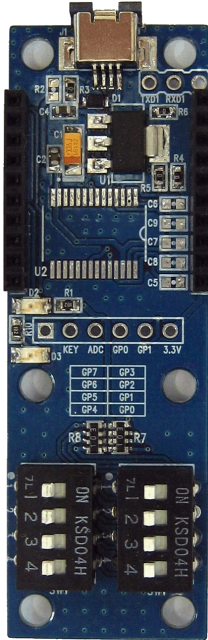


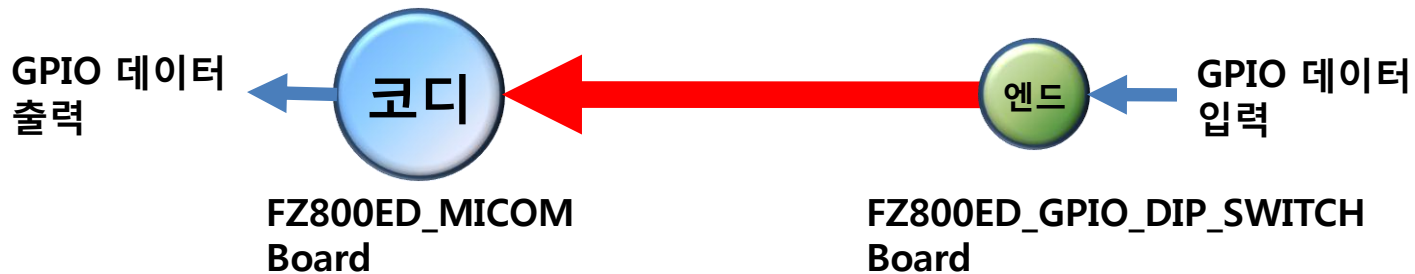
FZ800ED_GPIO_DIP_SWITCH

Application Guide

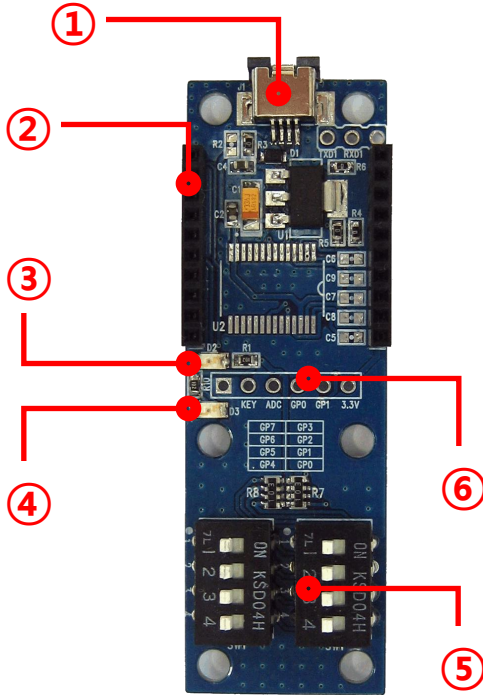
1. FZ800ED_GPIO_DIP_SWITCH Board



- FZ800ED_GPIO_DIP_SWITCH Board는 엔드디바이스로 설정된 FZ750BS 또는 FZ750BC를 장착하여 데이터를 송신하는 보드
- FZ800ED_GPIO_DIP_SWITCH Board는 엔드디바이스로 설정된 FZ750BS 또는 FZ750BC의 GPIO Port에 DIP Switch를 이용하여 GPIO 데이터 입력
- FZ800ED_GPIO_DIP_SWITCH Board는 FZ800ED_MICOM보드에 코디네이터로 설정하여 장착된 FZ750BS 또는 FZ750BC와 통신 진행

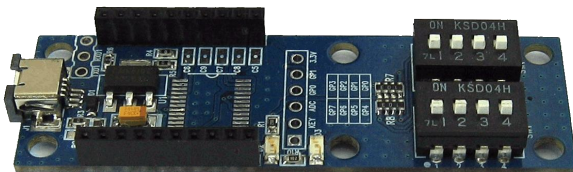


2. FZ800ED_GPIO_DIP_SWITCH 제품 외형

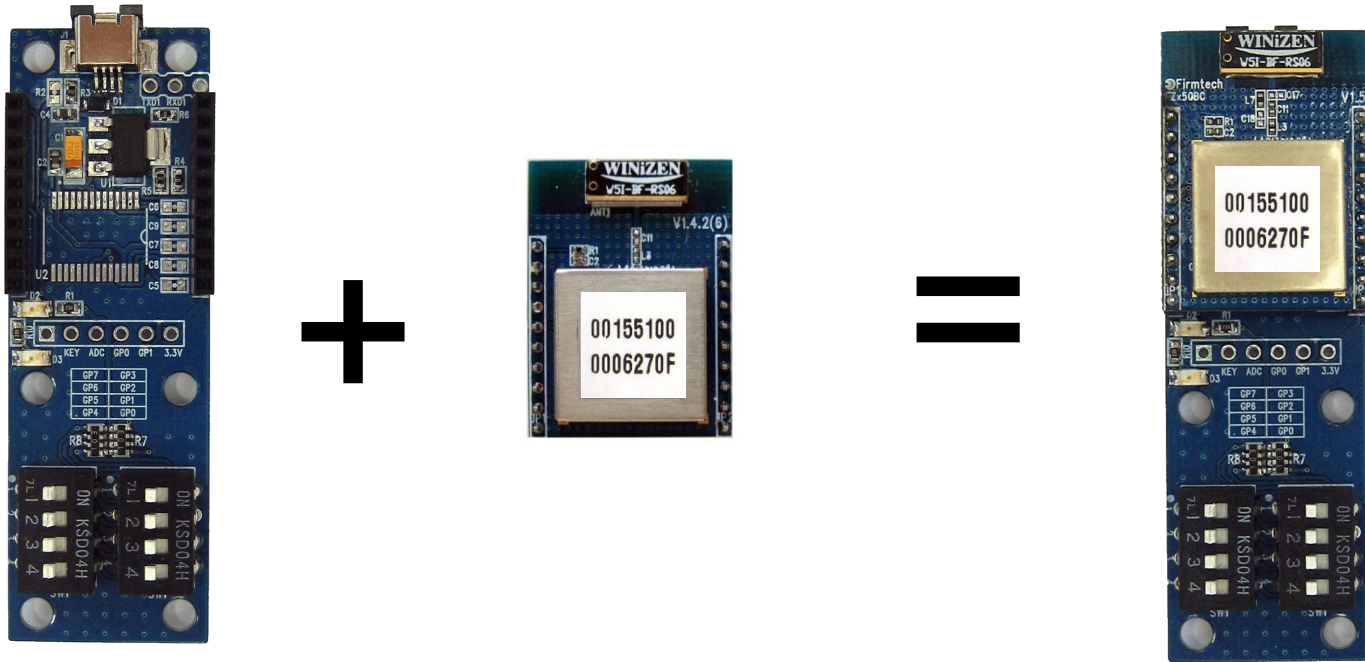


- 지그비 교육용 Sub 보드
- 입력 전원 : 5V
- FZ750BS 또는 FZ750BC 모듈의 GPIO 포트에 Dip Switch 데이터 입력
- 주로 엔드디바이스를 장착하여 사용

NO	Description
1	USB 전원 입력 단자
2	FZ750BS 또는 FZ750BC 연결 커넥터
3	STATUS LED(OK Port와 연결)
4	STATUS LED(GPIO_0와 연결)
5	GPIO 데이터 입력용 Dip Switch
6	확장 포트(GND/KEY/ADC/GPIO_0/GPIO_1/3.3V)



3. FZ750BX가 장착된 FZ800ED_GPIO_DIP_SWITCH

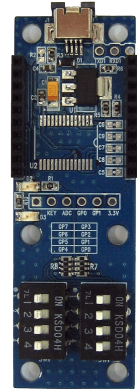


FZ750BS 또는 FZ750BC

설정 진행

4. FZ800ED_GPIO_DIP_SWITCH 를 사용하기 위한 사항(시간에 의한 송신)

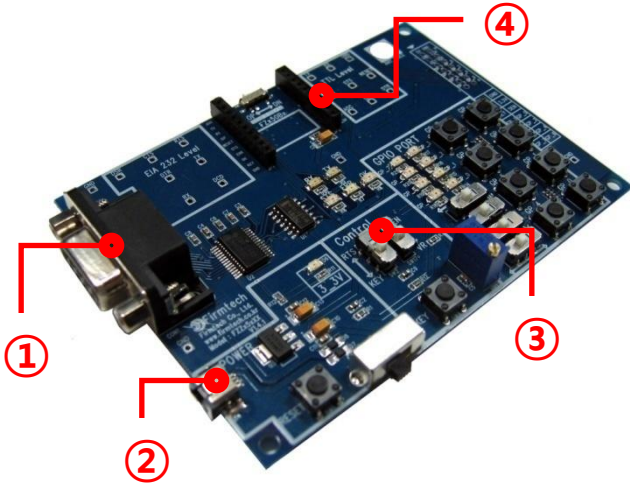
- FZ750BS 또는 FZ750BC 2개를 사용합니다.(코디네이터 1개, 엔드디바이스 1개)
- 디바이스 설정은 FZZx5xXX 인터페이스 보드를 사용합니다.
- 디바이스 설정은 하이퍼 터미널을 사용합니다.
- FZ800ED_GPIO_DIP_SWITCH 보드는 FZ800ED_MICOM 보드와 쌍으로 동작됩니다.
- FZ800ED_MICOM 보드는 FZ800ED_PARSING_GPIO 프로그램이 운영됩니다.
- FZ750BS 또는 FZ750BC는 공장 초기 값을 기준으로 설명합니다.



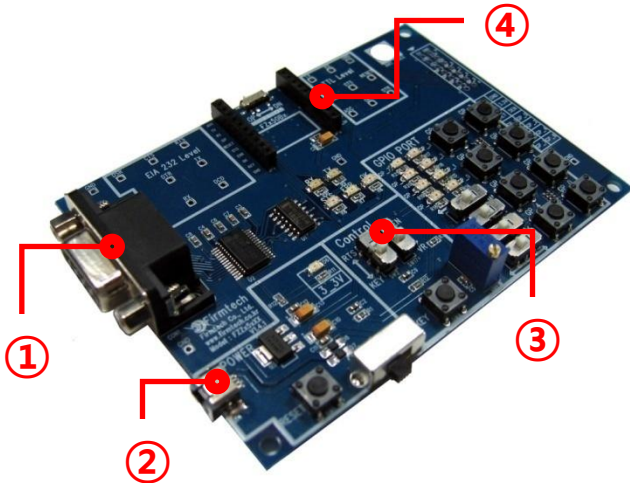
NO	Description
1	FZ750BS 또는 FZ750BC 코디네이터 설정(FZZx5xXX 인터페이스 보드 사용)
2	FZ750BS 또는 FZ750BC 엔드디바이스 설정(FZZx5xXX 인터페이스 보드 사용)
3	엔드디바이스의 타겟 디바이스를 코디네이터로 설정(FZZx5xXX 인터페이스 보드 사용)
4	엔드디바이스 ADC 데이터 사용 설정(FZZx5xXX 인터페이스 보드 사용)
5	엔드디바이스 데이터 송신 간격(시간) 설정(FZZx5xXX 인터페이스 보드 사용)
6	엔드디바이스 GPIO 데이터 사용 설정(FZZx5xXX 인터페이스 보드 사용)
7	FZ800ED_MICOM 보드에 FZ800ED_PARSING_GPIO 프로그램 다운로드
8	FZ800ED_MICOM 보드에 코디네이터로 설정한 FZ750BS 또는 FZ750BC 장착
9	FZ800ED_GPIO_DIP_SWITCH 보드에 엔드디바이스로 설정한 FZ750BS 또는 FZ750BC 장착
10	FZ800ED_GPIO_DIP_SWITCH 보드의 Dip Switch를 사용하여 Dip Switch(GPIO) 데이터 송신
11	FZ800ED_MICOM 보드에서 수신 받은 Dip Switch(GPIO) 데이터 시리얼 출력
12	FZ800ED_MICOM 보드에서 수신 받은 데이터가 분석하고자 하는 Dip Switch(GPIO) 데이터인 경우 USERS_OPERATION 동작

5. FZ750BS 또는 FZ750BC 설정

(1) FZZx5xXX 인터페이스 보드 연결 및 체크 : 설정은 FZZx5xXX 인터페이스 보드 사용



NO	연결 및 체크 사항(코디네이터 설정 용)
1	RS-232 Cable을 연결하여 PC와 연결
2	USB Power Cable을 연결하여 PC와 연결
3	"RTS/KEY"선택 스위치 KEY 선택
4	코디네이터로 설정할 FZ750BS 또는 FZ750BC 장착



