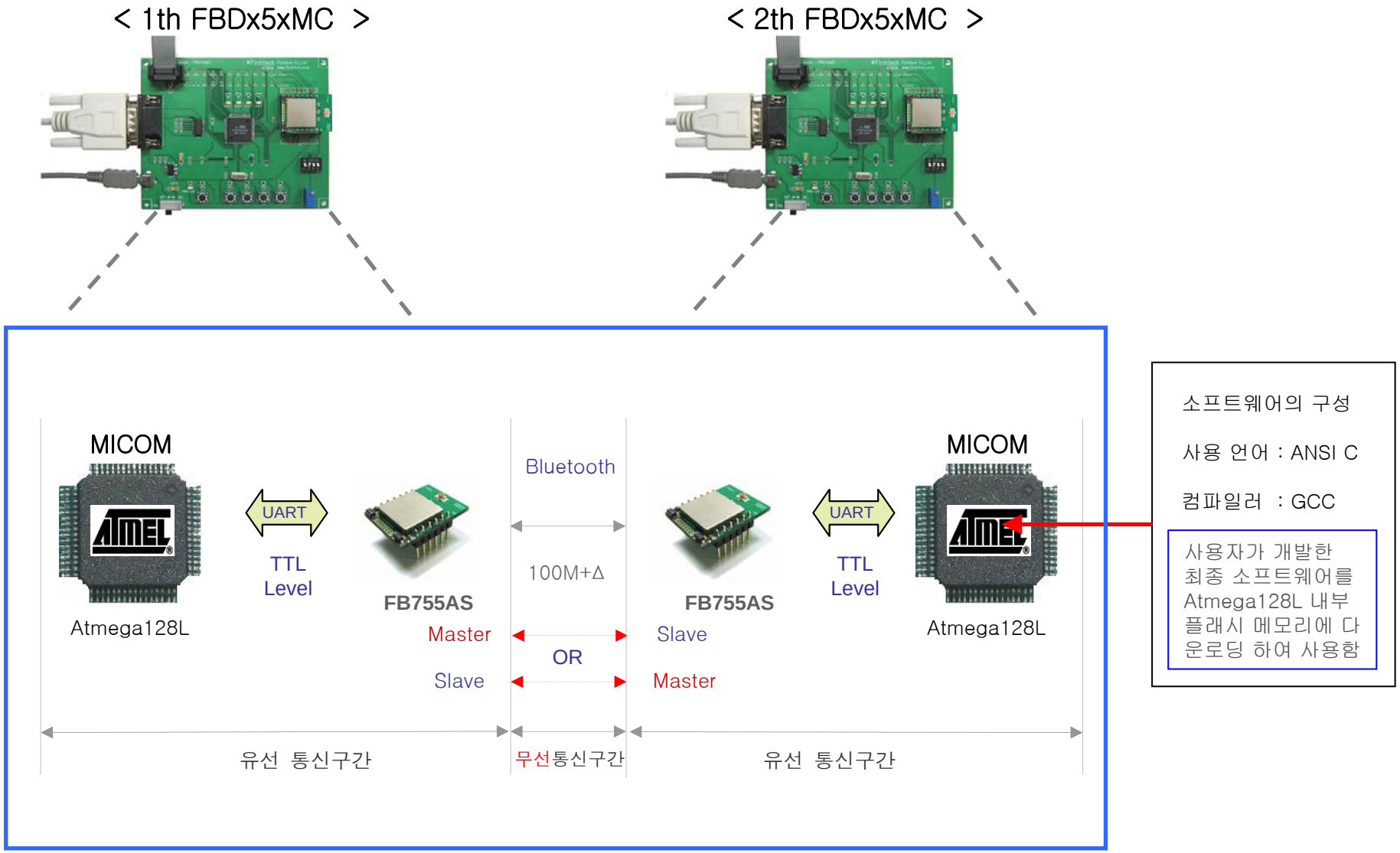
참고 : UART : Universal Asynchronous Receiver Transmitter

MICOM의 경우 현재 시중에 널리 사용되는 ATMega128L 사용

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



2. 전체 구성도

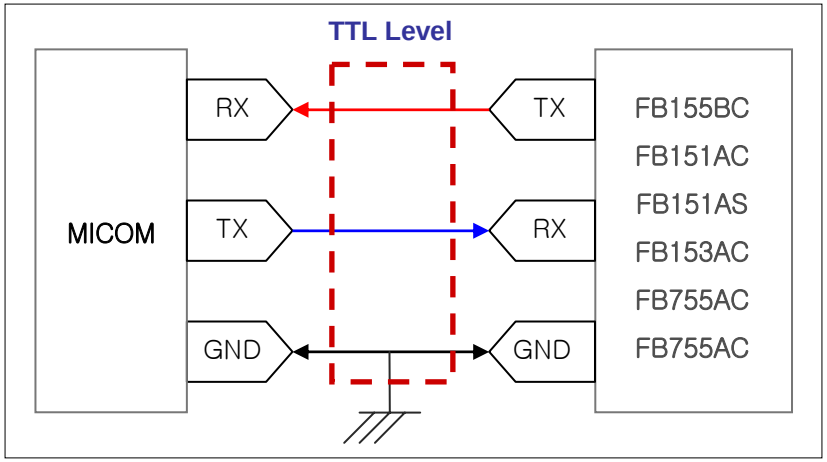

< 1th FBDx5xMC >
< 2th FBDx5xMC >
MICOM
Atmega128L
UART
TTL
Level
FB755AS
Master
Slave
Bluetooth
100M+Δ
OR
Slave
Master
FB755AS
UART
TTL
Level
MICOM
Atmega128L
유선 통신구간
무선통신구간
유선 통신구간
소프트웨어의 구성
사용 언어 : ANSI C
컴파일러 : GCC
사용자가 개발한
최종 소프트웨어를
Atmega128L 내부
플래시 메모리에 다
운로딩 하여 사용함

3. FB755AS 1:N 컨트롤을 위한 인터페이스

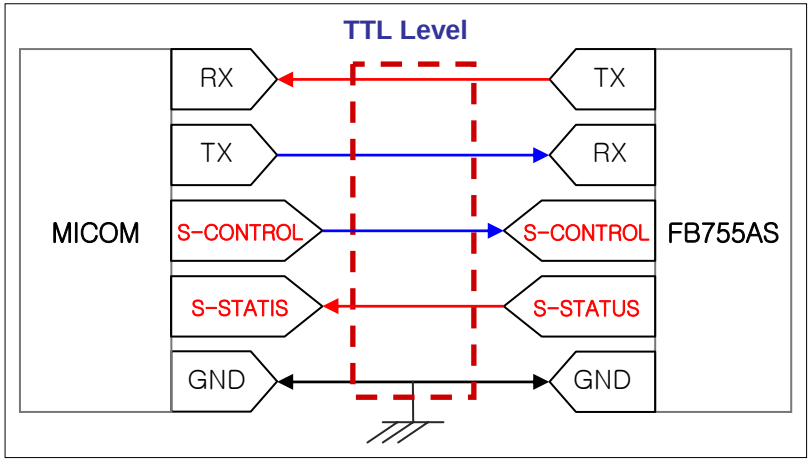
- MICOM과 ㈜펌테크의 Bluetooth Embedded Module을 “기본 인터페이스”로 구성한 경우, 1개의 Master와 1개의 Slave 연결 가능

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

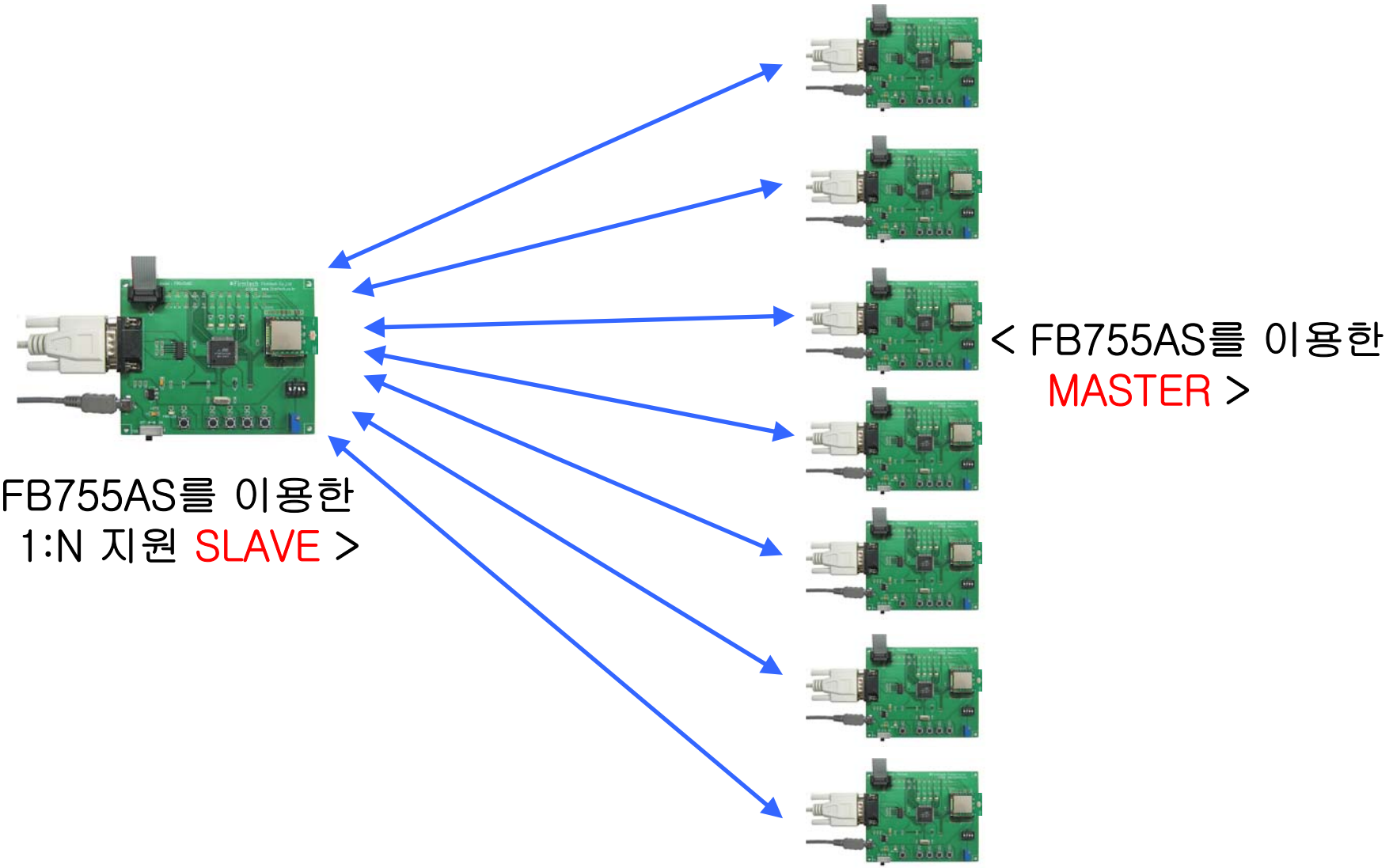
<1> MICOM 과 Bluetooth Embedded Module 과의 인터페이스 방법 (기본 인터페이스)



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



4. FB755AS를 이용한 1:N 실제 구성도



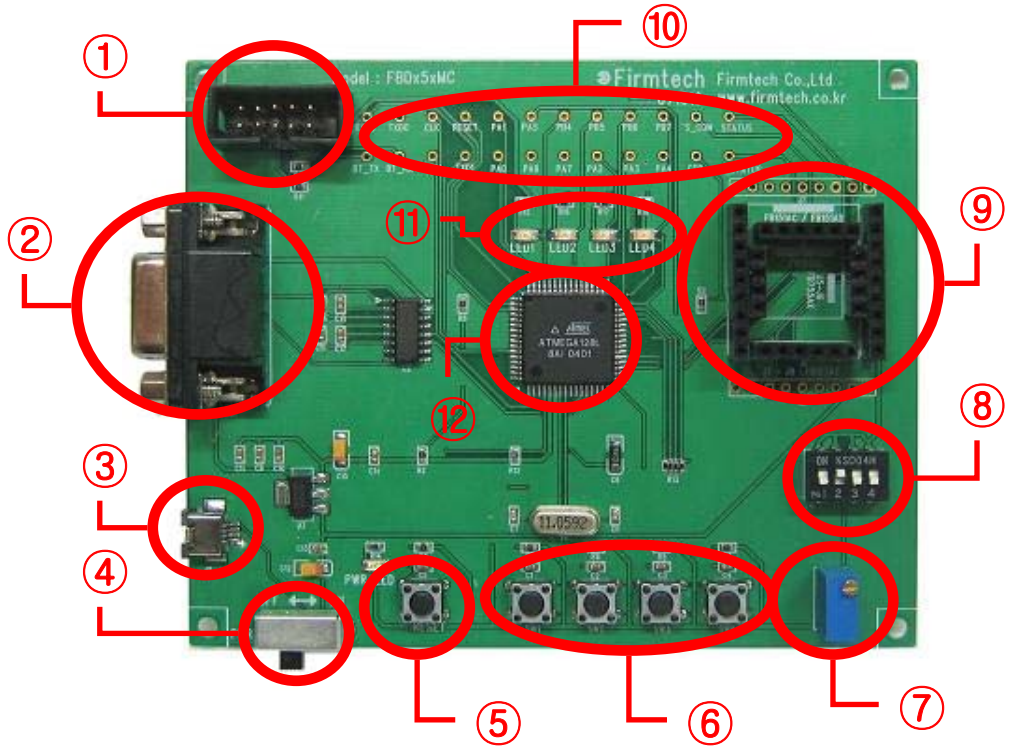
< FB755AS를 이용한
1:N 지원 SLAVE >

< FB755AS를 이용한
MASTER >

5. FBD755MC_gcc 소개

- FBD755MC_gcc는 ㈜펌테크의 FBDx5xMC 보드에 ㈜펌테크의 임베디드 모듈인 FB755AS(1:N 연결 지원)를 장착하여, FB755AS에서 제공되는 AT-Command를 이용하여 1: N Bluetooth 연결 / 데이터 송-수신을 MICOM으로 제어하는 프로그램 소스 명
- 사용하는 MICOM으로는 ATMEL사의 ATMega128L 사용
- 사용하는 컴파일러는 WINAVR 사용
- 실행 파일을 다운로드하기 위해 사용하는 프로그램은 PonyProg 사용
- 시리얼 데이터 송/수신 및 디버깅 포트로 DB-9 사용
- 디버깅을 위한 테스트 포인트 사용
- 전원 입력은 PC의 USB 포트 사용
- 데이터의 입력으로 RS-232 포트 사용
- 데이터의 출력으로 RS-232 포트 사용
- FB755AS N개와 FBDx5xMC 보드 N개 사용(최대 8개 사용, Slave 1개/Master 7개)
- FBD755MC_gcc는 FB755AS의 Slave Bluetooth address를 직접 소스에 입력하여 "소스 컴파일->프로그램 다운로드->Bluetooth 연결->시리얼 데이터 송/수신" 진행
- Slave에서 데이터를 전송할 타겟(Master)은 Switch(4개)를 이용하여 설정