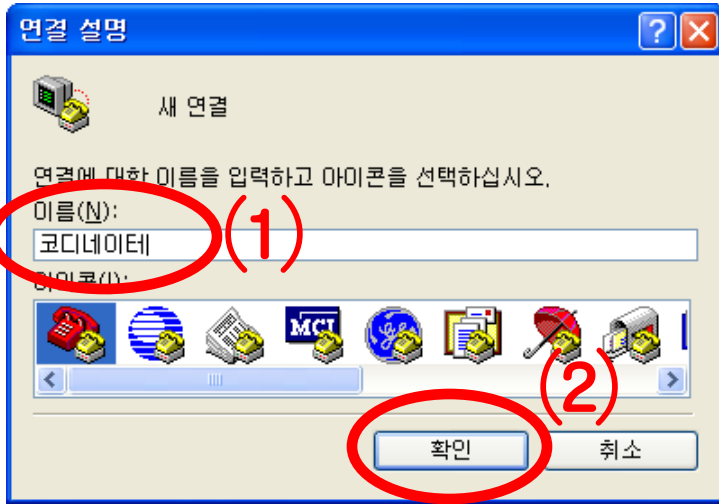
NO	연결 및 체크 사항(엔드디바이스 설정 용)
1	RS-232 Cable을 연결하여 PC와 연결
2	USB Power Cable을 연결하여 PC와 연결
3	"RTS/KEY"선택 스위치 KEY 선택
4	엔드디바이스로 설정할 FZ750BS 또는 FZ750BC 장착

(2) 하이퍼 터미널 실행하기



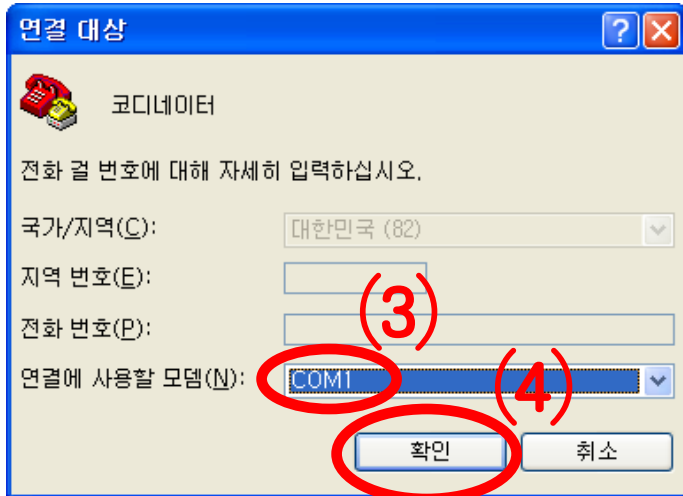
NO	실행 사항
1	Windows의 "시작" 선택
2	"프로그램" 선택
3	"보조프로그램" 선택
4	"통신" 선택
5	"하이퍼터미널" 선택

(3) 하이퍼 터미널 설정 - 이름 입력



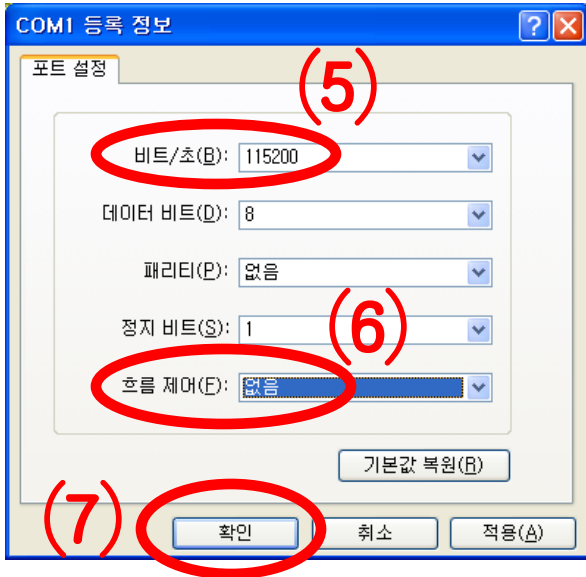
- 코디네이터로 설정할 FZ750BX와 연결된 하이퍼 터미널을 설정
- “이름”에 “코디네이터” 입력
- “확인”을 선택하여 다음 진행

(4) 하이퍼 터미널 설정 - 사용 포트 입력



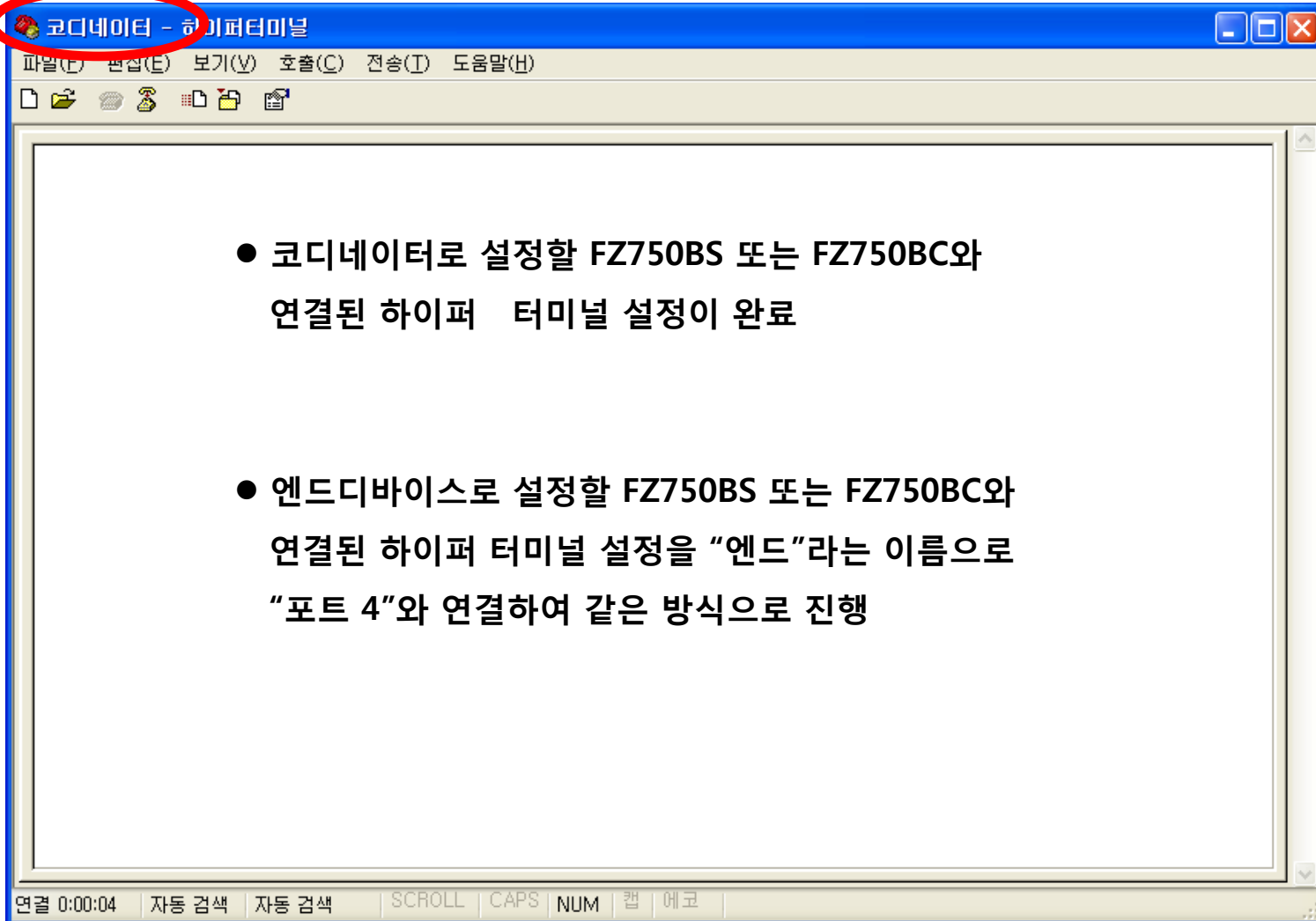
- 코디네이터로 설정할 FZ750BX와 연결된 “포트”를 선택
- “확인”을 선택하여 다음 진행

(5) 하이퍼 터미널 설정 - 통신 속도 외 설정

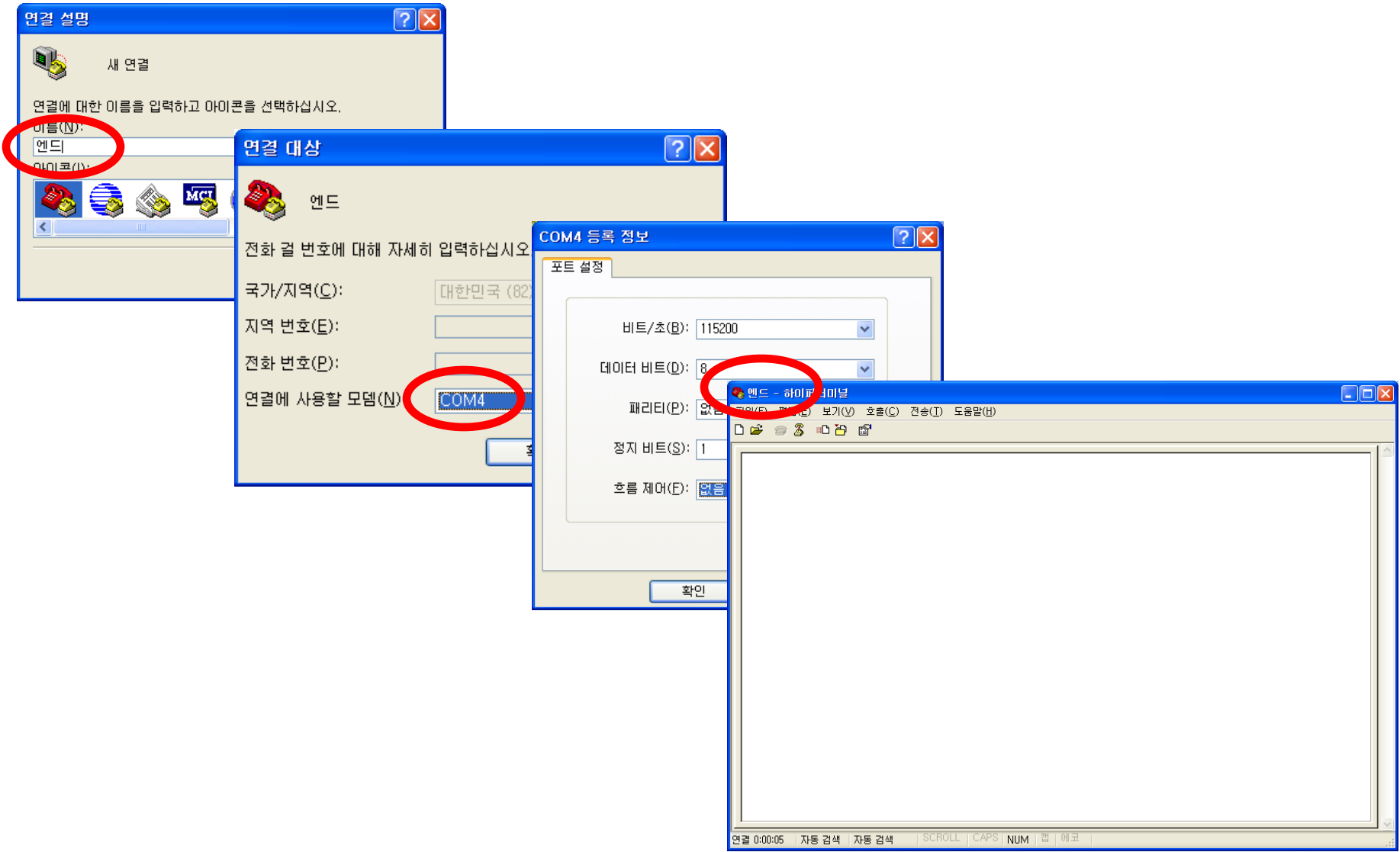


- "비트/초(B)"를 "115200"을 설정
- "흐름제어(F)"를 "없음"으로 설정
- 다른 사항은 변경하지 않음
- "확인"을 선택

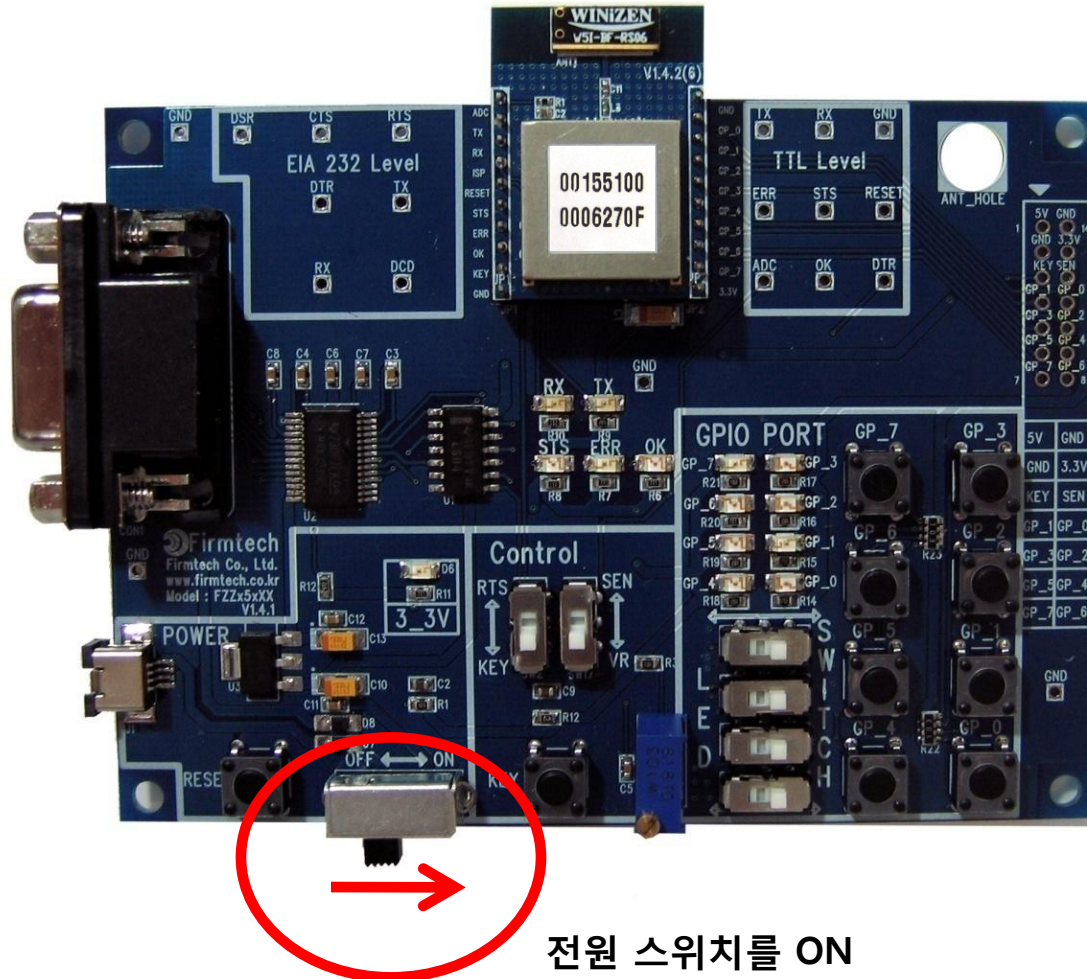
(6) 하이퍼 터미널 설정 - 완료



(7) 엔드디바이스로 설정할 FZ750BS/FZ750BC와 연결된 하이퍼 터미널 설정 완료된 화면 - 포트 4 사용

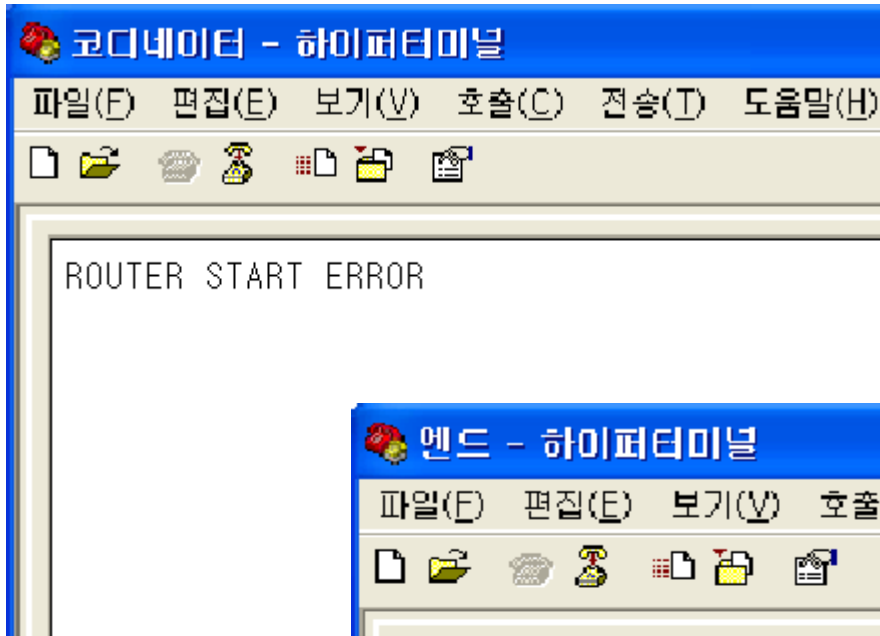


(8) FZ750BS 또는 FZ750BC 전원 ON

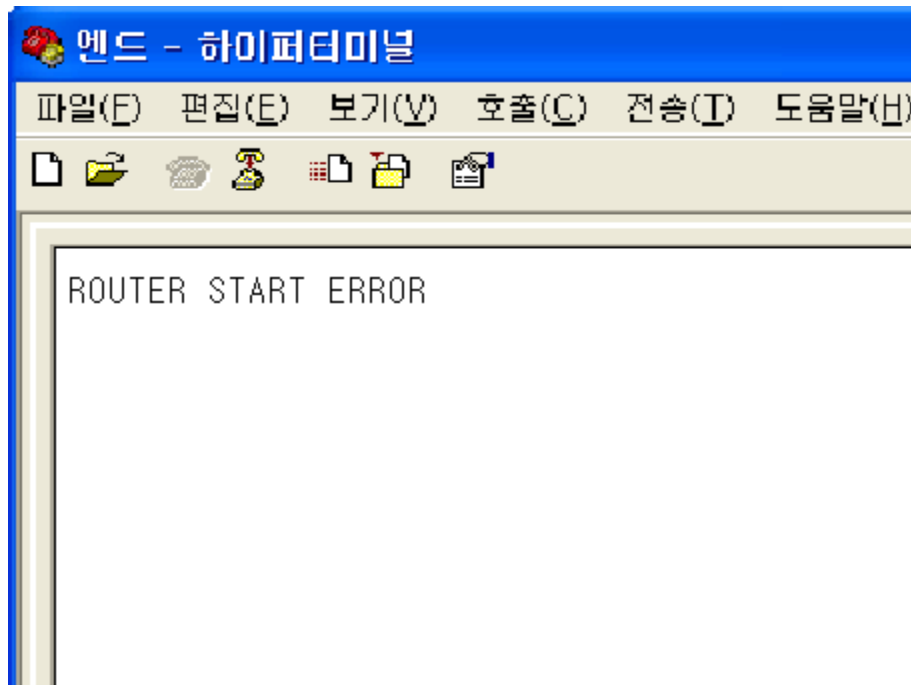


전원 스위치를 ON

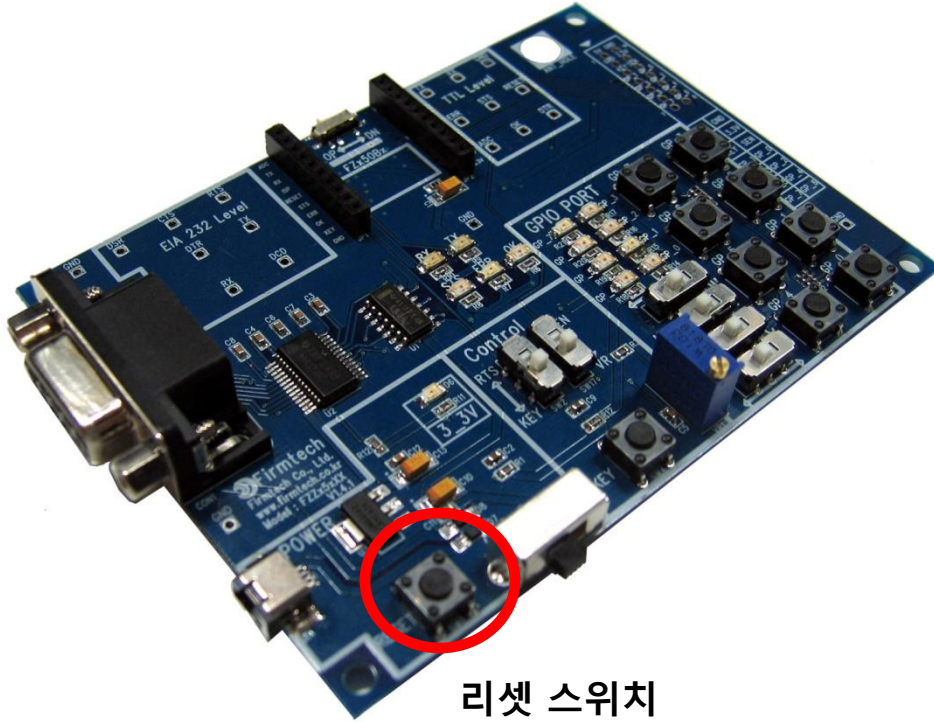
(9) 하이퍼 터미널 출력 화면



- FZ750BS 또는 FZ750BC 공장 초기 값인 경우,
"ROUTER START" 메시지 출력
- 주위에 지그비 네트워크가 생성되지 않은 경우
"ERROR" 메시지 출력



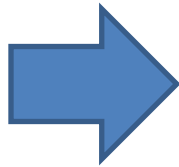
< 디바이스 리셋 >



리셋 스위치

- 정상 동작 되지 않거나 하이퍼 터미널에 아무런 문자가 출력되지 않으면 FZ750BS 또는 FZ750BC 재 시작 진행
- FZx5xXX Board의 Reset Switch를 눌러 재 시작 진행
- 통신 속도와 기타 연결 사항도 체크
- FZ750BS또는 FZ750BC 설정을 진행함에 있어 주위에 지그비 네트워크가 없는 상태에서 설정 진행
- 만약, 지그비 네트워크(코디네이터/라우터)가 존재하는 경우, 해당 디바이스의 전원을 모두 OFF한 상태에서 설정 진행
- 설정을 진행하는 FZ750BS 또는 FZ750BC는 공장 초기 값을 기준으로 설정 진행

< STS LED 상태 확인 >



0.1초 간격
STS LED ON

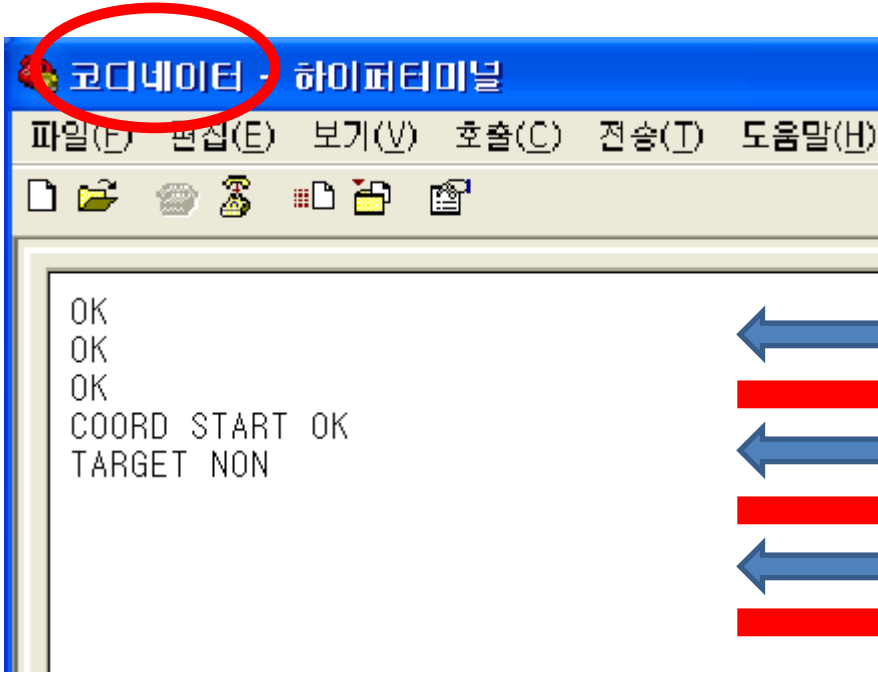


0.1초 간격
STS LED OFF



- FZZx5xXX 보드의 전원이 ON 되면 STS LED의 상태 체크
- 네트워크 구축/참여가 안된 상태 임으로, STS LED는 0.1초 간격으로 빠르게 깜빡임

(10) FZ750BS 또는 FZ750BC 코디네이터 설정



코디네이터로 설정할 디바이스와 연결된 하이퍼 터미널에 다음과 같이 입력

- ← 하이퍼 터미널에 “+++”입력
- FZ750BX에서 “OK”출력
- ← 하이퍼 터미널에 “AT+SETCOORD”입력 후 엔터키 입력
- FZ750BX에서 “OK”출력
- ← 하이퍼 터미널에 “ATZ”입력 후 엔터키 입력
- FZ750BX에서 “OK”출력
- FZ750BX 디바이스 재 시작
- “COORD START OK” 출력
- “TARGET NON”출력