6. FBDx5xMC 제품 외형



< FBDx5xMC 제품 외형 >

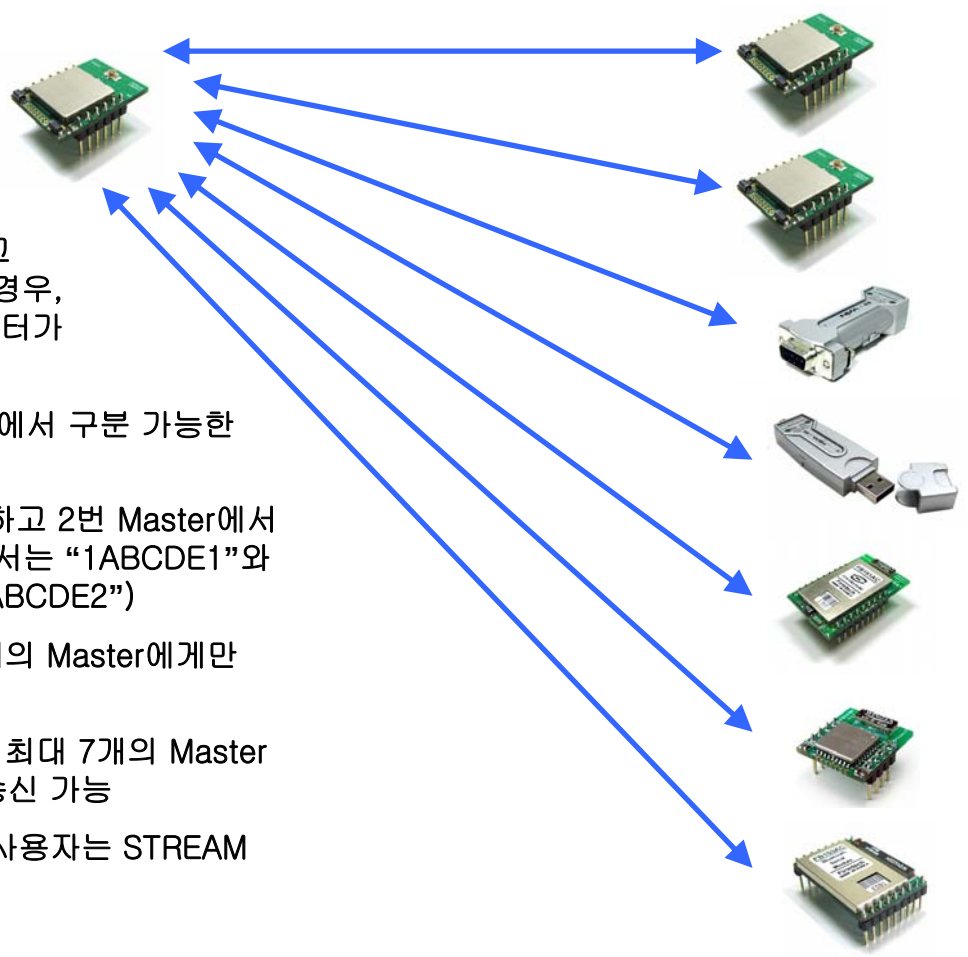
1. AVR Loader 연결 커넥터
2. RS-232 시리얼 연결 커넥터
3. USB 전원 연결 커넥터
4. 전원 ON/OFF 스위치
5. 리셋 스위치 (ATMega128L 리셋 용)
6. 스위치 (데이터 입력 용)
7. 가변 저항 (데이터 입력 용)
8. 딥 스위치 (데이터 입력 용)
9. 임베디드 제품 연결 커넥터
10. 테스트 포인트 (디버거 용)
11. LED (데이터 출력 용)
12. ATMega128L (프로그램용 MICOM)

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. FB755AS를 이용한 1:N 개념의 이해

1. 1개의 Slave에 최대 7개의 Master 동시 연결 가능 1 (Slave) : N (Master : 최대 7개)
2. Master Device 수시 변경 가능
3. Master Device는 FB755AS가 아닌 타 Bluetooth 장치 사용 가능
4. 1번 Master에서 송신한 “ABCDE”라는 데이터는 FB755AS에서 그대로 “ABCDE”라고 수신됨
   - 1번 Master에서 “ABCDE”라는 데이터를 송신하고 2번 Master에서 “ABCDE”라는 데이터를 송신한 경우, FB755AS에서는 “ABCDE”와 “ABCDE”라는 데이터가 그대로 수신됨 (“ABCDEABCDE”)
   - 사용자는 Master에서 데이터를 송신 할 때 Slave에서 구분 가능한 형태로 송신해야 함
   - 1번 Master에서 “1ABCDE1”라는 데이터를 송신하고 2번 Master에서 “2ABCDE2”라는 데이터를 송신하면 FB755AS에서는 “1ABCDE1”와 “2ABCDE2”라는 데이터가 수신됨 (“1ABCDE12ABCDE2”)
5. FB755AS에서 Master로 데이터를 송신할 때, 한번에 1개의 Master에게만 데이터 송신 가능
   - STREAM 채널이라는 것을 사용하여, FB755AS는 최대 7개의 Master 중에서 선별적으로 1개의 Master에게만 데이터 송신 가능
   - 7개의 Master에 데이터를 모두 송신하기 위해서 사용자는 STREAM 채널을 컨트롤 하여 7번의 데이터를 송신해야 함



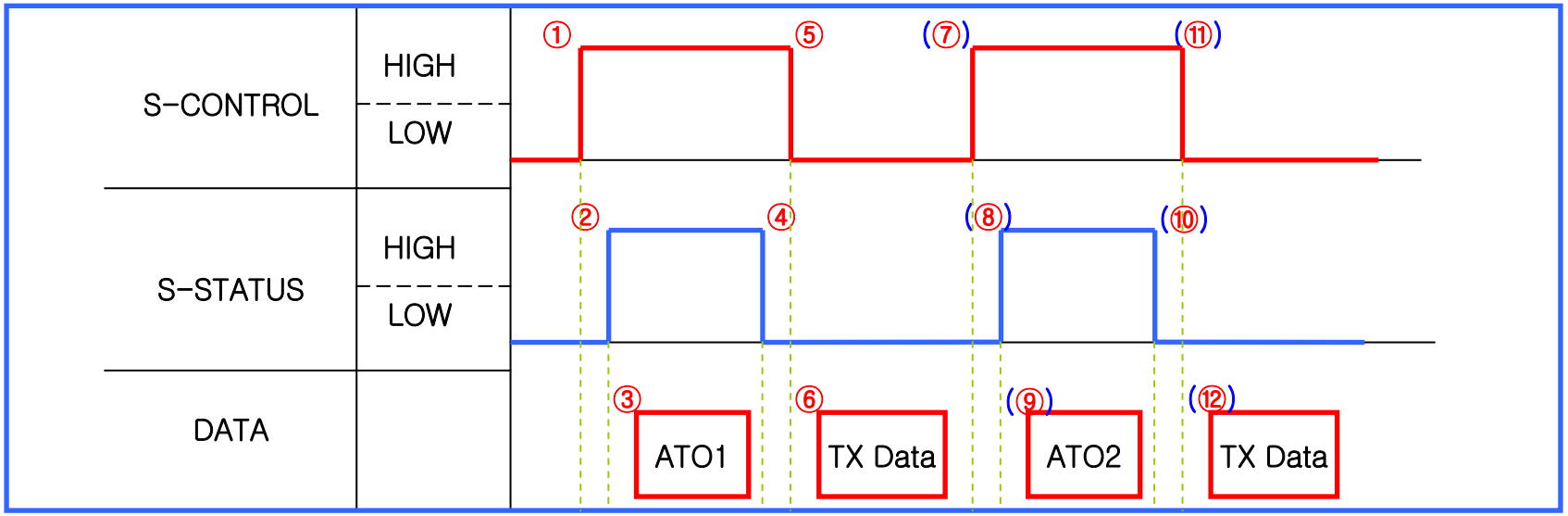
9. STREAM 채널의 이해

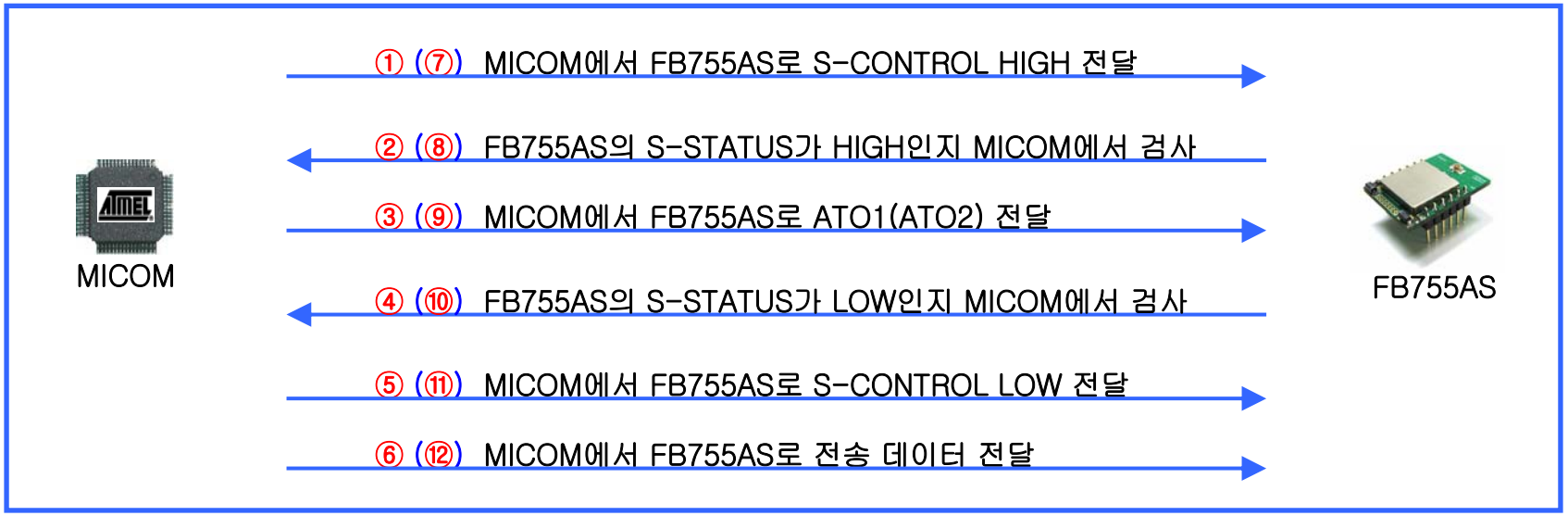
1. STREAM 채널은 7개의 Master 동시 연결과 별개의 개념
   - STREAM 채널을 해제하거나 변경해도, 7개의 Master와 동시적으로 연결된 상태에 아무런 영향 없음
2. STREAM 채널은 Master에서 송신한 데이터를 Slave(FB755AS)에서 수신하는 것과 별개의 개념
   - STREAM 채널을 해제하거나 변경해도, 7개의 Master에서 송신한 데이터가 Slave(FB755AS)에서 수신되는 것에 아무런 영향 없음 (OP_MODE1: 모니터링 동작)
3. STREAM은 FB755AS에서 7개의 Master중 1개의 Master를 선택하여 데이터를 송신하기 위한 채널의 개념
   - FB755AS에서 1번 Master로 데이터를 송신하기 위해서는 STREAM 채널을 1로 설정해야 함
     - FB755AS는 첫 번째 연결된 Master를 1번 Master로 설정
     - 1번 Master로 STREAM 채널을 변경하기 위한 Command : ATO1
   - FB755AS에서 4번 Master로 데이터를 송신하기 위해서는 STREAM 채널을 4로 설정해야 함
     - FB755AS는 4 번째 연결된 Master를 4번 Master로 설정
     - 4번 Master로 STREAM 채널을 변경하기 위한 Command : ATO4
   - FB755AS는 Broadcast 개념이 존재하지 않음
     - FB755AS에서 다수의 Master에 데이터를 송신하기 위해서, 사용자는 STREAM 채널을 변경하고 데이터를 입력하는 작업을 다수의 Master 수 만큼 진행 해야 함
   - FB755AS는 7개의 Master로부터 데이터를 동시에 수신 하면서 1개의 Master로 데이터를 송신함 (OP_MODE1 : 모니터링 동작)
   - STREAM 채널 변경 작업을 하기 위해 FB755AS의 S-CINTROL 포트와 S-STATUS 포트 사용
   - STREAM 채널 변경 작업을 하기 위해 FB755AS의 ATOn Command 사용