FZ750BS 또는 FZ750BC 코디네이터 설정 완료

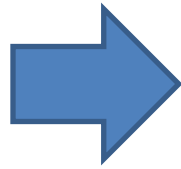
< AT Command Mode 상태의 STS/OK/ERR LED 상태 >



- Operation Mode인 상태에서 하이퍼 터미널에 “+++”을 입력하면 AT Command Mode로 변경
- AT Command Mode인 경우, STS LED는 OFF된 상태 유지
- AT Command Mode인 경우, ERR/OK LED는 ON된 상태 유지

- AT Command Mode에서 하이퍼 터미널에 “ATO”를 입력하고 엔터키를 입력하면 Operation Mode로 변경
- AT Command Mode에서 하이퍼 터미널에 “ATZ”를 입력하고 엔터키를 입력하면 디바이스가 리셋 되면서 Operation Mode로 변경

< 네트워크 구축/참여가 완료된 Operation Mode 의 STS LED 상태 >



0.1초 간격
STS LED ON

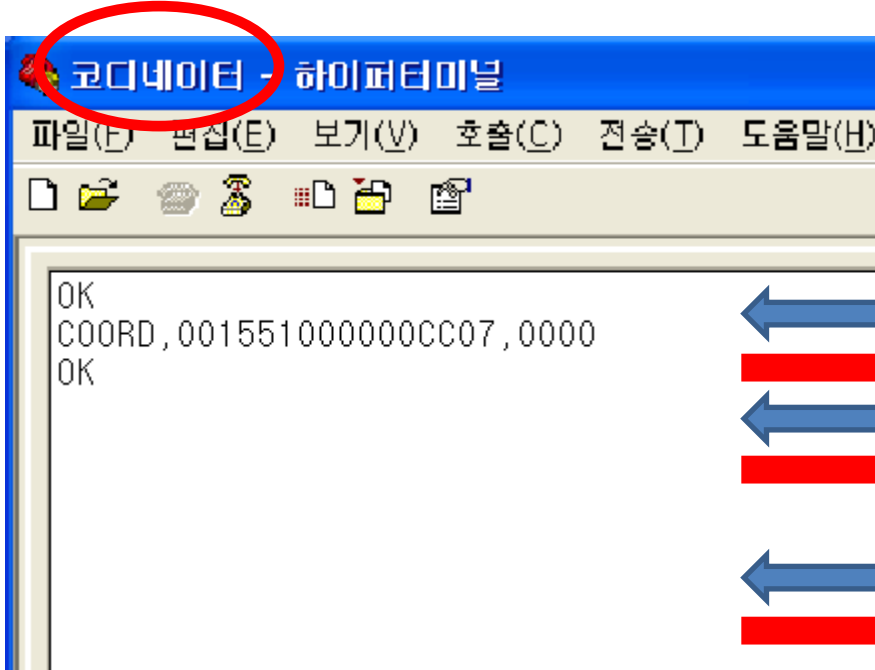


1초 간격
STS LED OFF



- 네트워크 구축/참여가 완료된 경우, STS LED는 1초 간격으로 천천히 깜빡임
- ERR/OK LED는 OFF된 상태 유지
- 네트워크 구축/참여가 1회 완료되면, 디바이스가 리셋 되어도 네트워크 구축/참여가 자동으로 진행됨

(11) 코디네이터 IEEE ADDRESS 조사하기



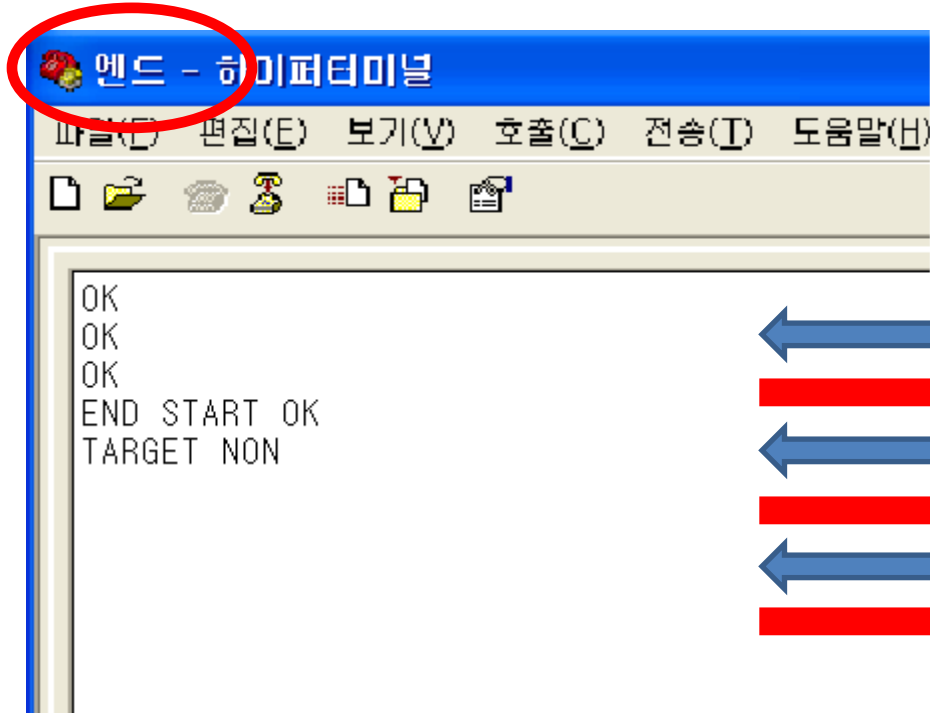
코디네이터로 설정한 디바이스 연결된 하이퍼 터미널에 다음과 같이 입력

- 하이퍼 터미널에 “+++”입력
- FZ750BX에서 “OK”출력
- 하이퍼 터미널에 “AT+GETLOCAL”입력 후 엔터키 입력
- FZ750BX에서 “COORD, 001551000000CC07, 0000”출력
- 코디네이터의 IEEE ADDRESS는 “001551000000CC07”
- 하이퍼 터미널에 “ATO”입력 후 엔터키 입력
- FZ750BX에서 “OK”출력

사용자가 사용하는 디바이스마다 IEEE ADDRESS는 다름

타겟디바이스를 설정하기 위해서는 각 디바이스의 IEEE ADDRESS를 알고 있어야 함

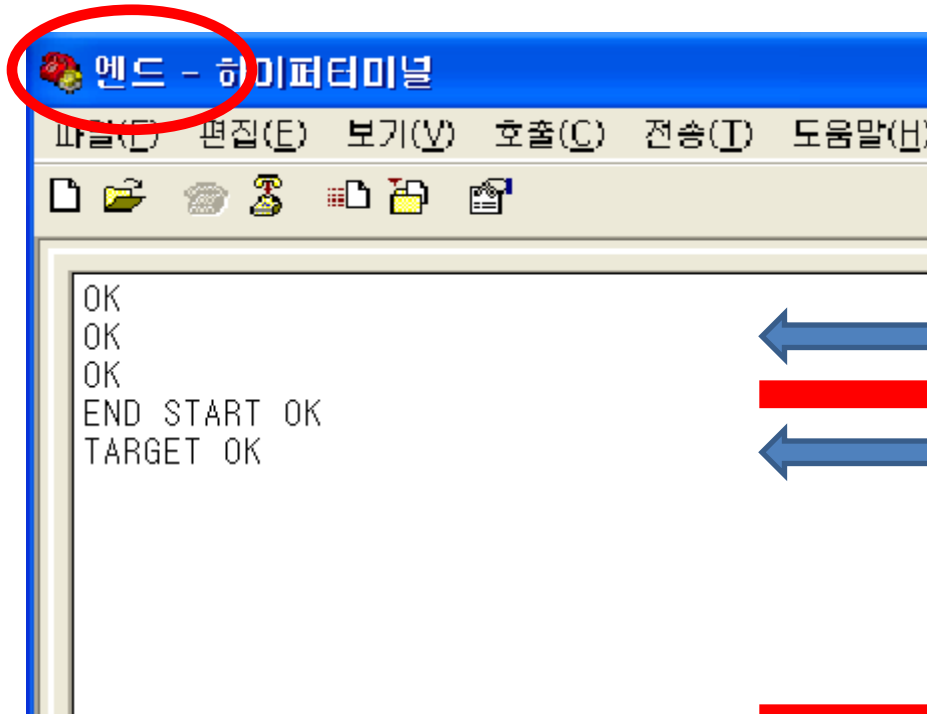
(12) FZ750BS 또는 FZ750BC 엔드디바이스 설정



엔드디바이스로 설정할 디바이스와 연결된 하이퍼 터미널에 다음과 같이 입력

- ← 하이퍼 터미널에 “+++”입력
- FZ750BX에서 “OK”출력
- ← 하이퍼 터미널에 “AT+SETEND”입력 후 엔터키 입력
- FZ750BX에서 “OK”출력
- ← 하이퍼 터미널에 “ATZ”입력 후 엔터키 입력
- FZ750BX에서 “OK”출력
- FZ750BX 디바이스 재 시작
- “END START OK” 출력
- “TARGET NON”출력

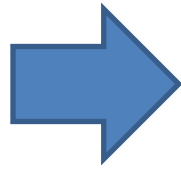
(13) 엔드디바이스의 타겟디바이스를 코디네이터로 설정하기



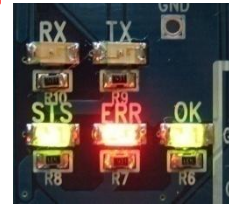
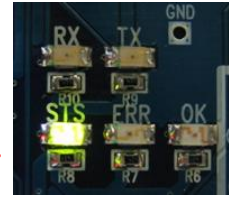
엔드디바이스로 설정한 디바이스와 연결된 하이퍼 터미널에 다음과 같이 입력

- ← 하이퍼 터미널에 “+++”입력
- FZ750BX에서 “OK”출력
- ← 하이퍼 터미널에 “AT+SETTARGET001551000000CC07” 입력 후 엔터키 입력
- 001551000000CC07은 앞에서 조사한 어드레스
- 디바이스가 다른 경우 다른 어드레스가 출력됨
- 조사된 어드레스를 입력해야 함
- FZ750BX에서 “OK”출력
- ← 하이퍼 터미널에 “ATZ”입력 후 엔터키 입력
- FZ750BX에서 “OK”출력
- FZ750BX 디바이스 재 시작
- “END START OK” 출력
- “TARGET OK”출력

< 타겟디바이스가 설정된 Operation Mode 의 STS LED 상태 (엔드디바이스) >



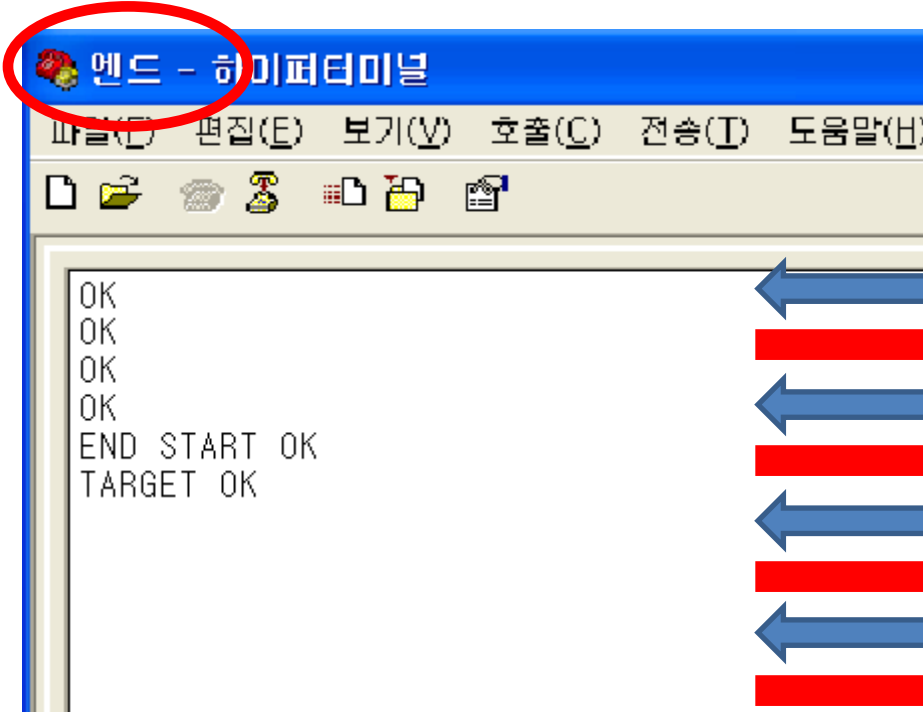
KEY 데이터
입력 시
또는 약 10
초 후에
Wake Up



1초 후
저전력 모드로 변경

- 타겟디바이스가 설정된 경우, STS LED는 ON된 상태 유지
- Operation Mode인 경우, ERR/OK LED는 OFF된 상태 유지
- **엔드디바이스의 경우**, 타겟디바이스가 설정되면 자동으로 저전력 모드로 진입하고, 정해진 동작(KEY 데이터 입력 또는 약 10초 후)에 한번씩 Wake Up
- 타겟디바이스 설정이 1회 완료되면, 디바이스가 리셋 되어도 타겟디바이스 설정은 자동 진행

(14) 엔드디바이스의 ADC 데이터 송신 사용 설정 & 송신 시간 10초 설정



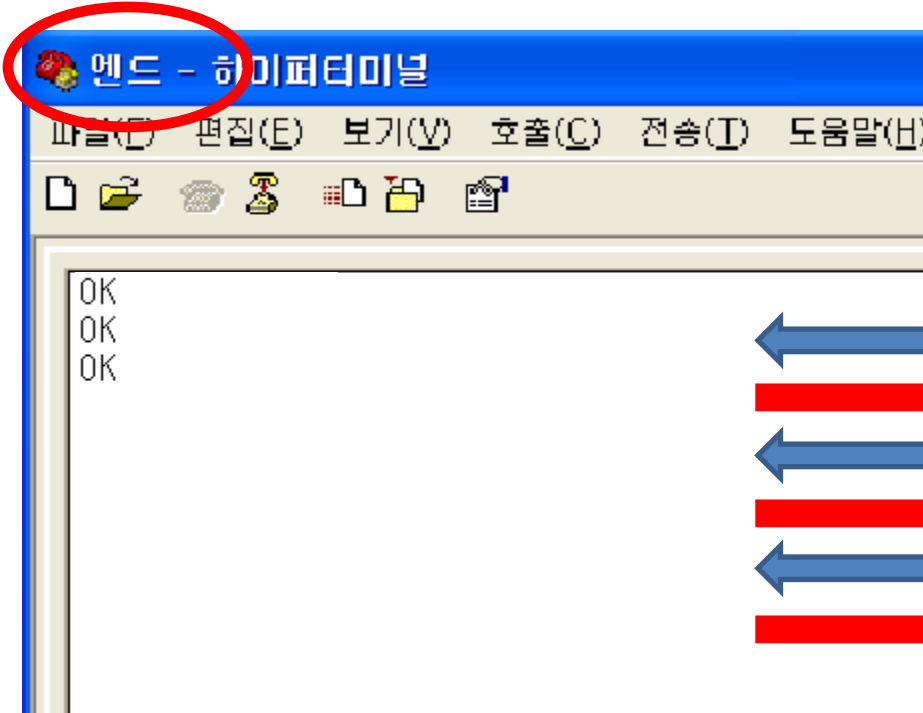
엔드디바이스를 **Wake Up** 시킨 이후, **엔드디바이스로 설정한** 디바이스와 연결된 하이퍼 터미널에 다음과 같이 입력

- 엔드디바이스를 Wake Up 시키는 방법은, KEY 스위치를 이용하여 KEY 데이터 입력
- 아무런 설정을 하지 않으면, 약 10초 후에 자동으로 Wake Up
- KEY 데이터 입력 이후, 다시 저전력 모드로 진입하기 이전에 +++ 입력하여 AT Command Mode 변경
- 시간 관련한 사항(AT+SETMR10)은 다비야스가 리셋 된 이후 적용됨

- 하이퍼 터미널에 "+++"입력
- FZ750BX에서 "OK"출력
- 하이퍼 터미널에 "AT+SETADC1"입력 후 엔터키 입력
- FZ750BX에서 "OK"출력
- 하이퍼 터미널에 "AT+SETTMR10"입력 후 엔터키 입력
- FZ750BX에서 "OK"출력
- 하이퍼 터미널에 "ATZ"입력 후 엔터키 입력
- FZ750BX에서 "OK"출력
- FZ750BX 디바이스 재 시작
- "END START OK" 출력
- "TARGET OK"출력

위와 같이 설정되면, 정해진 시간에 1회씩 ADC 데이터 송신하기 시작함

(15) 엔드디바이스의 GPIO 입력 설정



엔드디바이스를 **Wake Up** 시킨 이후, **엔드디바이스로 설정한** 디바이스와 연결된 하이퍼 터미널에 다음과 같이 입력

- 하이퍼 터미널에 “+++”입력
- FZ750BX에서 “OK”출력
- 하이퍼 터미널에 “AT+SETGPIO1”입력 후 엔터키 입력
- FZ750BX에서 “OK”출력
- 하이퍼 터미널에 “ATO”입력 후 엔터키 입력
- FZ750BX에서 “OK”출력

- 엔드디바이스를 Wake Up 시키는 방법은, KEY 스위치를 이용하여 KEY 데이터 입력
- 아무런 설정을 하지 않으면, 약 10초 후에 자동으로 Wake Up
- KEY 데이터 입력 이후, 다시 저전력 모드로 진입하기 이전에 +++ 입력하여 AT Command Mode 변경

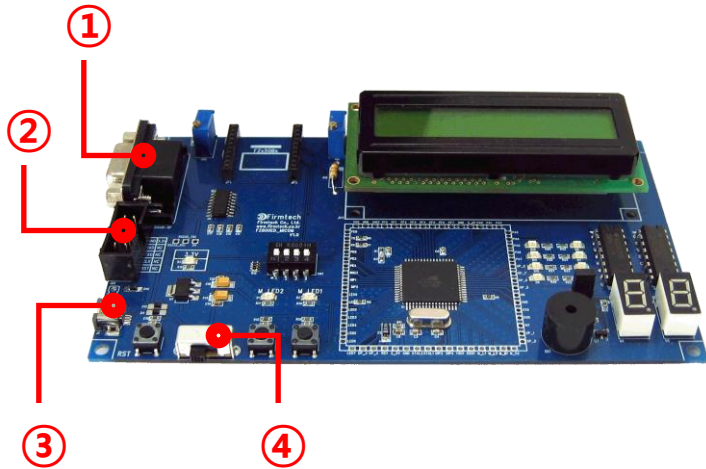
위와 같이 설정되면, 정해진 시간에 1회씩 ADC 데이터를 송신하는 대신 GPIO포트를 읽은 값을 데이터로 생성하여 송신함

FZZx5xXX 보드의 전원을 OFF하고, 다음 사항 진행

FZ800ED_PARSING_GPIO

HEX File Download

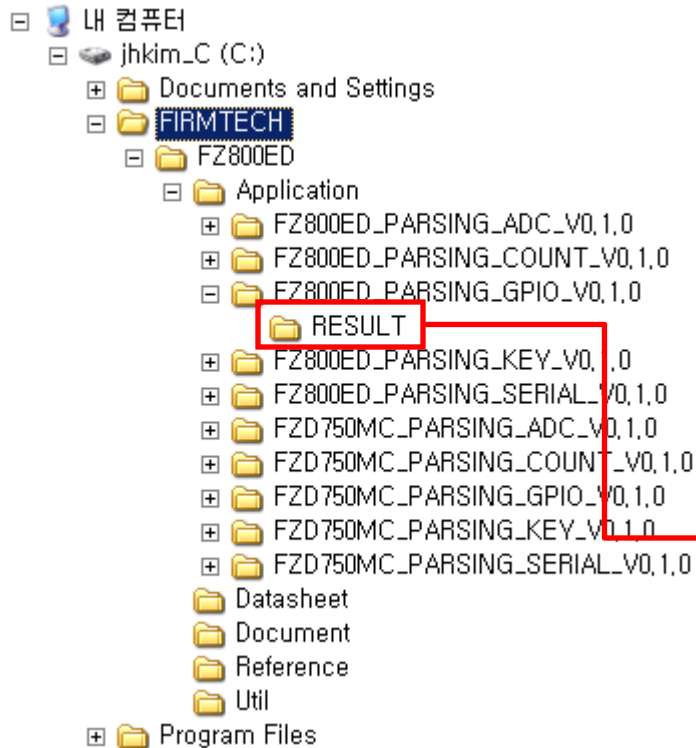
6. FZ800ED_MICOM Board 연결



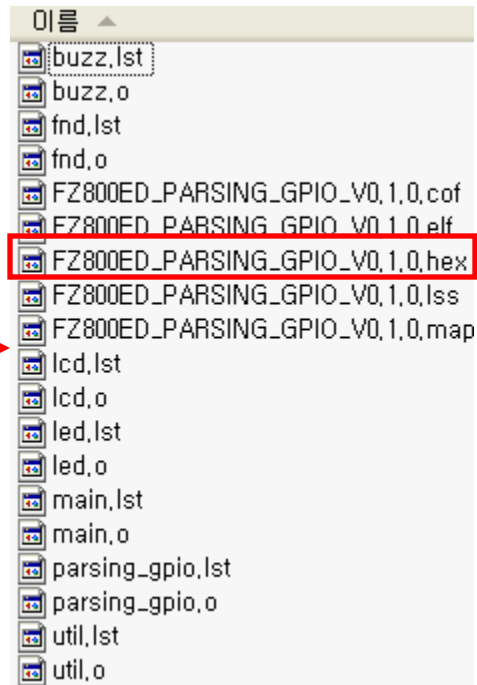
NO	연결 및 체크사항
1	FZ800ED_MICOM보드와 RS-232 Cable을 연결하여 PC와 연결
2	FZ800ED_MICOM보드와 AVR Loader(다운로드 케이블)연결하여 PC와 연결
3	FZ800ED_MICOM보드와 USB Power Cable을 연결하여 PC와 연결
4	FZ800ED_MICOM보드 전원 ON

기존에 설정했던 하이퍼 터미널(코디네이터 설정용)을 그대로 사용

7. FZ800ED_MICOM Board 다운로드 Hex File 확인

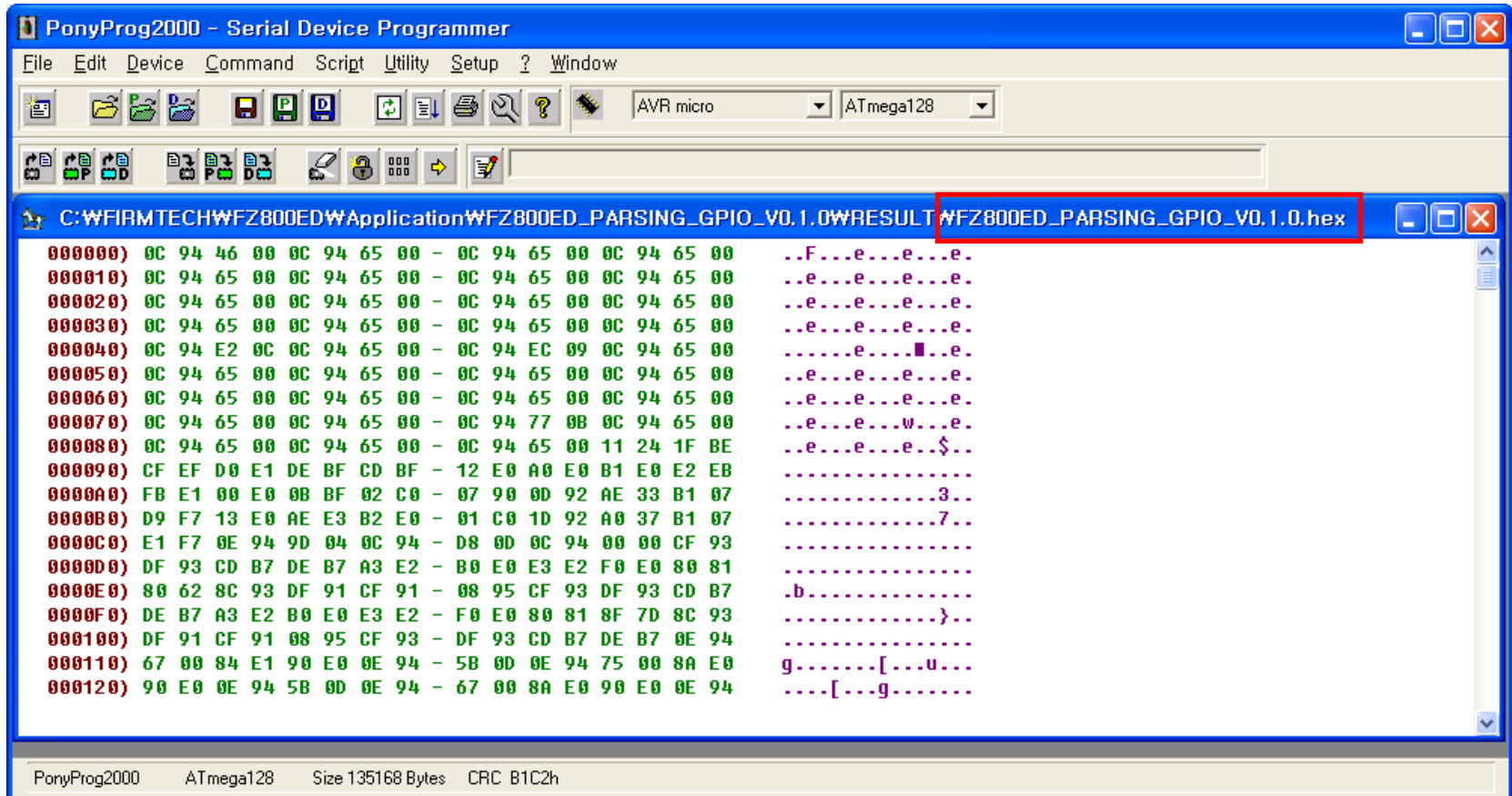


FZ800ED_GPIO_DIP_SWITCH 보드에서 송신한 데이터를
분석하기 위해 FZ800ED_MICOM 보드에
FZ800ED_PARSING_GPIO_V0.1.0.hex 파일 다운로드