10. FB755AS를 사용하기 전에 알아야 할 사항

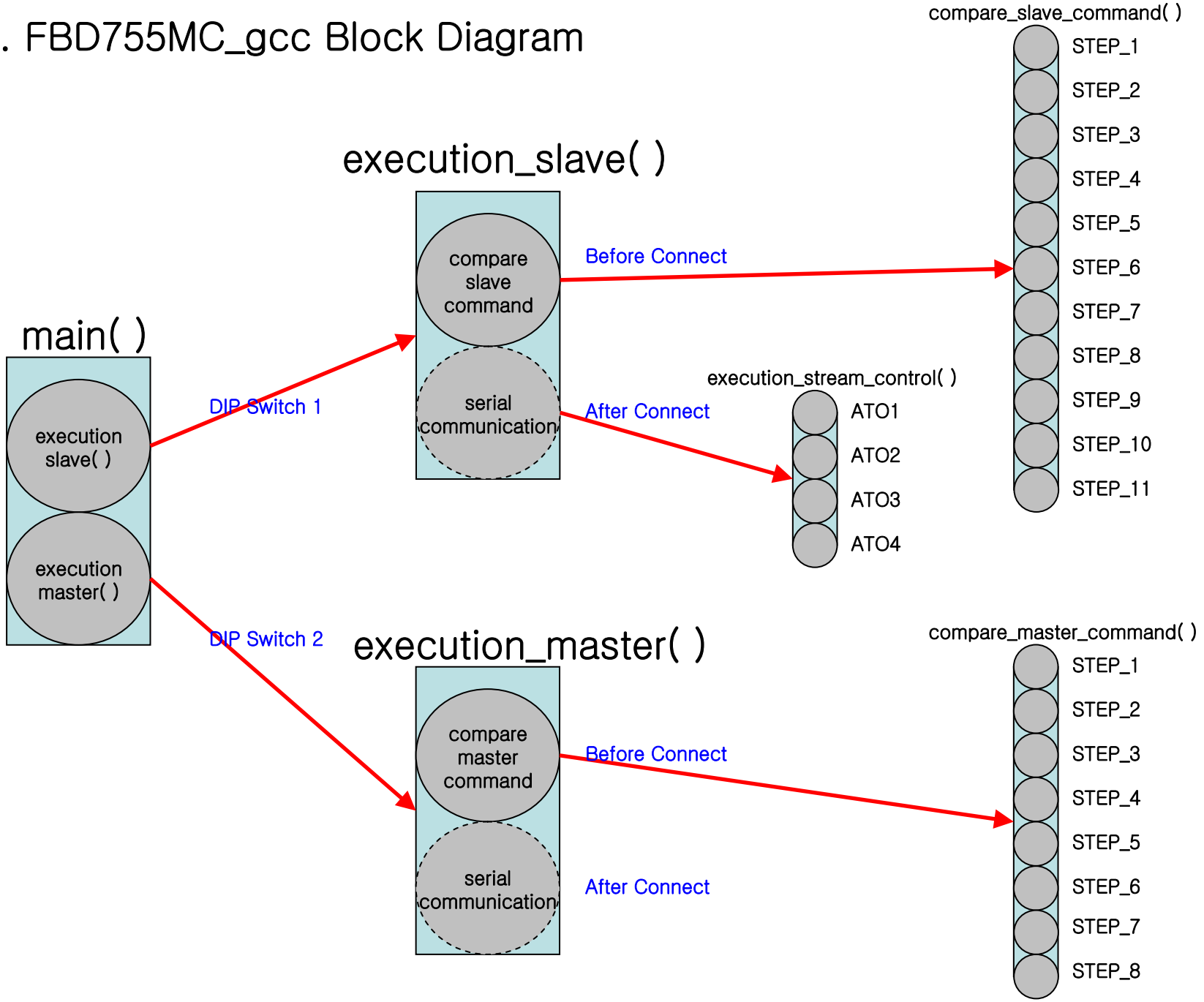
- FB755AS를 이용한 Master/Slave 연결 방법
  - ✓ FB755AS는 4가지의 Master/Slave 연결 방법 제공 (Connection Mode)
  - ✓ FBD755MC_gcc는 FB755AS 연결 방법 중 CNT_MODE4 (AT-Command를 이용한 연결) 사용
  - ✓ Connection Mode에 대한 자세한 사항은 ㈜펌테크 홈페이지에서 FB755AS 매뉴얼 참고
- FB755AS를 이용한 Master/Slave 데이터 송/수신 방법
  - ✓ FB755AS는 3가지의 데이터 송/수신 방법 제공 (Operation Mode)
  - ✓ FBD755MC_gcc는 FB755AS 데이터 송/수신 방법 중 OP_MODE1 (모니터링 방식) 사용
  - ✓ Operation Mode에 대한 자세한 사항은 ㈜펌테크 홈페이지에서 FB755AS 매뉴얼 참고
- FB755AS Device Number 설정
  - ✓ FB755AS를 1:N으로 사용하기 위해 연결될 Master Device 수 설정
  - ✓ FB755AS(Slave)는 설정된 Master Device 수에 따라 SCAN (Master에서 Slave를 검색할 수 있는 기능) 동작의 회수가 정해짐
  - ✓ 설정 가능한 최대 Device 수 : 7
- FB755AS Buffer Size 설정
  - ✓ FB755AS는 1:N으로 사용할 때 원활한 데이터의 송/수신을 위해 데이터 송/수신 버퍼 사이즈 설정
  - ✓ FBD755MC_gcc에서 사용하는 버퍼 사이즈 : 0
  - ✓ 버퍼 사이즈에 대한 자세한 사항은 ㈜펌테크 홈페이지에서 FB755AS 매뉴얼 참고
- FB755AS를 1:N으로 사용하기 위한 하드웨어적인 구성
  - ✓ FB755AS를 1:N으로 사용하기 위해서 2개의 포트 사용(S-CONTROL, S-STATUS)
  - ✓ S-CONTROL : MICOM에서 데이터를 보낼 타겟(Master)을 선택하기 위해서 FB755AS로 S-CONTROL 신호를 내 보냄
  - ✓ S-STATUS : MICOM으로부터 S-CONTROL 신호를 받은 FB755AS가 명령어(ATOn)를 받을 준비가 된 경우MICOM으로 S-STATUS 신호를 내 보냄
  - ✓ MICOM에서 S-STATUS 신호를 받으면 FB755AS로 ATO1~ATO7까지의 명령어를 보낼 수 있음

11. FB755AS Slave에서 Master로 데이터를 송신하기 위한 타이밍도





12. FBD755MC_gcc Block Diagram



13. compare_slave_command( ) STEP Description (1)

step_slave_bt	Description 1th	Description
STEP_1	Bluetooth Device로부터 최초 수신한 메시지 비교 루틴	• Bluetooth Device로부터 “BTWIN Slave mode start”메시지가 수신되었는지 검사 – Bluetooth Device가 Slave로 시작된 경우, 다음 스텝 STEP_3 설정 • Bluetooth Device로부터 “BTWIN Master mode start”메시지가 수신되었는지 검사 – Bluetooth Device가 Master로 시작된 경우, 다음 스텝 STEP_2 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_2	Bluetooth Device Role 변경 루틴 1번째	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – Bluetooth Device가 “Master”로 시작된 후 “OK” 메시지 수신 – “at+btrole=s₩r” Command 이용하여 Bluetooth Device를 Slave로 설정 – 다음 스텝 STEP_3 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_3	Bluetooth Device 데이터 송/수신 방법 설정 루틴	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – Bluetooth Device가 Slave로 시작 또는 Role이 변경된 후 “OK” 메시지 수신 – “at+btopmode,1₩r” Command 이용하여 Bluetooth Device 데이터 송/수신 방법 설정 – 다음 스텝 STEP_4 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_4	Bluetooth Device 에 연결될 Master Device 수 설정	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – “at+btopmode,1₩r” Command 이용하여 데이터 송/수신 방법이 설정된 경우 “OK”메시지 수신 – 최대 설정 가능한 Device 수인 7보다 큰지 검사 – “at+btdev=n” Command 이용하여 Bluetooth Device에 연결될 Master Device 수 설정 – 다음 스텝 STEP_5 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_5	Bluetooth Device 버퍼 사이즈 설정	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – “at+btdev=n₩r” Command 이용하여 Bluetooth Device에 연결될 Device수 설정된 경우 “OK” 메시지 수신 – “at+btbuff=0₩r” Command 이용하여 Bluetooth Device Buffer size 설정 – 다음 스텝 STEP_6 설정 • 비교 데이터가 틀린 경우 error루틴 처리

14. compare_slave_command( ) STEP Description (2)

step_slave_bt	Description 1th	Description
STEP_6	Bluetooth Device 설정 적용을 위한 리셋 진행	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – “at+btbuff=0₩r” Command 이용하여 Bluetooth Device Buffer size 설정된 경우 “OK”메시지 수신 – 각 스텝 별로 설정한 값 적용하기 위해 “atz₩r” Command 이용하여 Bluetooth Device Reset – 다음 스텝 STEP_7 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_7	Bluetooth Device Slave Start 수신 메시지 비교 루틴	• Bluetooth Device로부터 “BTWIN Slave mode start”메시지가 수신되었는지 검사 – Bluetooth Device가 Slave로 시작된 경우, 다음 스텝 STEP_8 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_8	Bluetooth Device 검색/연결 대기 설정 루틴 1번째	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – Bluetooth Device가 Slave로 시작된 경우 “OK” 메시지 수신 – “at+btscan₩r” Command 이용하여 Bluetooth Device를 검색/연결 대기 상태로 설정 – 다음 스텝 STEP_9 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_9	Bluetooth Device 검색/연결 대기 설정 루틴 2번째	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – “at+btscan₩r” Command 이용하여 Bluetooth Device를 검색/연결 대기 상태로 설정한 경우 “OK” 메시지 수신 – 다음 스텝 STEP_10 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_10	Bluetooth Device 연결 상태 체크 루틴	• Bluetooth Device로부터 “CONNECT”메시지가 수신되었는지 검사 – Slave가 검색/연결 대기 상태인 경우, Master에서 검색/연결 가능 – Slave가 Master와 연결된 경우, “CONNECT”메시지 수신 – STEP_4에서 설정한 Device 수만큼 “CONNECT”메시지를 받을 때까지 STEP_10 반복 – STEP_4에서 설정한 Device 수만큼 “CONNECT”메시지를 받은 경우, 데이터 송/수신 루틴 진행 (루틴상 다음 스텝 STEP_11 설정) • 비교 데이터가 틀린 경우 error루틴 처리

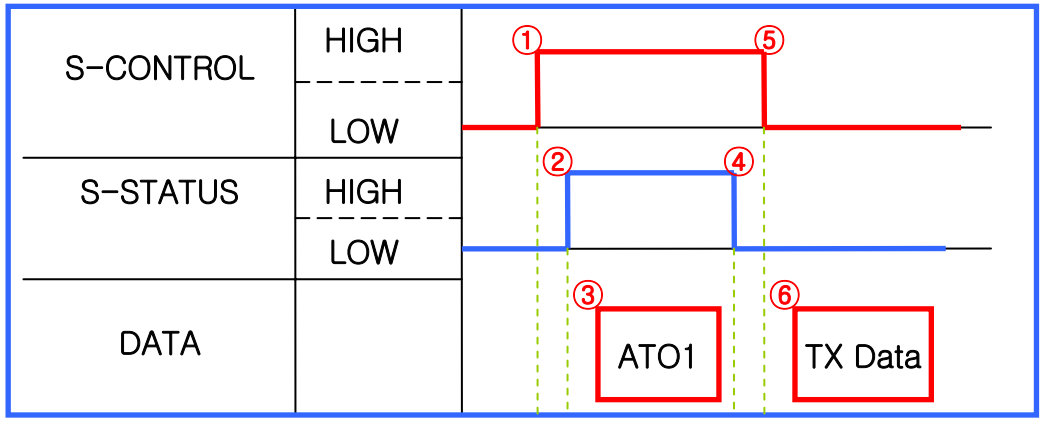
15. compare_master_command( ) STEP Description (1)

step_slave_bt	Description 1th	Description 2th
STEP_1	Bluetooth Device로부터 최초 수신한 메시지 비교 루틴	• Bluetooth Device로부터 “BTWIN Slave mode start”메시지가 수신되었는지 검사 – Bluetooth Device가 Slave로 시작된 경우, 다음 스텝 STEP_2 설정 • Bluetooth Device로부터 “BTWIN Master mode start”메시지가 수신되었는지 검사 – Bluetooth Device가 Master로 시작된 경우, 다음 스텝 STEP_3 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_2	Bluetooth Device Role 변경 루틴	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – Bluetooth Device가 “Slave”로 시작된 후 “OK” 메시지를 수신 – “at+btrole=m₩r” Command 이용하여 Bluetooth Device를 Master로 설정 – 다음 스텝 STEP_3 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_3	Bluetooth Device 데이터 송/수신 방법 설정 루틴	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – Bluetooth Device가 Master로 시작/설정 된 후 “OK”메시지를 수신 – “at+btopmode,0₩r” Command 이용하여 Bluetooth Device 데이터 송/수신 방법 설정 (Master와 Slave의 데이터 송/수신 방법은 다름) – 다음 스텝 STEP_4 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_4	Bluetooth Device 설정 적용을 위한 리셋 진행	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – “at+btopmode,0₩r” Command 이용하여 Bluetooth Device 데이터 송/수신 방법 설정된 경우 “OK”메시지 수신 – 각 스텝 별로 설정한 값 적용하기 위해 “atz₩r” Command 이용하여 Bluetooth Device Reset – 다음 스텝 STEP_5 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_5	Bluetooth Device Master Start 수신 메시지 비교 루틴	• Bluetooth Device로부터 “BTWIN Master mode start”메시지가 수신되었는지 검사 – Bluetooth Device가 Master로 시작된 경우, 다음 스텝 STEP_6 설정 • 비교 데이터가 틀린 경우 error루틴 처리