8. FZ800ED_MICOM Board 프로그램 다운로드

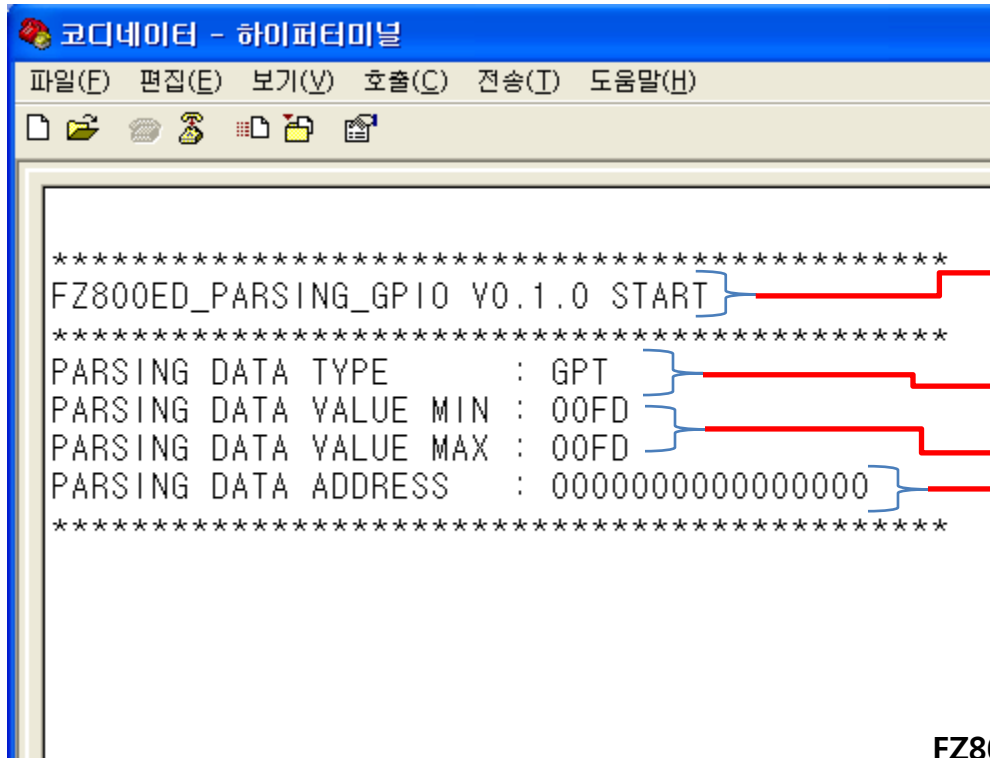
1. FZ800ED_PARSING_GPIO.hex 파일을 FZ800ED_MICOM 보드에 다운로드
 - 다운로드 프로그램은 PonyProg 사용
 - <http://www.lancos.com/ppwin95.html> 에서 다운로드 하여 설치
 - AVR Loader로 PC와 FZ800ED_MICOM 보드 연결
 - FZ800ED_MICOM 보드에 생성된 HEX 파일을 다운로드



< HEX 파일 다운로드 프로그램 PonyProg >

9. FZ800ED_MICOM Board 동작상태 체크

FZ800ED_PARSING_GPIO.hex파일이 정상적으로 다운로드 된 경우, FZ800ED_MICOM보드를 리셋 시키면 하이퍼 터미널 상에서 운영되는 프로그램 관련 데이터 확인 가능



The screenshot shows a HyperTerminal window titled '코디네이터 - 하이퍼터미널'. The menu bar includes '파일(F)', '편집(E)', '보기(V)', '호출(C)', '전송(T)', and '도움말(H)'. The main text area displays the following output:

```
*****  
FZ800ED_PARSING_GPIO V0.1.0 START  
*****  
PARSING DATA TYPE      : GPT  
PARSING DATA VALUE MIN : 00FD  
PARSING DATA VALUE MAX : 00FD  
PARSING DATA ADDRESS  : 0000000000000000  
*****
```

현재 운영되는 프로그램 정보

수신 데이터의 타입을 구분하기 위한 정보

수신 데이터의 최소/최대 값을 구분하기 위한 정보

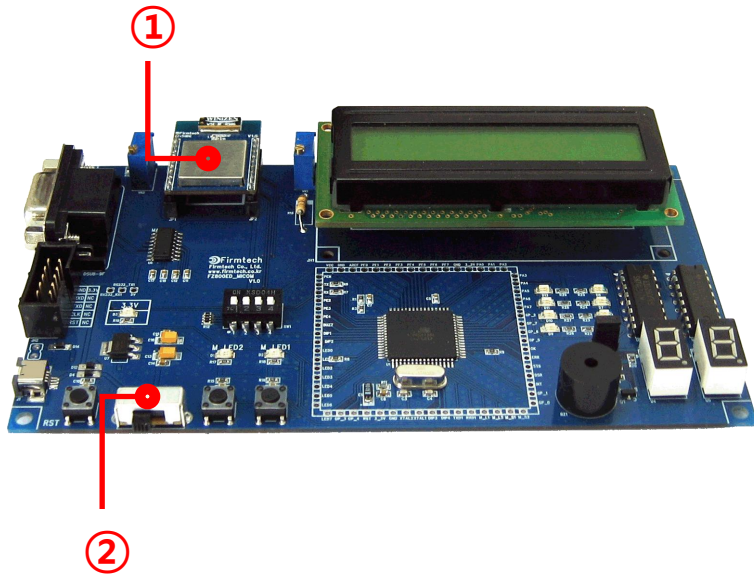
수신 데이터의 어드레스를 구분하기 위한 정보

FZ800ED_MICOM 보드의 전원을 OFF하고, 다음 사항 진행

FZ800ED_GPIO_DIP_SWITCH

동작 시키기

10. FZ800ED_MICOM Board에 코디네이터 장착



NO	동작
1	코디네이터로 설정된 FZ750BS 또는 FZ750BC를 FZ800ED_MICOM 보드에 장착
2	FZ800ED_MICOM Board 전원 ON

11. FZ800ED_MICOM Board 전원 ON

< FZ800_MICOM보드의 전원을 ON하면 >

1. BUZZ 울림
2. LCD 기본 데이터 출력
3. FND 0부터 9까지 카운트
4. LED 순차적 켜졌다 꺼짐
5. BUZZ 울림
6. 시리얼 포트에 시리얼 데이터 출력

```
코디네이터 - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)
*****
FZ800ED_PARSING_GPIO V0.1.0 START
*****
PARSING DATA TYPE      : GPT
PARSING DATA VALUE MIN : 00FD
PARSING DATA VALUE MAX : 00FD
PARSING DATA ADDRESS  : 0000000000000000
*****
COORD START OK
TARGET NON
```

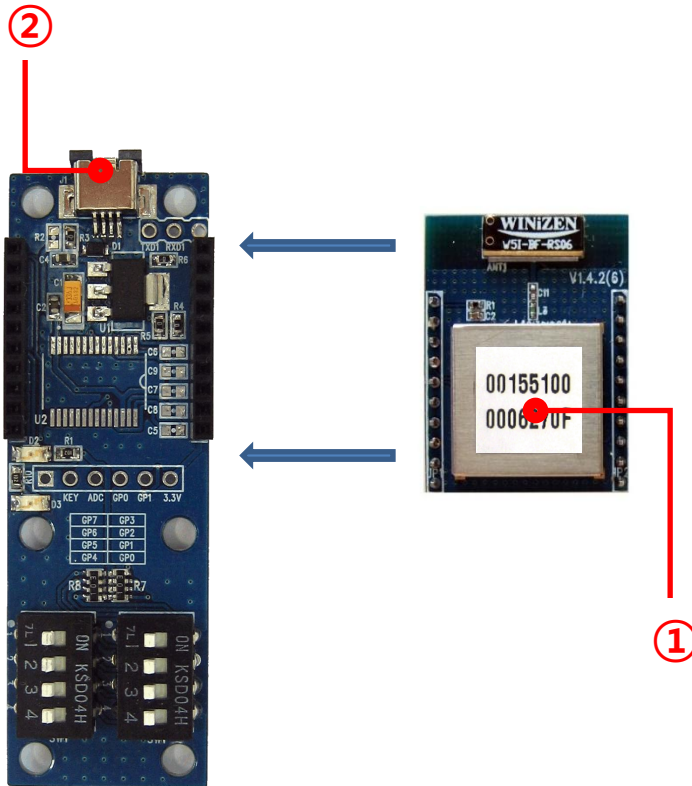
- 현재 운영되는 프로그램 정보
- 수신 데이터의 타입을 구분하기 위한 정보
- 수신 데이터의 최소/최대 값을 구분하기 위한 정보
- 수신 데이터의 어드레스를 구분하기 위한 정보
- 장착된 코디네이터가 정상적으로 동작

✚ 체크 사항

FZ800ED_MICOM보드의 전원을 ON 했을 때, 하이퍼 터미널에 "COORD START OK"메시지와 "TARGET NON" 메시지가 출력되지 않는 경우, AVR Loader가 PC와 연결되어 있는지 체크해 본다.

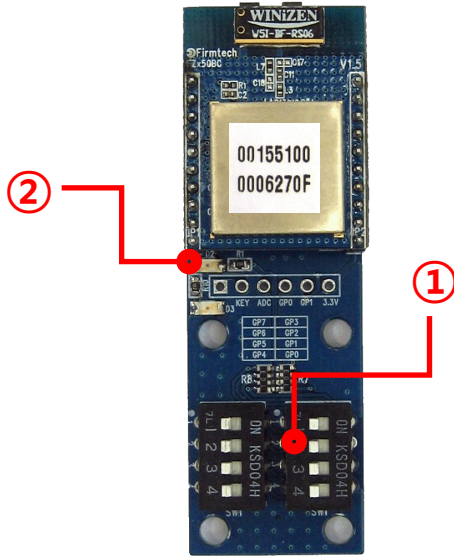
AVR Loader가 FZ800ED_MICOM보드에 연결되어 PC와 연결되어 있는 상태에서, Ponyprog 프로그램이 PC에서 실행되어 있지 않으면, 코디네이터로 설정되어 FZ800ED_MICOM보드에 장착된 FZ750BS 또는 FZ750BC에서 출력되는 시리얼 데이터가 정상적으로 동작되지 않는다.