16. compare_master_command( ) STEP Description (2)

step_slave_bt	Description 1th	Description 2th
STEP_6	Bluetooth Device 연결 요청 루틴 1번째	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – Bluetooth Device가 Master로 시작된 후 “OK”메시지 수신 – “atd” Command와 저장되어 있는 Slave의 어드레스를 이용하여 연결 요청 진행 – 다음 스텝 STEP_7 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_7	Bluetooth Device 연결 요청 루틴 2번째	• Bluetooth Device로부터 “OK”메시지가 수신되었는지 검사 – “atd” Command와 Slave의 어드레스를 이용하여 연결 요청이 진행되는 경우 “OK”메시지 수신 – 다음 스텝 STEP_8 설정 • 비교 데이터가 틀린 경우 error루틴 처리
STEP_8	Bluetooth Device 연결 상태 체크 루틴	• Bluetooth Device로부터 “CONNECT”메시지가 수신되었는지 검사 – Master가 Slave와 연결을 하기 위해서 Slave는 검색/연결 대기 상태이어야 함 – Master가 Slave와 연결된 경우 “CONNECT”메시지 수신 – Master와 Slave가 연결된 경우, 데이터 송/수신 루틴 진행(루틴상 다음 스텝 STEP_9 설정) • Bluetooth Device로부터 “ERROR”메시지가 수신되었는지 검사 – Master가 Slave와 연결되지 못한 경우 “ERROR”메시지 수신 – Master와 Slave가 연결되지 못한 경우, “atd” Command와 Salve의 어드레스를 이용하여 재 연결 요청 진행 – 다음 스텝 STEP_7 설정 • 비교 데이터가 틀린 경우 error루틴 처리

17. execution_stream_control( ) STEP Description



< SWITCH에 따른 전달 Command >

1. SWTICH 1인 경우 “ATO1” 전달
2. SWITCH 2인 경우 “ATO2” 전달
3. SWITCH 3인 경우 “ATO3” 전달
4. SWITCH 4인 경우 “ATO4” 전달

STEP	Description 1th	Description 2th
①	연결되어 있는 STREAM 해제 요청	• MICOM에서 FB755AS의 S-CONTROL 포트에 HIGH 신호 전달(Switch 사용) • FB755AS에 연결되어 있는 STREAM을 변경하기 위해, 현재의 STREAM 해제 요청 신호
②	STREAM 해제 OK Command 수신 가능 상태	• FB755AS의 S-STATUS 포트에서 HIGH 신호 출력 • FB755AS에 연결되어 있는 STREAM이 해제되었음을 나타내는 신호 • FB755AS가 STREAM 채널 변경에 대한 Command 수신 가능 상태를 나타냄
③	STREAM 변경 Command 전송	• MICOM에서 FB755AS로 ATOn Command를 전달하여 STREAM 채널 변경 • ATO1 : 1번째 채널과 STREAM 구간 형성 요청 • ATO7 : 7번째 채널과 STREAM 구간 형성 요청
④	Command 수신 OK STREAM 연결 OK	• FB755AS의 S-STATUS 포트에서 LOW 신호 출력 • FB755AS가 STREAM 채널 변경에 대한 Command를 올바르게 수행한 경우를 나타냄
⑤	S-CONTROL LOW	• MICOM에서 FB755AS의 S-CONTROL 포트에 LOW 신호 전달 • 다음 STREAM 해제 요청을 위해 LOW 상태로 변경
⑥	데이터 송신	• 새로 형성된 STREAM 채널로 Slave는 데이터 송신 가능

18. 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);
        if(read_avr_port == DIP_1)
            execution_slave();
        else if(read_avr_port == DIP_2)
            execution_master();
        else if(read_avr_port > 0x70 && read_avr_port <= 0xf0)
        {
            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
3. 2 번 딥 스위치가 ON 이면 마스터 루틴 진행
4. 4 번 딥 스위치가 ON이면 Address 읽기 루틴 진행

19. 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;
            uart0_length = Check_Rx_Buf0();
            if(uart0_length)
            {
                for(i = 0;i < uart0_length;i++)
                    PutChar1(GetChar0());
            }
            execution_stream_control();
            uart1_length = Check_Rx_Buf1();
            if(uart1_length)
            {
                for(i = 0;i < uart1_length;i++)
                {
                    if(stream_status == OK)
                        PutChar0(GetChar1());
                    else
                        GetChar1();
                }
            }
        } ? end if status_sequence==AFTE... ?
    } ? 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 2개 인 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. Bluetooth로부터 수신 받은 데이터가 있는 경우 호스트(PC)로 전달
2. 스트림 제어 함수 호출
3. 호스트(PC)로부터 수신 받은 데이터가 있는 경우, 스트림 상태 값이 OK인 경우 Bluetooth로 데이터 전달, 스트림 상태 값이 ERROR인 경우 데이터 무시

20. execution_master( ) Source

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

    //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;
            uart0_length = Check_Rx_Buf0();
            if(uart0_length)
            {
                for(i = 0;i < uart0_length;i++)
                    PutChar1(GetChar0());
            }
            uart1_length = Check_Rx_Buf1();
            if(uart1_length)
            {
                for(i = 0;i < uart1_length;i++)
                    PutChar0(GetChar1());
            }
        }
    } ? 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. Bluetooth로부터 수신 받은 데이터가 있는 경우 호스트(PC)로 전달
2. 호스트(PC)로부터 수신 받은 데이터가 있는 경우 Bluetooth로 전달

21. 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

22. 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

23. 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 : 0]");
        wait_1ms(100);
        DispStr0("at+btbuff=0\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 0으로 설정
3. 비교 데이터가 틀린 경우 execution_error()루틴 진행

24. 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

25. 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

26. main.h

```
/*
*****************************************
Define Part
*****************************************
*/
#define READ_SWITCH             0x01
#define READ_DIP                0x02
//
#define WRITE_LED_1_ON          0x01
#define WRITE_LED_1_OFF         0x02
#define WRITE_LED_2_ON          0x03
#define WRITE_LED_2_OFF         0x04

//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
```

< FB755AS의 연결 디바이스 수 설정을 위한 값 >

1. main.h 파일에 연결 디바이스 수 설정 값 Define
2. #define CONNECT_DEV_NUM 0x33
   - 연결 디바이스 수를 3개로 설정
   - 헥사 값 0x33은 10진수 캐릭터 값 3과 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)

27. execution_stream_control( )

```
void execution_stream_control(void)
{
    unsigned char temp_read_avr_port;

    read_avr_port = read_port_value(READ_SWITCH);
    temp_read_avr_port = read_avr_port;
    if(read_avr_port == SWITCH_1||read_avr_port == SWITCH_2||
       read_avr_port == SWITCH_3||read_avr_port == SWITCH_4)
    {
        if(old_switch_value != read_avr_port)
        {
            old_switch_value = read_avr_port;
            wait_1ms(10);
            write_port_value(WRITE_S_CONTROL_HIGH);
            execution_check_s_status_high();
            if(s_status_value == OK)
            {
                switch(temp_read_avr_port)
                {
                    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]");
                }
            } ? end if s_status_value==OK ?
            else
            {
                write_port_value(WRITE_S_CONTROL_LOW);
                stream_status = ERROR;
                DispStr1("[STREAM STATUS ERROR]");
            }
        } ? end if old_switch_value!=rea... ?
    } ? 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 temp_read_avr_port에 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 >

1. 이전에 눌린 스위치와 현재 눌린 스위치를 비교 하여 다른 경우 새로운 STREAM 채널 변경 적용
2. 만약, 이전에 눌린 스위치와 현재 눌린 스위치가 같으면 기존의 STREAM 채널 그대로 적용하고 이전 루틴으로 복귀
3. 현재 눌린 스위치 저장 (old_switch_value)
4. S-CONTROL에 HIGH 출력
5. S-STATUS가 HIGH가 되는지 검사 (execution_check_s_status_high())
6. 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
7. S-STATUS가 LOW가 되는지 검사 (execution_check_s_status_low())
8. s_status_value가 ERROR인 경우
   - S-CONTROL LOW 설정
   - STREAM 채널 변경 실패 설정
9. s_status_value가 OK인 경우
   - S-CONTROL LOW 설정
   - STREAM 채널 변경 완료 설정

28. execution_check_s_status_high( )

```
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)

29. execution_check_s_status_low( )

```
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)

30. 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

31. 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

32. 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

33. 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;
        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;
} ? end switch step_master_bt ?
init_uart0_read_data();
```

< 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 송/수신 루틴 시작
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

34. 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)

35. 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

36. 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

37. 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

38. 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

39. 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

40. 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

41. 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

42. Interrupt Vector

```
SIGNAL(SIG_UART0_RECV)
{
    rx0_buf[p_rx0_wr++] = UDR0;
    if(rx0_buf[p_rx0_wr-1] == LF_VALUE)lf_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)

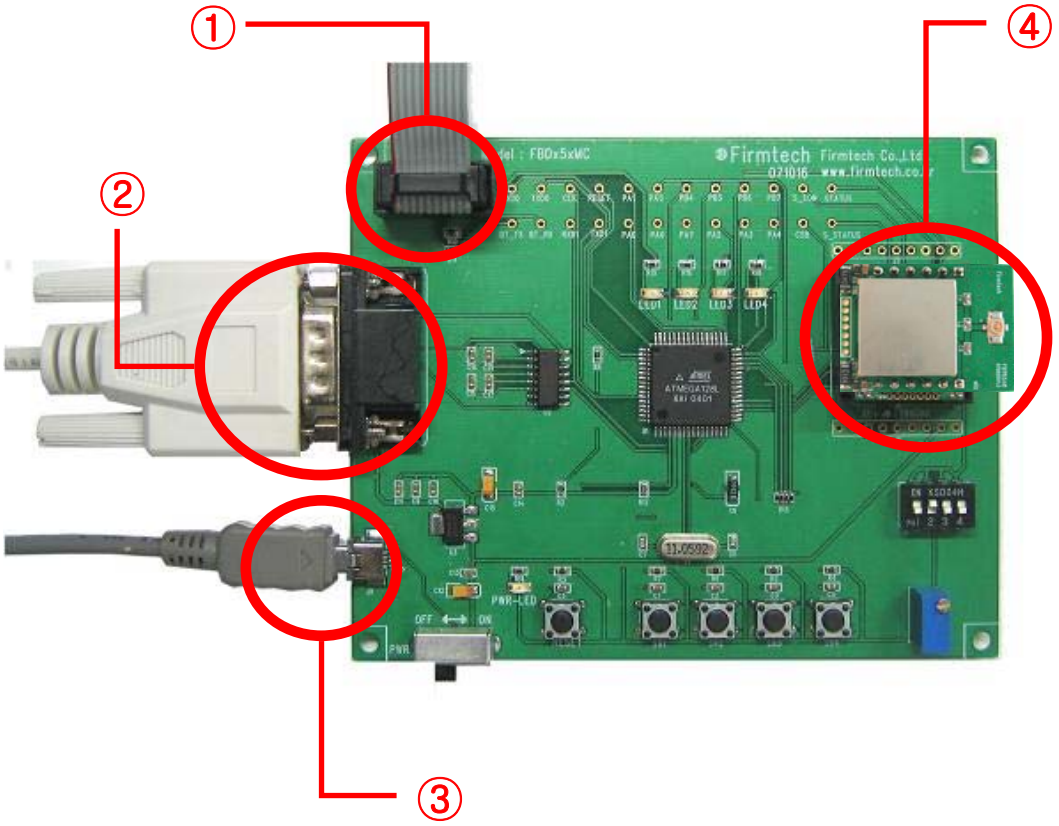
```
SIGNAL(SIG_OVERFLOW0)
{

    timer0_counter++;
    if (timer0_counter > 200 )
    {
        timer0_counter = 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

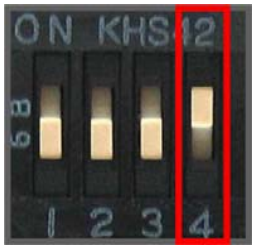
43. FBDx5xMC 보드의 올바른 연결



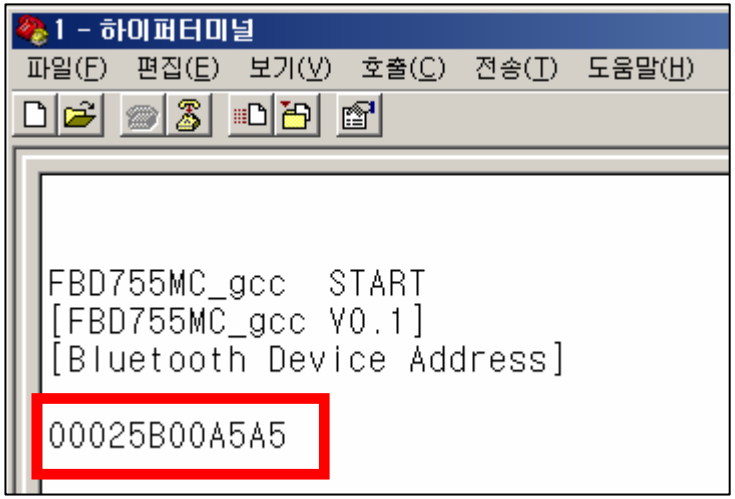
< 제품의 올바른 연결 >

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)

44. 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

45. FBD755MC_gcc 프로그램 수정 (Slave Address 수정)

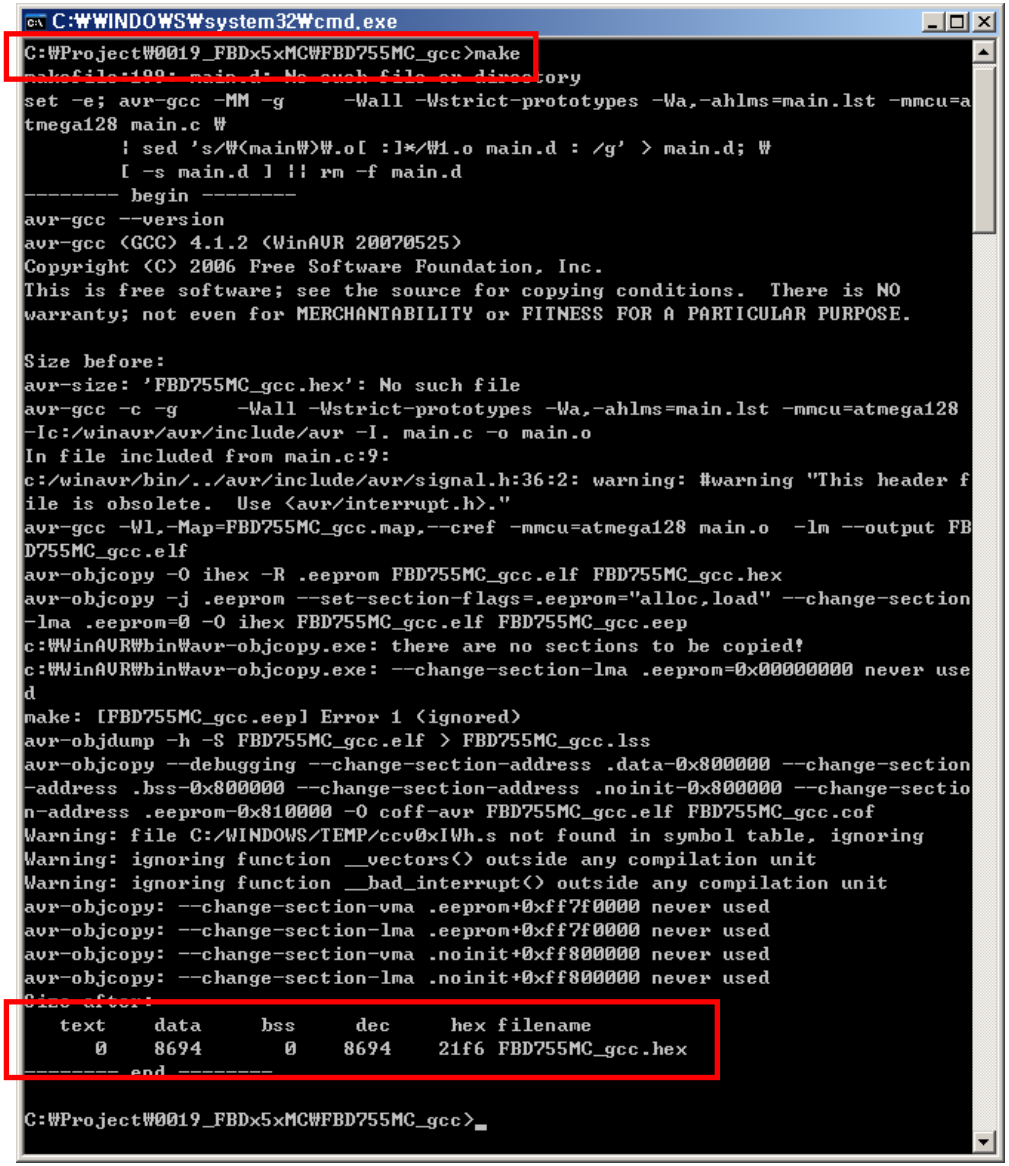
```
void execution_master(void)
{
    unsigned int i;

    //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;
            uart0_length = Check_Rx_Buf0();
            if(uart0_length)
            {
                for(i = 0;i < uart0_length;i++)
                    PutChar1(GetChar0());
            }
            uart1_length = Check_Rx_Buf1();
            if(uart1_length)
            {
                for(i = 0;i < uart1_length;i++)
                    PutChar0(GetChar1());
            }
        }
    } ? 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

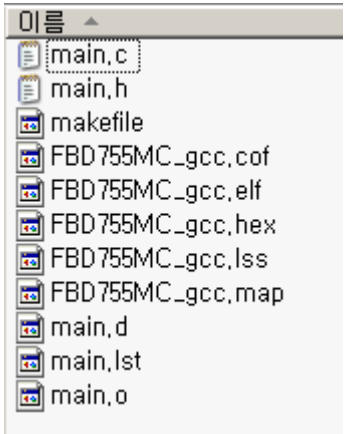
46. FBD755MC_gcc 프로그램 컴파일 (WINAVR 컴파일러 사용)



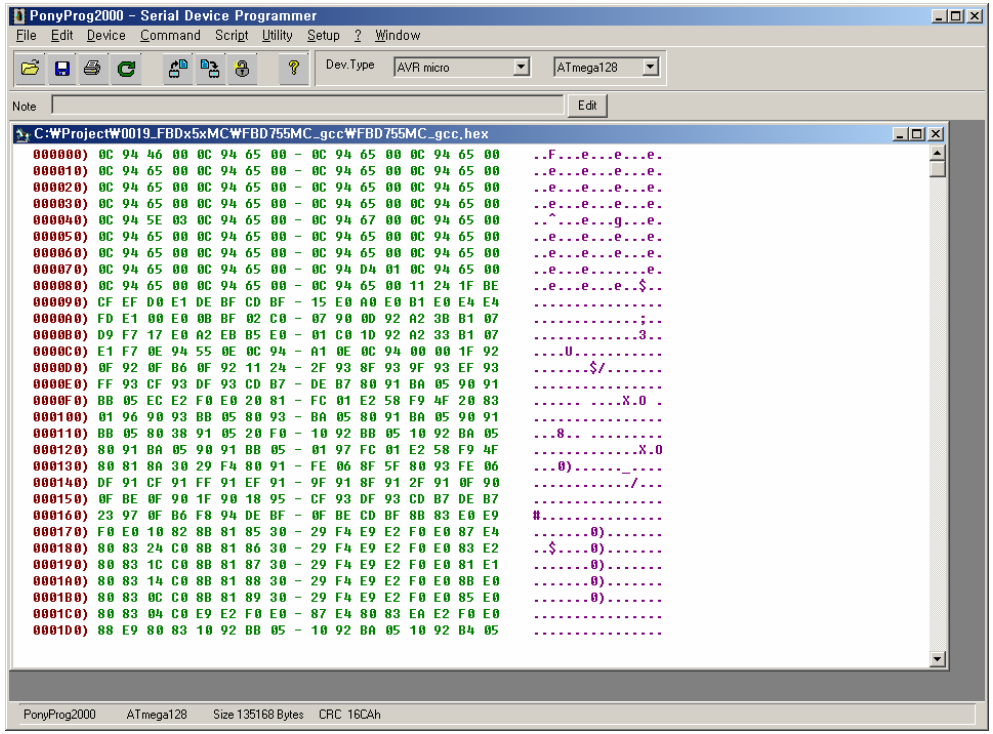
1. 사용 컴파일러
   - 사용하는 컴파일러는 WINAVR 사용
   - http://winavr.sourceforge.net 에서 다운로드 하여 설치
2. 컴파일에 사용하는 파일
   - main.c
   - main.h
   - makefile
3. 컴파일 방법
   - Window Command 창을 이용하여 컴파일에 사용할 파일이 있는 위치로 이동
   - Make 입력 후 Enter Key 입력
   - ERROR 없이 진행된 경우, HEX 파일과 기타 파일 생성

< make 실행 이전 파일>

< make 실행 이후 파일>



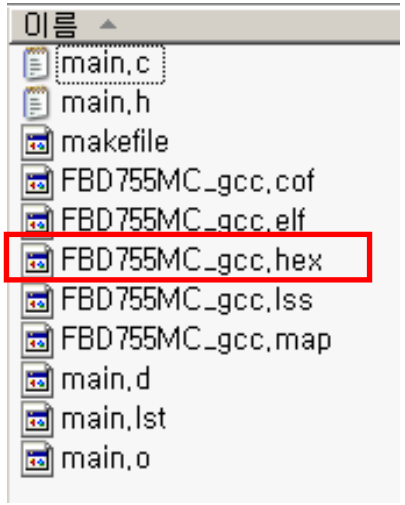
47. FBDx5xMC 보드에 프로그램 다운로드



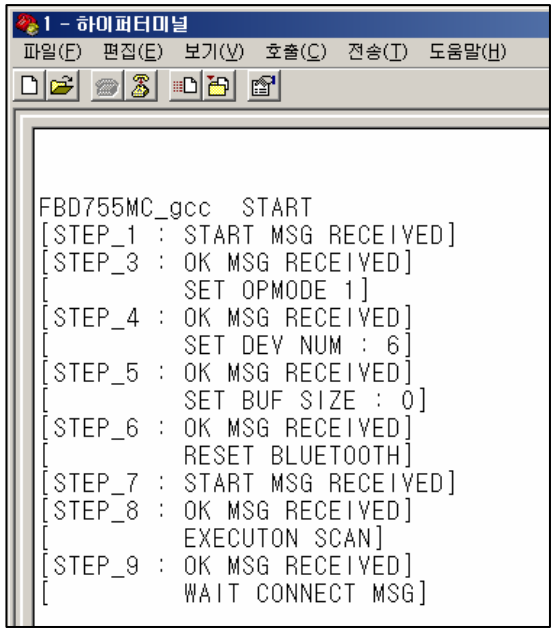
< HEX 파일 다운로드 프로그램 PonyProg>

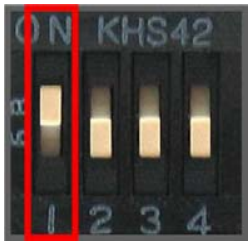
1. Slave Bluetooth Device Address를 확인하고, 확인한 Slave Bluetooth Device Address를 소스에 입력 후 컴파일 진행
   - WINAVR을 이용하여 컴파일 진행
2. 컴파일에 의해 생성된 HEX 파일을 FBDx5xMC 보드에 다운로드
   - 다운로드 프로그램은 PonyProg 사용
   - http://www.lancos.com/ppwin95.html 에서 다운로드 하여 설치
   - AVR Loader로 PC와 FBDx5xMC 보드 연결
   - FBDx5xMC 보드에 생성된 HEX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

< 다운로드 할 HEX 파일>



48. FBDx5xMC 보드의 Slave 설정 & 최초 동작 화면



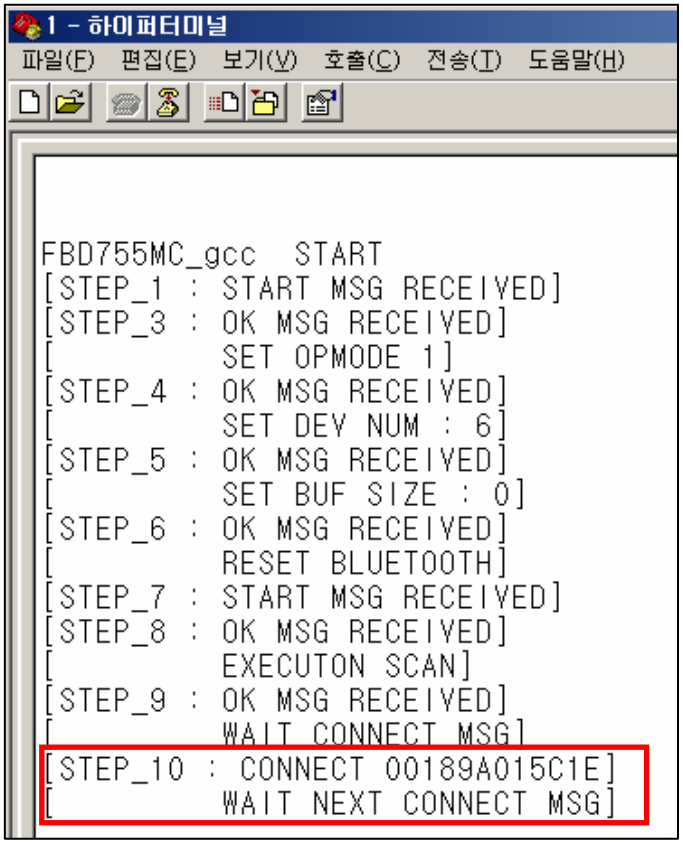


< 딥 스위치 1번 ON 상태 >

< Slave 설정 & 동작 화면 >

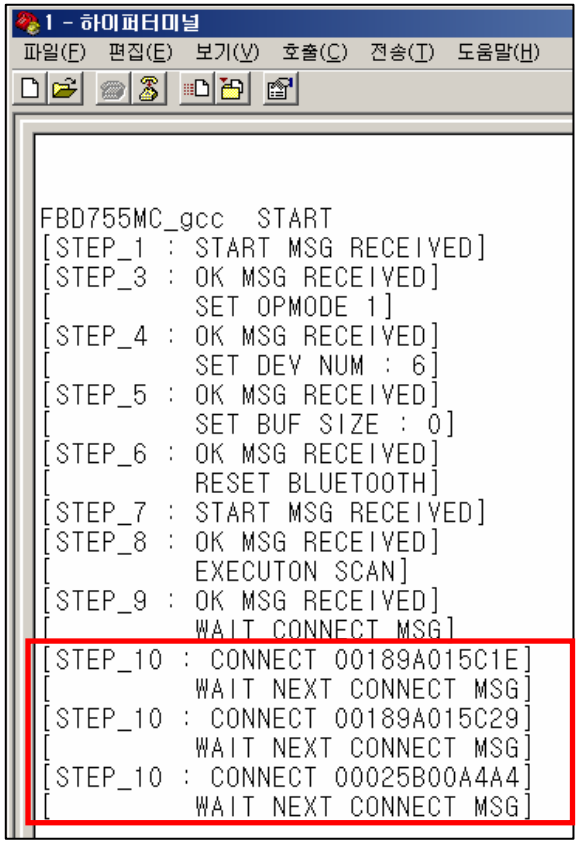
1. FBDx5xMC 보드를 Slave로 동작 시키기 위해 딥 스위치 1번 ON
2. FBDx5xMC 보드에 Bluetooth Device를 장착 후 전원 ON
3. Slave로 동작되는 경우, 1번 LED가 깜빡임
4. 하이퍼 터미널에 각 스텝 별 메시지 출력
   - STEP_1 : BTWIN Slave mode start 메시지 수신
   - STEP_3 : OK 메시지 수신, at+btopmode,1 명령 전달
   - STEP_4 : OK 메시지 수신, at+btdev=6 명령 전달
   - STEP_5 : OK 메시지 수신, at+btbuff=0 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)

49. FBDx5xMC 보드의 Slave 동작 화면



< FB755AS에 1개의 Master가 연결된 화면 >

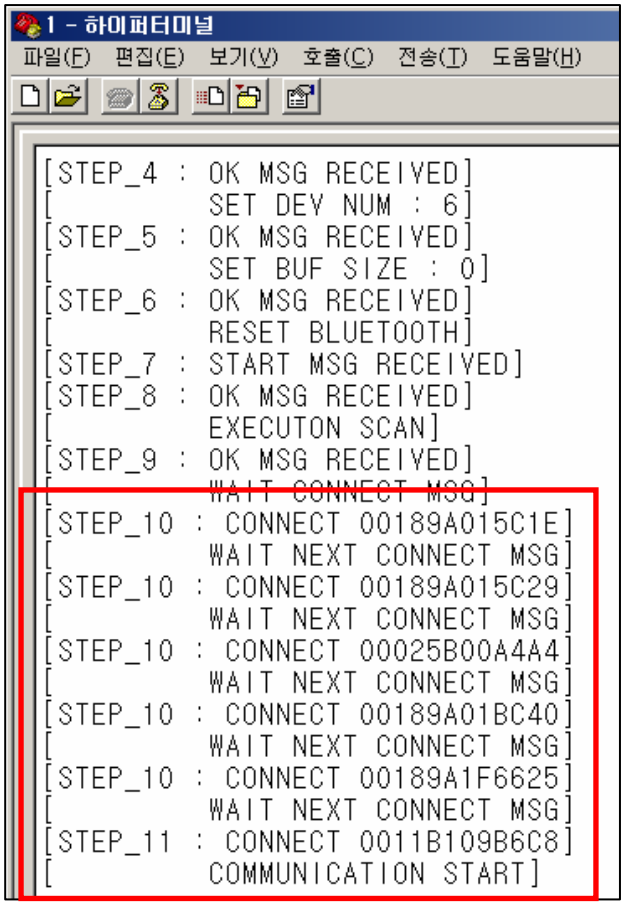
- FB755AS가 00189A015C1E 어드레스에 해당하는 Master 1개와 연결된 상태를 나타냄
- FB755AS가 2번째 마스터와 연결되기를 기다리는 상태를 나타냄



< FB755AS에 3개의 Master가 연결된 화면 >