12. FZ800ED_GPIO_DIP_SWITCH Board에 엔드디바이스 장착



NO	동작
1	엔드디바이스로 설정된 FZ750BS 또는 FZ750BC를 FZ800ED_GPIO_DIP_SWITCH 보드에 장착
2	USB Power Cable 연결하여 PC와 연결
	자동 전원 ON

13. FZ800ED_GPIO_DIP_SWITCH 보드 GPIO 데이터 송신



NO	동작
1	FZ800ED_GPIO_DIP_SWITCH 보드의 Dip Switch 변경
2	FZ750BS 또는 FZ750BC 정해진 시간에 Wake Up(STATUS OK LED OFF)
3	FZ800ED_MICOM(코디네이터)보드 GPIO 데이터 수신
4	약 1초 후 저전력 모드 진입(STSTATUS OK LED ON)
	다음 번 ADC 데이터 송신 가능한 상태

14. FZ800ED_MICOM 보드 GPIO 데이터 수신

```
코디네이터 - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)
*****
FZ800ED_PARSING_GPIO V0.1.0 START
*****
PARSING DATA TYPE      : GPT
PARSING DATA VALUE MIN : 00FD
PARSING DATA VALUE MAX : 00FD
PARSING DATA ADDRESS  : 0000000000000000
*****
COORD START OK
TARGET NON
GPT00FF_001551000000CC0A
GPT00EF_001551000000CC0A
GPT00FE_001551000000CC0A
GPT00FD_001551000000CC0A => DETECT
GPT00BF_001551000000CC0A
GPT00FF_001551000000CC0A
```

- 어드레스 비교 값이 "0000000000000000"임으로 어드레스 비교 안함
- GPIO 데이터 수신 시, GPIO 값이 최소값-1 보다 큰지 비교 진행
- GPIO 데이터 수신 시, GPIO 값이 최대값+1 보다 작으리 비교 진행

< 분석하고자 하는 GPIO 데이터와 같은 데이터를 수신 받으면 >

1. FZ800ED_MICOM 보드는 LED2를 2회 깜빡임
2. BUZZ 울림
3. 시리얼 데이터 출력

어드레스가 001551000000CC0A인 디바이스로 부터 GPIO 데이터가 수신된 것을 분석한 상태

FZ800ED_PARSING_GPIO

프로그램 소스 분석

Main Process : main.c

Parsing Process : parsing_gpio.c

BUZZ Process : buzz.c

FND Process : fnd.c

LCD Process : lcd.c

LED Process : led.c

Utility Process : util.c

15. Main Process : INIT_PORT()

```
void INIT_PORT ()
{
    //포트 A를 출력으로 설정(= 1111 1111)
    DDRA = 0xff;
    //포트 A 초기값 설정
    PORTA = 0x00;
    //
    //포트 B를 출력으로 설정(= 1111 1111)
    DDRB = 0xff;
    //포트 B 초기값 High 설정(= 1111 1111)
    PORTB = 0xff;
    //
    //포트 C의 방향 설정
    DDRC = 0x03;
    //포트 C 초기값 설정
    PORTC = 0x03;
    //
    //포트 D의 방향 설정
    DDRD = 0x38;
    //포트 D 초기값 설정
    PORTD = 0x38;
    //
    //포트 E의 방향 설정
    DDRE = 0x3e;
    //포트 E 초기값 설정
    PORTE = 0x1f;
    //
    //포트 F를 출력으로 설정(= 1111 1111)
    DDRF = 0xff;
    //포트 F 초기값 High 설정(= 1111 1111)
    PORTF = 0xff;
    //
    //포트 G를 출력으로 설정(= xxx1 1111)
    DDRG = 0x00;
    //포트 G 초기값 High 설정(= 1111 1111)
    PORTG = 0xff;
} ? end INIT_PORT ?
```

< ATmega128 포트 초기화 >

1. 포트 A 출력 설정 (DDRA = 0xff)
2. 포트 A 초기값 Low 설정 (PORTA = 0x00)
3. 포트 B 출력 설정 (DDRB = 0xff)
4. 포트 B 초기값 High 설정 (PORTB = 0xff)
5. 포트 G 입력 설정 (DDRG = 0x00)
6. 포트 G 초기값 High 설정 (PORTG = 0xff)
7. FZ800ED_PARSING_KEY에서는, 2개의 LED를 사용
 - LED1은 포트 D_4 사용
 - LED2는 포트 D_5 사용
8. 포트 D_4에 연결된 LED1은, FZ800ED_PARSING_KEY프로그램이 운영되면서 일정한 간격으로 깜박거림
9. 포트 D_5에 연결된 LED2는, FZ800ED_PARSING_KEY프로그램이 운영되면서, 수신 받은 데이터가 분석하고자 하는 데이터인 경우 2회 깜박거림
10. FZ800ED_PARSING_KEY에서는, 1개의 BUZZ 사용
 - BUZZ는 포트 E_5 사용
11. 포트 E_5에 연결된 BUZZ는, LED2가 ON될 때 같이 ON되고 LED2가 OFF될 때 같이 OFF됨

16. Main Process : DISPLAY_EEPROM_DATA()

```
void DISPLAY_EEPROM_DATA()
{
    unsigned int loof_count;
    //
    DISPSTR_UART1((unsigned char *)"*****\r\n");
    DISPSTR_UART1((unsigned char *)"PARSING DATA TYPE :");
    DISPSTR_UART1(received_data_type);
    DISPSTR_UART1((unsigned char *)"\r\n");
    DISPSTR_UART1((unsigned char *)"PARSING DATA VALUE MIN :");
    DISPSTR_UART1(received_data_value_min);
    DISPSTR_UART1((unsigned char *)"\r\n");
    DISPSTR_UART1((unsigned char *)"PARSING DATA VALUE MAX :");
    DISPSTR_UART1(received_data_value_max);
    DISPSTR_UART1((unsigned char *)"\r\n");
    DISPSTR_UART1((unsigned char *)"PARSING DATA ADDRESS :");
    DISPSTR_UART1(received_data_address);
    DISPSTR_UART1((unsigned char *)"\r\n*****\r\n");
    //
    //UART0_BUF 초기화
    p_rx0_wr = 0;
    p_rx0_rd = 0;
    for (loof_count=0; loof_count<UART0_BUF_SIZE; loof_count++) rx0_buf[loof_count] = 0;
    //UART1_BUF 초기화
    p_rx1_wr = 0;
    p_rx1_rd = 0;
    for (loof_count=0; loof_count<UART1_BUF_SIZE; loof_count++) rx1_buf[loof_count] = 0;
} ? end DISPLAY_EEPROM_DATA ?
```

< ATmega128 분석 데이터 출력 >

1. 수신 받은 데이터의 타입을 비교하기 위한 값 출력 (received_data_type)
2. 수신 받은 데이터의 최소 값을 비교하기 위한 값 출력 (received_data_value_min)
3. 수신 받은 데이터의 최대 값을 비교하기 위한 값 출력 (received_data_value_max)
4. 데이터를 송신한 어드레스를 비교하기 위한 값을 출력 (received_data_address)
5. UART0과 UART1의 수신 버퍼로 사용하는 rx0_buf와 rx1_buf와 관련 변수를 초기

17. Main Process : CHECK_CR_LF_PROCESS()

```
void CHECK_CR_LF_PROCESS()
{
    if((cr0_check_flag == 1)&&(lf_check_flag == 1))
    {
        cr0_check_flag = 0;
        lf_check_flag = 0;
        step_count = STEP_3_DATA_PARSING;
    }
}
```

< ATmega128 수신 데이터의 마지막 2바이트 조사 >

1. FZ750BS로부터 수신 받은 데이터의 마지막 데이터는 CR(0x0d)과 LF(0x0a)
2. 수신 받은 데이터의 마지막 데이터가 CR과 LF인 경우, 수신 데이터 분석 함수를 진행할 수 있게끔 스텝을 변경

18. Main Process : CHECK_USERS_INPUT_DATA()

```
void CHECK_USERS_INPUT_DATA()
{
    unsigned int uart1_length;
    unsigned char loof_count;
    //-----
    uart1_length = CHECK_RX_BUF_UART1();
    if(uart1_length > 0)
    {
        for(loof_count = 0; loof_count < uart1_length; loof_count++)
        {
            PUTCHAR_UART0(GETCHAR_UART1());
        }
    }
    //-----
}
```

< ATmega128 사용자 입력 데이터 처리 >

1. UART 1은 사용자(PC)와 연결된 포트
2. 만약 사용자(PC)가 시리얼 데이터를 입력하면, 입력 받은 시리얼 데이터를 UART 0으로 출력
3. UART 0은 FZ750BS와 연결된 포트

19. Main Process : USERS_OPERATION()

```
void USERS_OPERATION()
{
    if(execution_users_operation == EXECUTION_OK)
    {
        //-----
        if(execution_count < 5000)
        {
            LED_2_ON;
            BUZZ_ON();
        }
        else if(execution_count < 10000)
        {
            LED_2_OFF;
            BUZZ_OFF();
        }
        else if(execution_count < 15000)
        {
            LED_2_ON;
            BUZZ_ON();
        }
        else
        {
            LED_2_OFF;
            BUZZ_OFF();
            execution_count = 0;
            received_operation_count--;
            if(received_operation_count == 0)
            {
                execution_users_operation = EXECUTION_NONE;
            }
        }
        //-----
        execution_count++;
        if(execution_count > 15000)
            execution_count = 0;
        //-----
    } ? end if execution_users_operati... ?
    //-----
    if(timer0_counter < 100)
        LED_1_ON;
    else
        LED_1_OFF;
    //-----
} ? end USERS_OPERATION ?
```

< ATmega128 사용자 지정 동작 >

1. 수신 받은 데이터와 분석 하고자 하는 데이터가 같은 경우, LED2를 2회 깜박이게 운영 합니다. LED2가 ON될 때 BUZZ도 ON됩니다. LED2가 OFF될 때 BUZZ도 OFF됩니다.
2. execution_count를 이용하여 카운트하면서 LED2를 2회를 깜박이게 합니다.
3. LED2가 2회 깜박이면, 사용자 지정 동작 (execution_users_operation)을 더 이상 동작하지 않게 설정합니다. (EXECUTION_NONE)
4. 분석하고자 하는 데이터를 수신 받은 경우, 사용자는 이곳에 알맞은 동작을 기술합니다.

< ATmega128 동작 상태 표시 >

1. ATmega128이 정상적으로 동작되는 경우, LED1을 천천히 깜박이게 운영합니다.
2. LED1의 동작은, ATmega128의 Timer0에 의해 규칙적으로 동작됩니다.
3. 만약 사용자 지정 동작에서 시간을 많이 잡고 있는 상태가 발생되면, 동작 상태를 알리는 LED1이 불규칙적으로 깜박일 것입니다.

20. Main Process : main() – 1th

```
int main(void)
{
```

```
//-----
//-----
INIT_PORT();
INIT_UART1(BAUD_115200);
INIT_UART0(BAUD_115200);
INIT_TIMER0();
sei();
//-----
```

```
RESET_PORT_LOW;
//
```

```
cr0_check_flag = 0;
lf_check_flag = 0;
step_count = STEP_0_INIT_EEPROM;
execution_users_operation = EXECUTION_NONE;
received_operation_count = 0;
execution_count = 0;
//-----
```

```
DISPSTR_UART1((unsigned char *)"\r\n*****\r\n");
DISPSTR_UART1((unsigned char *)DEVICE_NAME);
DISPSTR_UART1((unsigned char *)" ");
DISPSTR_UART1((unsigned char *)DEVICE_VERSION);
DISPSTR_UART1((unsigned char *)" START\r\n");
//-----
```

```
INIT_BUZZ();
INIT_LCD();
INIT_FND();
INIT_LED();
INIT_BUZZ();
//
```

```
RESET_PORT_HIGH;
//
```

AVR 포트 초기화
AVR UART 초기화
AVR Timer 초기화
AVR Interrupt 초기화

FZ750BS 모듈 리셋포트 LOW
(FZ750BS 동작 안됨)

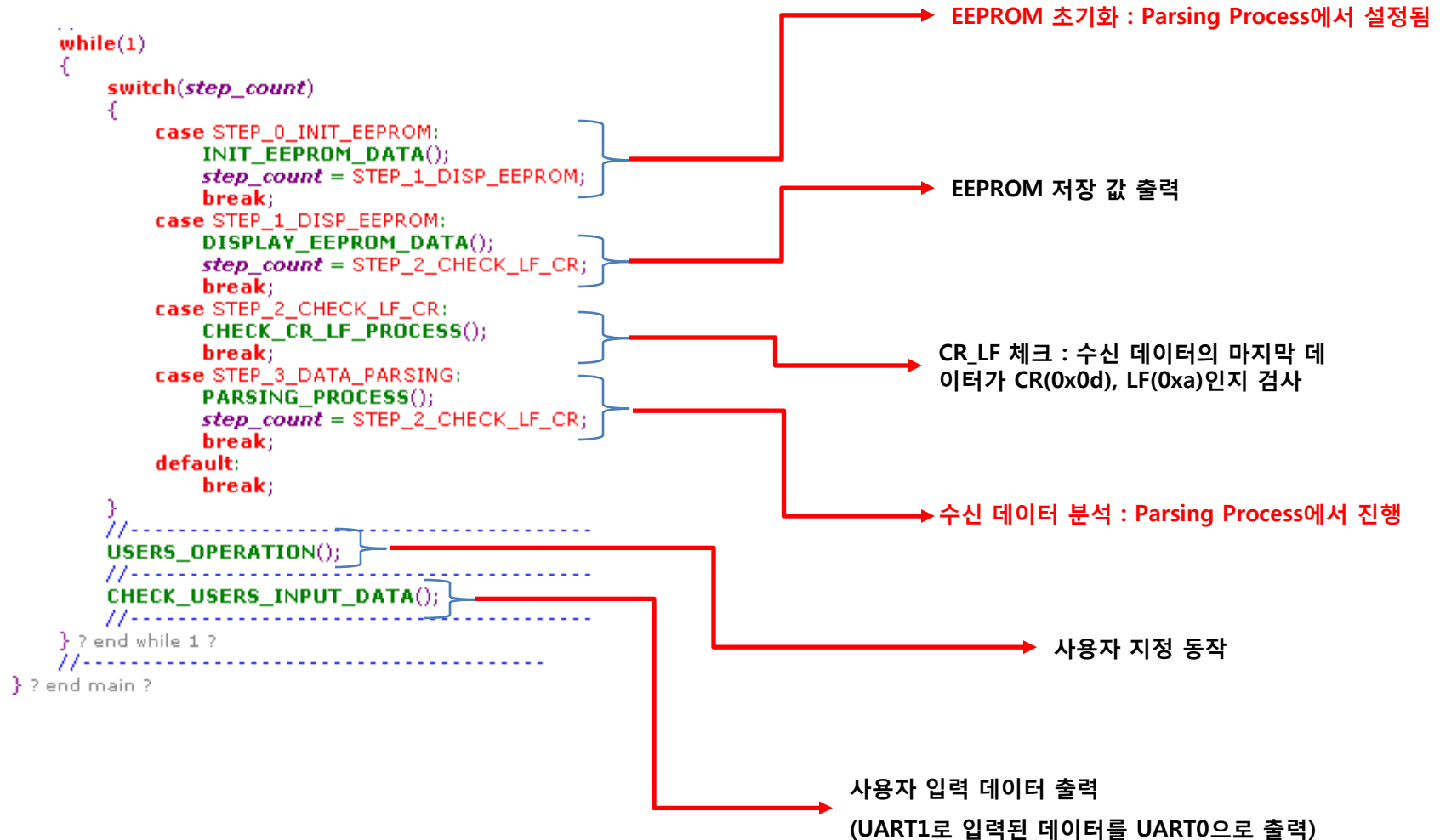
전역변수 초기화

프로그램 정보 출력 : Parsing Process에서 설정됨

BUZZ, LCD, FND, LED 동작 체크

FZ750BS 모듈 리셋포트 High
(FZ750BS 동작 시작)