- FB755AS가 00189A015C1E / 00189A015C29 / 00025B00A4A4 어드레스에 해당하는 Master 3개와 연결된 상태를 나타냄
- FB755AS가 4번째 마스터와 연결되기를 기다리는 상태를 나타냄

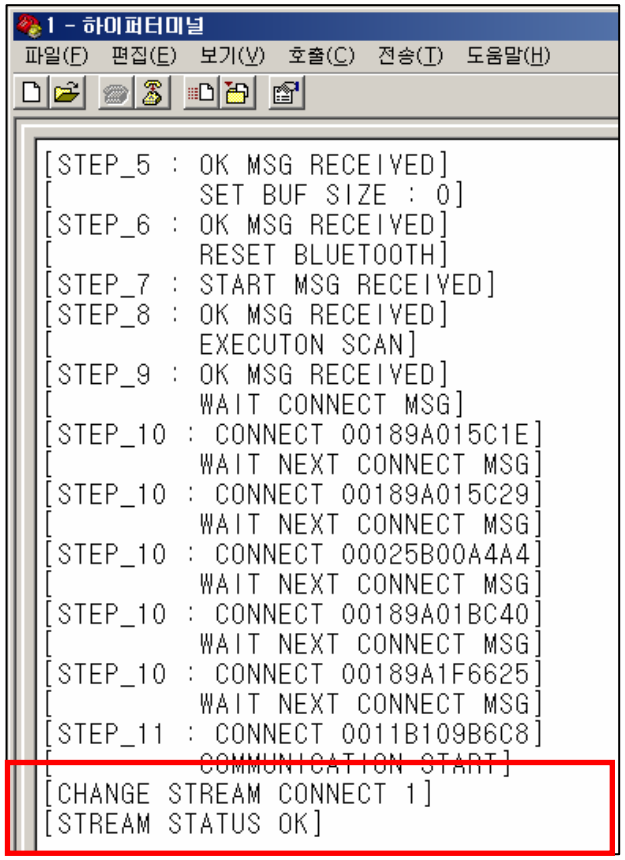
50. FB755AS에 설정한 “연결될 디바이스 수”만큼 연결된 화면



< FB755AS에 “연결될 디바이스 수”를 6개로 설정한 화면 >

- FB755AS가 00189A015C1E / 00189A015C29 / 00025B00A4A4 / 00189A01BC40 / 00189A1F6625 / 0011B109B6C8 어드레스에 해당하는 Master 6개와 연결된 상태를 나타냄
- 6개의 Master와 연결된 후, 데이터 송/수신 루틴 진행 가능한 상태를 나타냄
- 6개의 Master와 연결된 후, 각각의 Master에서 송신하는 데이터는 FB755AS를 통해서 출력됨
- 6개의 Master와 연결된 후, FB755AS에서 각각의 Master에 데이터를 송신하기 위해서는 Switch를 이용하여 STREAM 채널을 변경하여 송신 가능

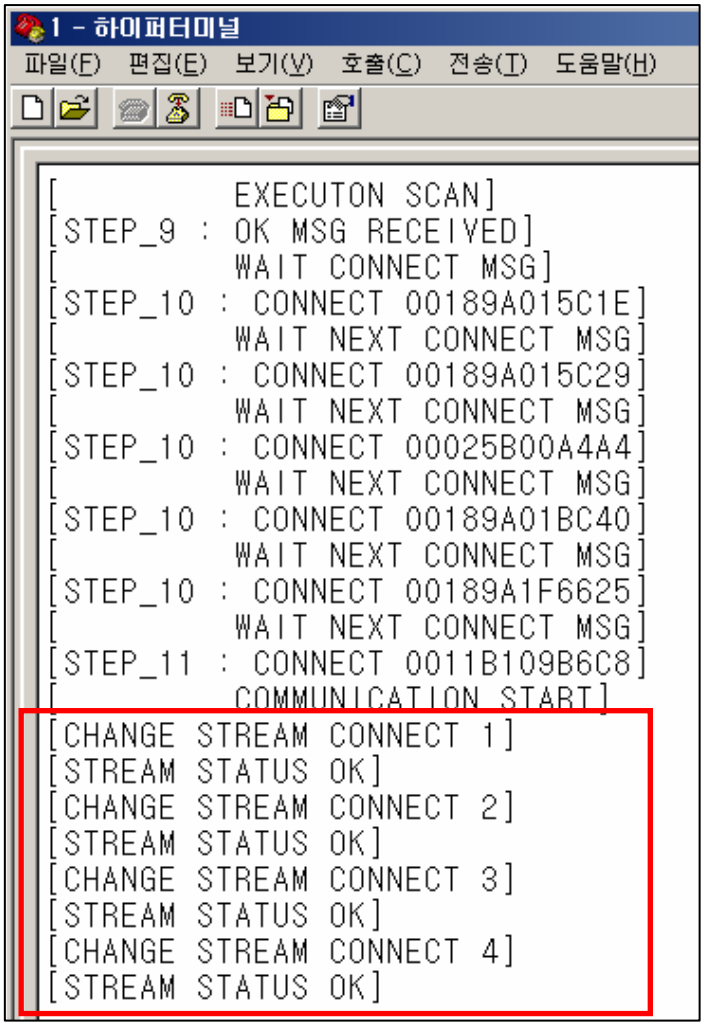
51. Switch 1을 이용한 FB755AS의 STREAM 채널 1 변경 화면



< 1번째 연결된 Master로 STREAM 채널을 변경한 경우 >

- FB755AS(1:N이 지원되는 Slave Bluetooth Module)를 이용하여 1번째 연결된 Master Bluetooth Device로 데이터를 전송하기 위한 절차
- FB755AS가 장착되어 Slave로 동작되는 FBDx5xMC보드의 1번 스위치 누름
- 하이퍼 터미널 창에 STREAM 채널이 1번으로 변경된다는 상태 메시지 출력
- 하이퍼 터미널 창에 STREAM 채널이 1번으로 변경 완료된 상태 메시지 출력
- FB755AS (Slave)에서 STREAM 채널 1을 통해 1번째 연결된 Master에 데이터 송신 가능
  - STREAM 채널 변경이 완료된 상태에서 시리얼 데이터를 입력하면 상대편 Bluetooth Device로 데이터가 전송됨

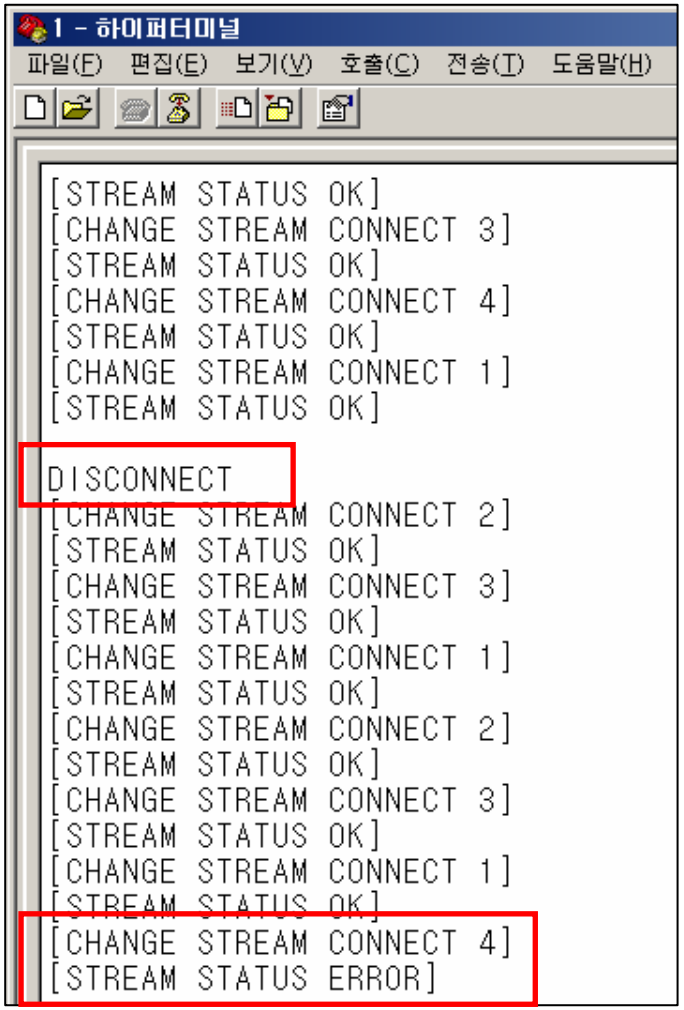
52. Switch 1/2/3/4를 이용한 FB755AS의 STREAM 채널 변경 화면



< 4개의 스위치를 이용하여 각각의 STREAM 채널을 변경하는 절차 >

- FB755AS(1:N이 지원되는 Slave Bluetooth Module)를 이용하여 설정된 디바이스 수 만큼 Master가 연결된 이후, STREAM 채널을 변경하여 보내고자 하는 Master를 선택하는 절차
- FBDx5xMC보드의 1번 스위치 누름
- STREAM 채널이 1번으로 변경 완료된 상태 메시지 출력
- Slave(FB755AS)에 시리얼 데이터를 입력하면 1번째 연결된 Master에 데이터가 수신됨
- 2번 스위치 누름
- STREAM 채널이 2번으로 변경 완료된 상태 메시지 출력
- Slave에 시리얼 데이터를 입력하면 2번째 연결된 Master에 데이터가 수신됨
- 3번 스위치 누름
- STREAM 채널이 3번으로 변경 완료된 상태 메시지 출력
- Slave에 시리얼 데이터를 입력하면 3번째 연결된 Master에 데이터가 수신됨
- 4번 스위치 누름
- STREAM 채널이 4번으로 변경 완료된 상태 메시지 출력
- Slave에 시리얼 데이터를 입력하면 4번째 연결된 Master에 데이터가 수신됨
- 나머지 5/6/7도 위와 같은 방식으로 STREAM 채널 변경 가능(FBDx5xMC보드에는 4개의 스위치가 존재 함으로 FBD755MC_gcc 프로그램에는 4개의 STREAM 채널 변경 방법만 구현되어 있음)

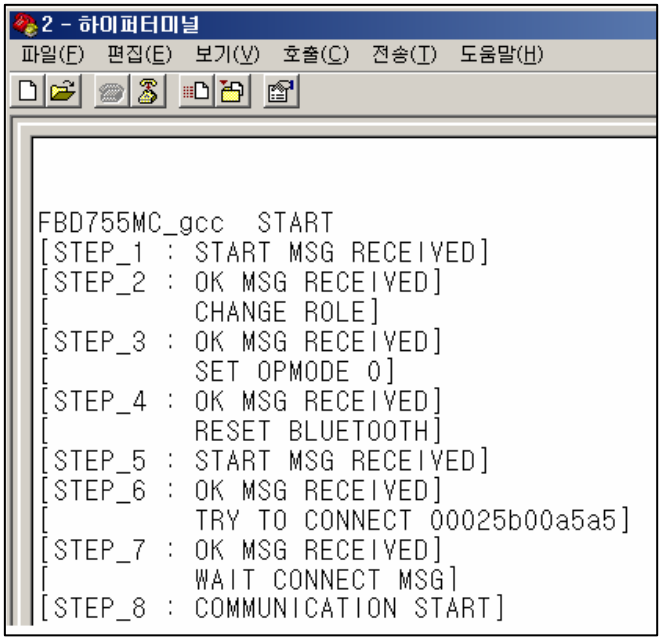
53. STREAM 채널 변경 실패 화면

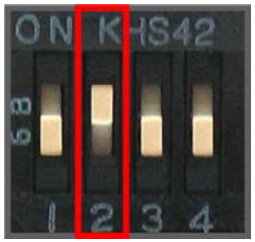


< STREAM 채널 변경에 대해 FB755AS에서 발생된 ERROR 출력 화면 >

- FB755AS(1:N이 지원되는 Slave Bluetooth Module)를 이용하여 설정된 디바이스 수 만큼 Master가 연결된 이후, STREAM 채널을 변경하여 데이터를 송신하는 과정 중에 STREAM 채널 변경 ERROR가 발생되는 경우
- N개의 Master중 1개의 Master와 연결이 끊어지는 경우, FB755AS는 DISCONNECT 메시지를 출력함
- DISCONNECT된 Master가 몇 번의 STREAM 채널에 연결되어 있었는지는 바로 알 수 없음
- FBD755MC_gcc 상에서는 Switch를 이용하여 STREAM 채널을 변경하는 루틴을 진행하면서 STREAM STATUS ERROR가 발생되는 채널을 확인하는 구조로 구성되어 있음
- FB755AS에서 어떤 STREAM 채널의 Master와 연결이 끊어 졌는지에 대한 조사는 AT Command를 이용하여 조사해야 함 (자세한 사항은 FB755AS의 AT-Command에 대한 매뉴얼 참고)
- 4번 스위치를 이용하여 STREAM 채널을 4로 변경하는 루틴 진행
- STREAM 채널이 4번으로 변경되지 못하고 ERROR상태 출력
- FB755AS는 4번째 연결된 Master에 데이터를 송신할 수 없는 상태(연결이 끊어진 상태)
- 끊어졌던 Master가 복구 되어 FB755AS와 다시 연결되면, FB755AS에서 CONNECT 메시지가 출력되고, STREAM 채널을 구성하여 다시 데이터를 송신할 수 있는 상태가 됨

54. FBDx5xMC 보드의 Master 설정 & 최초 동작 화면





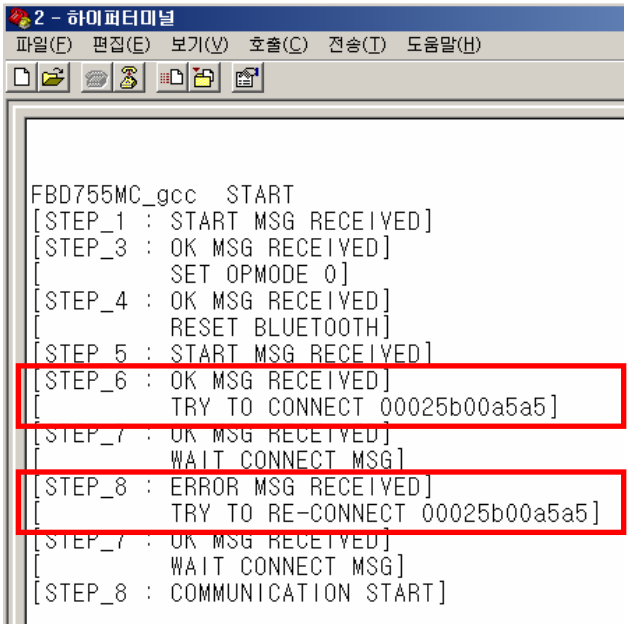
< 딥 스위치 2번 ON 상태 >

< Master 설정 & 동작 화면>

* Master가 실행되기 이전에 Slave가 SCAN 동작을 진행 하고 있어야 함

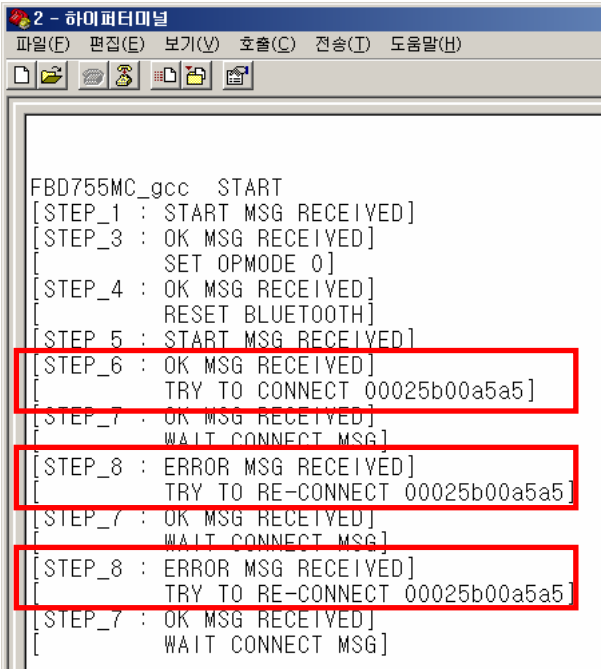
1. FBDx5xMC 보드를 Master로 동작 시키기 위해 딥 스위치 2번 ON
2. FBDx5xMC 보드에 Bluetooth Device를 장착 후 전원 ON
3. Master로 동작되는 경우, 2번 LED가 깜빡임
4. 하이퍼 터미널에 각 스텝 별 메시지 출력
   - STEP_1 : BTWIN Slave mode start 메시지 수신
   - STEP_2 : OK 메시지 수신, at+btrole=m 명령 전달
   - STEP_3 : OK 메시지 수신, at+btopmode,0 명령 전달
   - STEP_4 : OK 메시지 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

55. FBDx5xMC 보드의 Master 동작 화면



< Master에서 Slave로 연결하는 화면 : 2차 시도 후 연결>

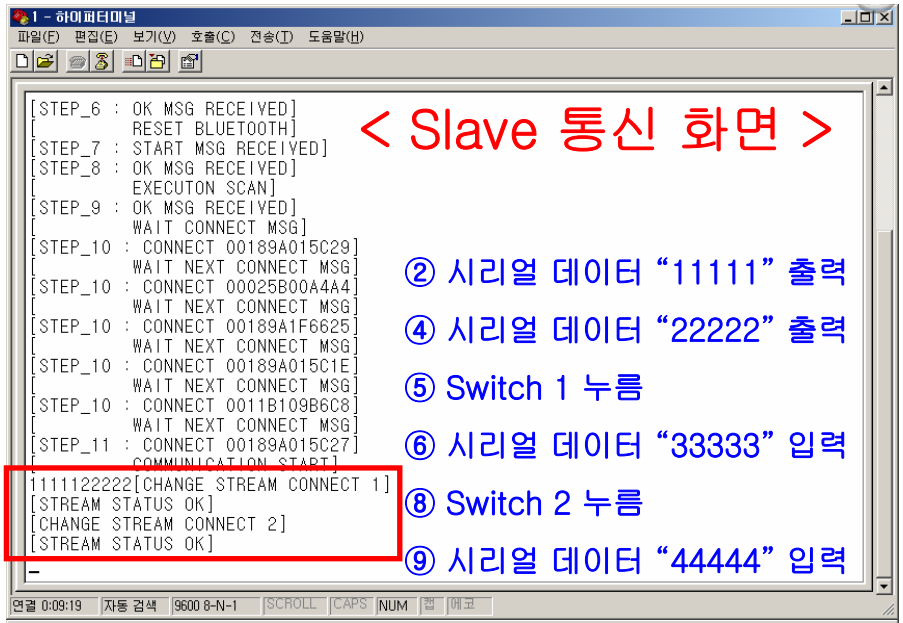
1. Master에서 어드레스가 00025b00a5a5인 Slave와 연결 시도
   - CONNECT 메시지 대신 ERROR 메시지 수신
2. Slave와 연결 재 시도
   - CONNECT 메시지 수신
3. 데이터 송/수신 루틴 진행
   - 시리얼 데이터 송/수신 가능



< Master에서 Slave로 연결하는 화면 : 3차 시도 후 연결>

1. Master에서 어드레스가 00025b00a5a5인 Slave와 연결 시도
   - CONNECT 메시지 대신 ERROR 메시지 수신
2. Slave와 연결 재 시도
   - CONNECT 메시지 대신 ERROR 메시지 수신
3. Slave와 연결 재 시도
   - CONNECT 메시지 수신
4. 데이터 송/수신 루틴 진행
   - 시리얼 데이터 송/수신 가능

56. Master와 Slave 통신 화면



< Slave 통신 화면 >

② 시리얼 데이터 “11111” 출력

④ 시리얼 데이터 “22222” 출력

⑤ Switch 1 누름

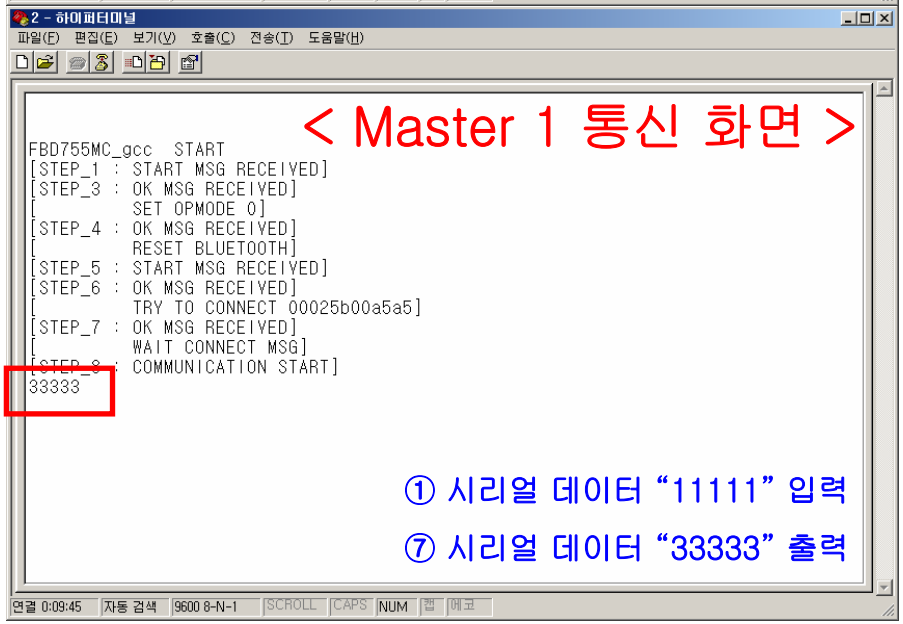
⑥ 시리얼 데이터 “33333” 입력

⑧ Switch 2 누름

⑨ 시리얼 데이터 “44444” 입력

< 데이터 송수신 테스트 절차 >

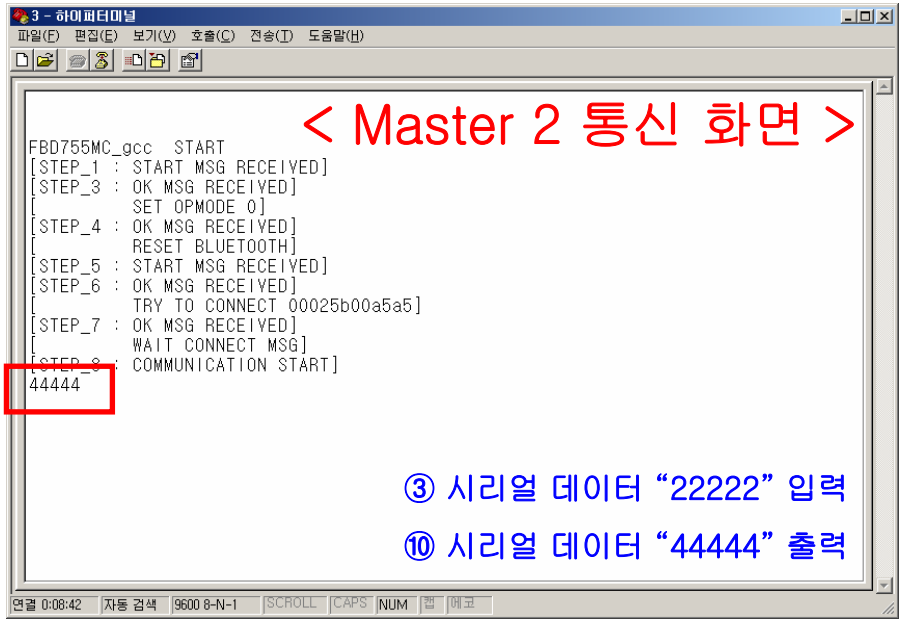
1. Master 1에서 데이터 송신 : “11111” 입력
2. Slave에 데이터 수신 : “11111” 출력
3. Master 2에서 데이터 송신 : “22222” 입력
4. Slave에 데이터 수신 : “22222” 출력
5. Slave에서 Switch 1 누름 : STREAM 채널 1번으로 변경
6. Slave에서 데이터 송신 : “33333” 입력
7. Master 1에 데이터 수신 : “33333” 출력
8. Slave에서 Switch 2 누름 : STREAM 채널 2번으로 변경
9. Slave에서 데이터 송신 : “44444” 입력
10. Master 2에 데이터 수신 : “44444” 출력



< Master 1 통신 화면 >

① 시리얼 데이터 “11111” 입력

⑦ 시리얼 데이터 “33333” 출력



< Master 2 통신 화면 >

③ 시리얼 데이터 “22222” 입력

⑩ 시리얼 데이터 “44444” 출력

57. 사용되는 상태 메시지 및 AT-Command 설명

1. “BTWIN Slave mode start” 또는 “BTWIN Master mode start”
   - Bluetooth Module 이 Slave 또는 Master로 설정되어 있는 경우, 시작되면서 출력되는 메시지
   - FBDx5xMC 보드에서 최초에 이 메시지를 검출
2. “OK”
   - 최초 메시지 출력이 후 AT-Command를 인식할 수 있는 상태를 나타내는 메시지
   - Bluetooth Module에 AT-Command를 입력한 경우, 입력된 AT-Command를 바르게 인식하거나 올바르게 동작한 경우 출력되는 메시지
   - 모든 AT-Command에 대해서 “OK” 메시지를 출력하지는 않음
3. “ERROR”
   - 입력된 AT-Command를 올바르게 인식하지 못하거나 수행하지 못한 경우 출력되는 메시지
4. “CONNECT”
   - Master와 Slave가 연결된 경우 출력되는 메시지
5. “at+btrole=s” 또는 “at+btrole=m”
   - Bluetooth Module의 Role을 Slave 또는 Master로 변경하기 위한 AT-Command
6. “at+btopmode,1” 또는 “at+btopmode,0” (1:N 지원 가능한 FB755AS 전용 AT-Command)
   - Bluetooth Module의 동작 모드(Operation Mode)를 1(1:N 모니터링 동작), 0(1:1 전용 동작)으로 설정하는 AT-Command
7. “at+btdev=n” (1:N 지원 가능한 FB755AS 전용 AT-Command)
   - Bluetooth Module에 연결 가능 디바이스 수 설정을 위한 AT-Command
8. “at+btbuff=nn” (1:N 지원 가능한 FB755AS 전용 AT-Command)
   - Bluetooth Module의 송/수신 버퍼 사이즈 설정을 위한 AT-Command
9. “ato1”, “ato2”, “ato3”, “ato4” (1:N 지원 가능한 FB755AS 전용 AT-Command)
   - STREAM 채널을 변경(1/2/3/4로 변경)하기 위한 AT-Command
10. “atz”
    - Bluetooth Module을 Soft Reset (전원을 껐다가 켜는 동작과 같음) 시키는 AT-Command
11. “at+btscan”
    - Slave로 설정된 Bluetooth Module 이 Master에서 검색/연결이 가능하도록 검색/연결 대기 상태로 진입하는 AT-Command
12. “atd”
    - 검색/연결 대기 중인 Slave Bluetooth Module과 연결을 하기 위한 AT-Command
    - “atd “ Command 이후에 연결하고자 하는 Slave Bluetooth Module의 Address 입력

58. 상태 메시지와 AT-Command Flow – Slave 동작

₩r₩nBTWIN Slave mode start₩r₩n

₩r₩nOK₩r₩n

at+btopmode,1₩n

₩r₩nOK₩r₩n

at+btdev=6₩n

₩r₩nOK₩r₩n

at+btbuff=0₩n

₩r₩nOK₩r₩n

atz₩n

₩r₩nBTWIN Slave mode start₩r₩n

₩r₩nOK₩r₩n

at+btscan₩n

₩r₩nOK₩r₩n

₩r₩nCONNECT001122334455₩r₩n

59. 상태 메시지와 AT-Command Flow – Master 동작

₩r₩nBTWIN Master mode start₩r₩n

₩r₩nOK₩r₩n

at+btopmode,0₩n

₩r₩nOK₩r₩n

atz₩n

₩r₩nBTWIN Master mode start₩r₩n

₩r₩nOK₩r₩n

atd저장된 어드레스₩n

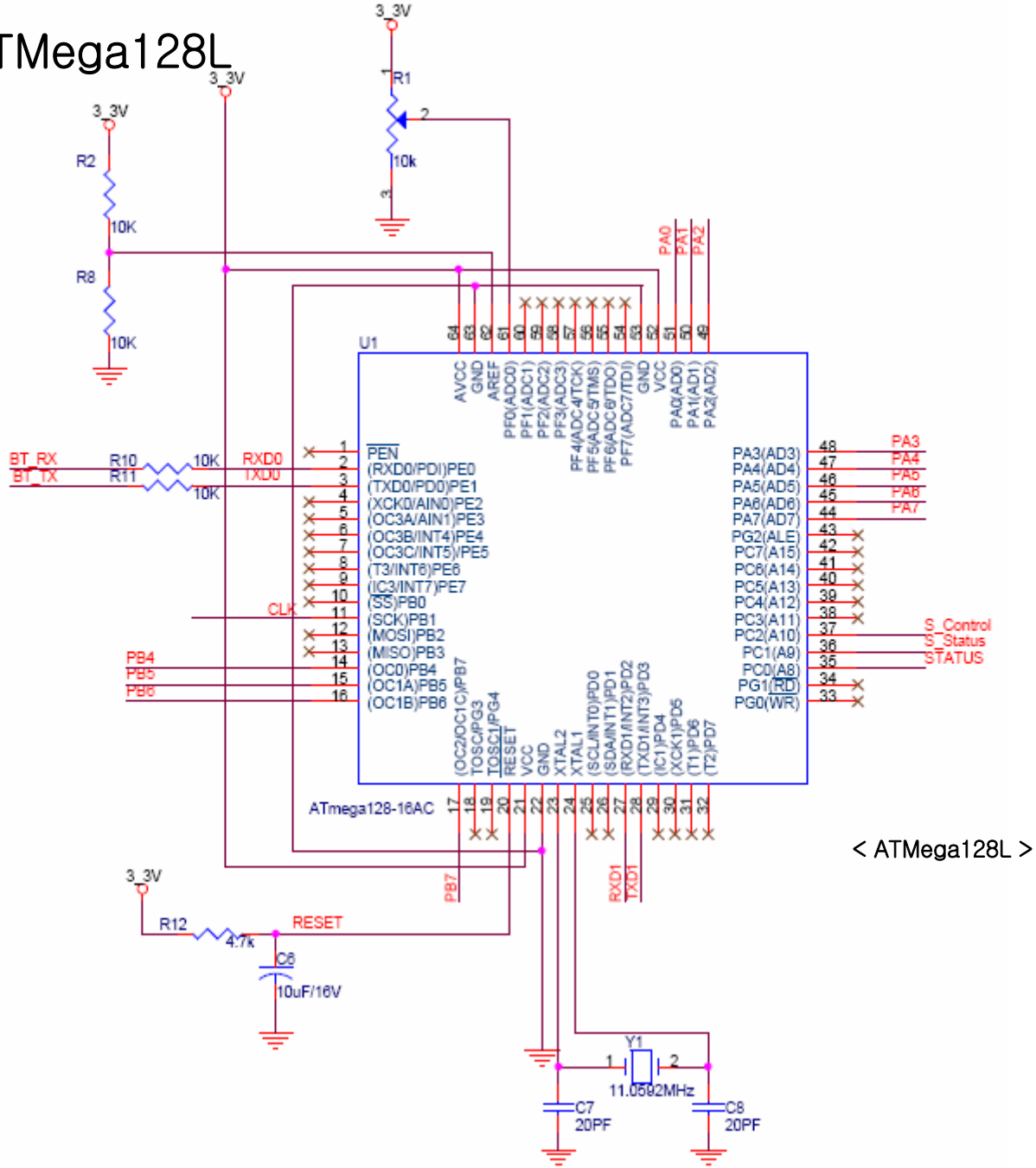
₩r₩nOK₩r₩n

₩r₩nCONNECT001122334455₩r₩n

※ FBDx5xMC(FBD755MC_gcc) 사용시 주의 사항

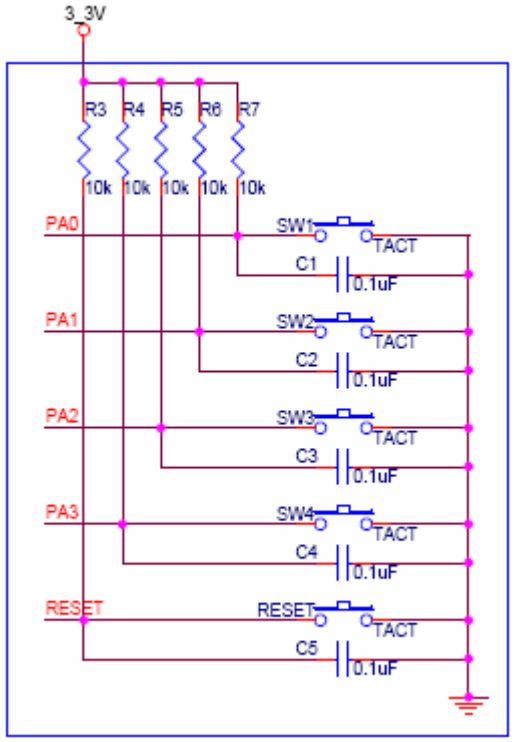
1. Slave Bluetooth Device Address의 정확한 입력
   - Master Bluetooth Device가 Slave Bluetooth Device와 연결하기 위해서 Slave Bluetooth Device의 Address 필요
   - Slave Bluetooth Device의 정확한 Address를 execution_master() Source에 정확히 입력 후 컴파일 진행
   - Slave Bluetooth의 Address가 정확하지 않으면 연결이 되지 않음
2. AVR Loader의 연결
   - 사용자의 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

회로도 – ATMega128L

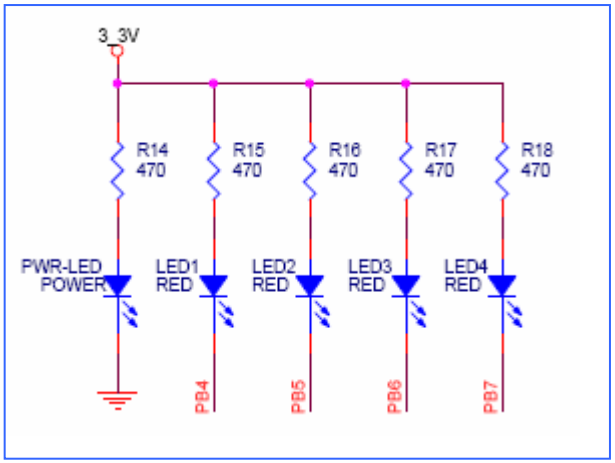


< ATMega128L >

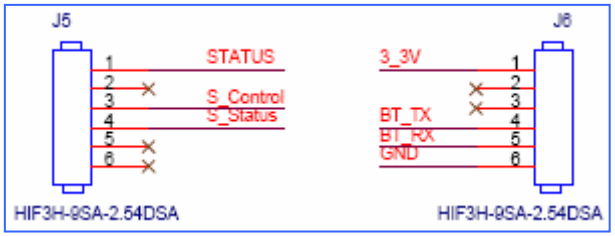
회로도 – 스위치 & LED & FB755BC 커넥터 & 딥 스위치



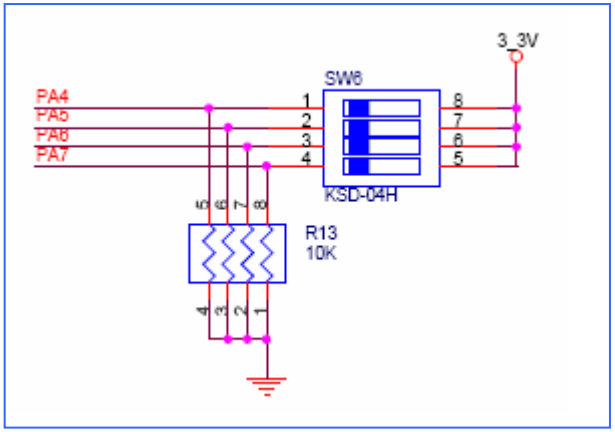
< Switch >



< LED >

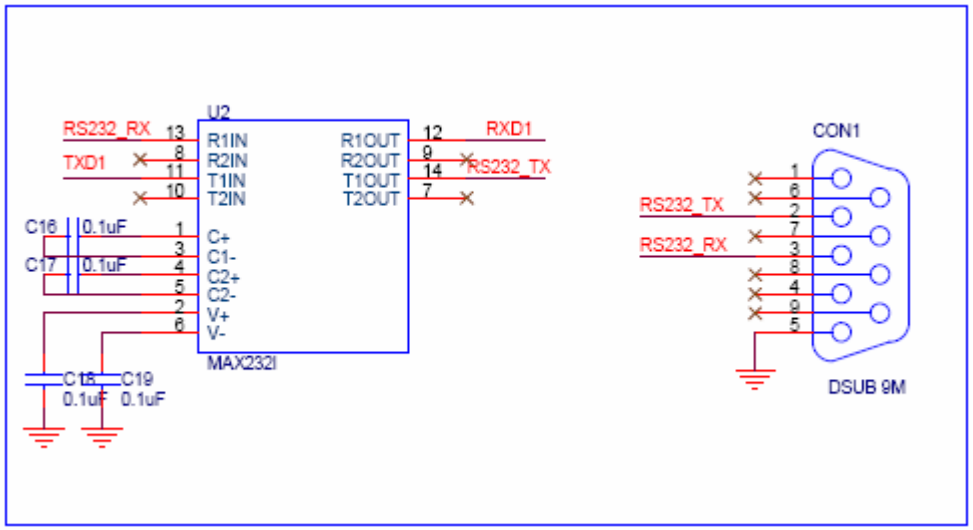


< FB755AS Connector >

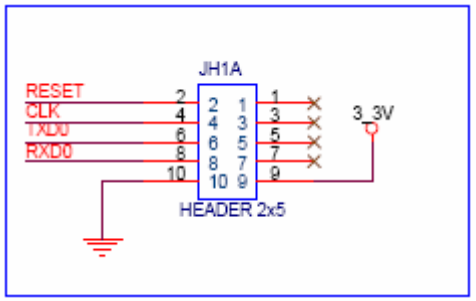


< Dip Switch >

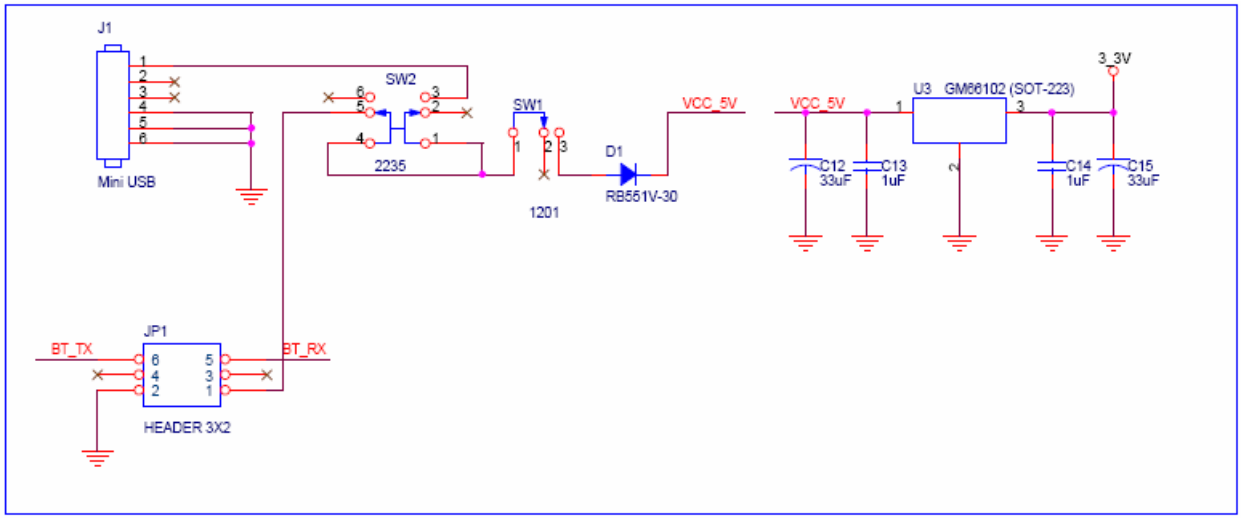
회로도 – RS-232 & AVR Loader & Power



< RS - 232 >



< AVR Loader >



< Power >