21. Main Process : main() – 2th



22. Parsing Process : INIT_EEPROM_DATA()

```
void INIT_EEPROM_DATA()
```

```
{
  memset(received_data_type,0x00,4);
  memset(received_data_value_min,0x00,5);
  memset(received_data_value_max,0x00,5);
  memset(received_data_address,0x00,17);
  //-----
  eeprom_read_block(received_data_type, (void *)EEPROM_DATA_TYPE, 3);
  eeprom_read_block(received_data_value_min, (void *)EEPROM_DATA_VALUE_MIN, 4);
  eeprom_read_block(received_data_value_max, (void *)EEPROM_DATA_VALUE_MAX, 4);
  eeprom_read_block(received_data_address, (void *)EEPROM_DATA_ADDRESS, 16);
  //-----
  if( (received_data_type[0] == 0xff)
      &&(received_data_type[1] == 0xff)
      &&(received_data_type[2] == 0xff))
  {
    //-----
    memcpy(received_data_type,"GPT",3);
    memcpy(received_data_value_min,"00FD",4);
    memcpy(received_data_value_max,"00FD",4);
    memcpy(received_data_address,"0000000000000000",16);
    //-----
    eeprom_write_block(received_data_type, (void *)EEPROM_DATA_TYPE, 3);
    eeprom_write_block(received_data_value_min, (void *)EEPROM_DATA_VALUE_MIN, 4);
    eeprom_write_block(received_data_value_max, (void *)EEPROM_DATA_VALUE_MAX, 4);
    eeprom_write_block(received_data_address, (void *)EEPROM_DATA_ADDRESS, 16);
    //-----
    eeprom_read_block(received_data_type, (void *)EEPROM_DATA_TYPE, 3);
    eeprom_read_block(received_data_value_min, (void *)EEPROM_DATA_VALUE_MIN, 4);
    eeprom_read_block(received_data_value_max, (void *)EEPROM_DATA_VALUE_MAX, 4);
    eeprom_read_block(received_data_address, (void *)EEPROM_DATA_ADDRESS, 16);
    //-----
  }
  //-----
} ? end INIT_EEPROM_DATA ?
```

변수 초기화

EEPROM에 저장되어 있는 값 리딩

EEPROM의 초기값이 0xFF인지 검사

1. EEPROM데이터 초기화 진행
2. GPIO 분석용으로 데이터 타입과 최소, 최대 값 저장
3. Address는 초기 00으로 설정
4. Address가 00인 경우, Address는 비교하지 않음
5. 정확한 Address를 입력하면 Address 비교 진행(사용자는 분석하고자 하는 송신 디바이스의 어드레스 16자리 입력)

EEPROM에 초기화 값 라이팅

EEPROM에 저장되어 있는 값 리딩

1. 수신 받은 데이터의 타입이 received_data_type(GPT)과 같은지 비교 진행
2. 수신 받은 데이터의 값이 received_data_value_min-1 보다 큰지 비교 진행
3. 수신 받은 데이터의 값이 received_data_value_max+1 보다 작으지 비교 진행
4. 송신 디바이스의 Address가 received_data_address과 같은지 비교 진행

23. HEXASCII와 INT의 상호 관계

ADC 수신 데이터 : FZ750BS에서 출력되는 값을 직접 하이퍼 터미널에 출력시키면 아래와 같이 출력됨

A	D	C	0	1	F	3	_	0	0	1	5	5	1	0	0	0	0	0	0	0	0	0	B	Wr	Wn
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----

비교를 위해 필요한 ADC 값

하이퍼 터미널에 위와 같이 출력되어 사용자가 인지할 수 있는 상태에서, ADC 값에 해당하는 01F3은 숫자(Integer)가 아니라 16진수(Hex Value)에 해당하는 문자(Character)열임

0	1	F	3
---	---	---	---

사용자가 인지한 값

30	31	46	33
----	----	----	----

실제 값

(Hex ASCII)

값의 대소 비교를 하기 위해서는 숫자(Integer)를 이용해야 함

즉, 문자열을 가지고 대소 비교를 하게 되면 정확한 대소 비교가 진행되지 않음

그러므로, 문자열에 해당하는 값을 숫자(Integer)로 변환해야 함

0	1	F	3
---	---	---	---

||

30	31	46	33
----	----	----	----

변환



00	00	01	F3
----	----	----	----

대소 비교가 가능한 Hex Value의 Integer

ANSI C에서 제공하는, 문자열을 숫자로 변경하는 함수(atoi())는 Hex Value에 해당하는 문자열의 변환이 정확히 이루어 지지 않음 (01F3과 같은 값)

atoi() 함수는 Integer Value에 해당하는 문자열의 변환에 사용됨 (1234와 같은 값)

24. Parsing Process : HEXASCII_TO_INT()

```
unsigned int HEXASCII_TO_INT(unsigned char *hexascii)
```

```
{
    unsigned char temp_char_1,temp_char_2,temp_char_3,temp_char_4;
    unsigned int temp_int;
    //-----
    temp_char_1 = *(hexascii);
    temp_char_2 = *(hexascii+1);
    temp_char_3 = *(hexascii+2);
    temp_char_4 = *(hexascii+3);
    //-----
    if(temp_char_1 <= 0x39)
    {
        temp_char_1 = temp_char_1 - 0x30;
        temp_char_1 = temp_char_1 << 4;
    }
    else
    {
        temp_char_1 = temp_char_1 - 0x37;
        temp_char_1 = temp_char_1 << 4;
    }
    temp_int = temp_char_1;
    //-----
    if(temp_char_2 <= 0x39)
    {
        temp_char_2 = temp_char_2 - 0x30;
        temp_char_2 = temp_char_2;
    }
    else
    {
        temp_char_2 = temp_char_2 - 0x37;
        temp_char_2 = temp_char_2;
    }
    temp_int |= temp_char_2;
    //-----
    temp_int = temp_int << 8;
    //-----
    if(temp_char_3 <= 0x39)
    {
        temp_char_3 = temp_char_3 - 0x30;
        temp_char_3 = temp_char_3 << 4;
    }
    else
    {
        temp_char_3 = temp_char_3 - 0x37;
        temp_char_3 = temp_char_3 << 4;
    }
    temp_int |= temp_char_3;
    //-----
    if(temp_char_4 <= 0x39)
    {
        temp_char_4 = temp_char_4 - 0x30;
        temp_char_4 = temp_char_4;
    }
    else
    {
        temp_char_4 = temp_char_4 - 0x37;
        temp_char_4 = temp_char_4;
    }
    temp_int |= temp_char_4;
    //-----
    return temp_int;
} ? end HEXASCII_TO_INT ?
```

4바이트 Hex ASCII 값을 temp_char_1/2/3/4에 한바이트씩 저장

1번째 Hex ASCII 값을 숫자(Integer)로 변환하여 temp_int에 저장

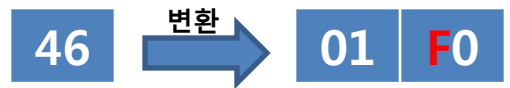
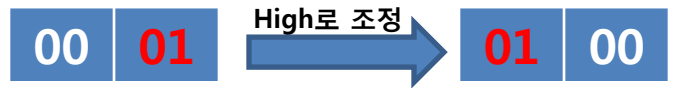
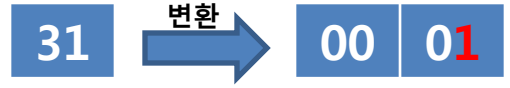
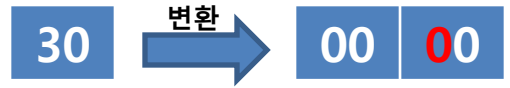
2번째 Hex ASCII 값을 숫자(Integer)로 변환하여 temp_int에 추가(더하기)

1 / 2번째 변환된 값을 High 비트로 조정

3번째 Hex ASCII 값을 숫자(Integer)로 변환하여 temp_int에 추가(더하기)

4번째 Hex ASCII 값을 숫자(Integer)로 변환하여 temp_int에 추가(더하기)

0	1	F	3
30	31	46	33



변환 완료

01	F3
----	----

25. Parsing Process : PARSING_PROCESS() – 1th

```
void PARSING_PROCESS()
{
    unsigned int uart0_length;
    unsigned char uart0_parsing_buf[60];
    unsigned char loof_count;
    //
    unsigned char value_buf[4];
    unsigned int received_value;
    unsigned int min_value;
    unsigned int max_value;
    //
    //-----
    memset(uart0_parsing_buf,0x00,60);
    uart0_length = CHECK_RX_BUF_UART0();
    if(uart0_length > 0)
    {
        for(loof_count = 0;loof_count < uart0_length;loof_count++)
        {
            uart0_parsing_buf[loof_count] = GETCHAR_UART0();
        }
        //-----
        memcpy(value_buf,&uart0_parsing_buf[uart0_length-23],4);
        received_value = HEXASCII_TO_INT(value_buf);
        min_value = HEXASCII_TO_INT(received_data_value_min);
        max_value = HEXASCII_TO_INT(received_data_value_max);
        //-----
    }
}
```

UART0으로 입력된 데이터가 있는 경우, 데이터 수집
UART0은 FZ750BS와 연결된 포트

즉, FZ750BS가 무선으로부터 데이터를 수신 받으면
시리얼 포트에 출력하고, 출력된 시리얼 데이터는
ATMega128의 UART0에 입력됨

수집된 데이터의 4번째 데이터부터 4바이트 value_buf에 복사
value_buf값을 Integer로 변환하여 received_value에 복사
분석용 최소 값을 Integer로 변환하여 min_value에 복사
분석용 최대 값을 Integer로 변환하여 max_value에 복사

26. Parsing Process : PARSING_PROCESS() – 2th

```
if(!memcmp(received_data_address,"0000000000000000",16))
{
    if( (!memcmp(received_data_type,&uart0_parsing_buf[uart0_length-26],3))
    &&(received_value > min_value-1)
    &&(received_value < max_value+1))
    {
        execution_users_operation = EXECUTION_OK;
        received_operation_count++;
        //-----
        if(received_operation_count > 60000)
            received_operation_count = 0;
        //
        for(loof_count = 0;loof_count < uart0_length-2;loof_count++)
            PUTCHAR_UART1(uart0_parsing_buf[loof_count]);
        DISPSTR_UART1("=> DETECT\r\n");
    }
    else
    {
        DISPSTR_UART1(uart0_parsing_buf);
    }
}
? end if !memcmp(received_data... ?
//-----
else
{
    if( (!memcmp(received_data_type,&uart0_parsing_buf[uart0_length-26],3))
    &&(!memcmp(received_data_address,&uart0_parsing_buf[uart0_length-18],16))
    &&(received_value > min_value-1)
    &&(received_value < max_value+1))
    {
        execution_users_operation = EXECUTION_OK;
        received_operation_count++;
        //-----
        if(received_operation_count > 60000)
            received_operation_count = 0;
        //
        for(loof_count = 0;loof_count < uart0_length-2;loof_count++)
            PUTCHAR_UART1(uart0_parsing_buf[loof_count]);
        DISPSTR_UART1("=> DETECT\r\n");
    }
    else
    {
        DISPSTR_UART1(uart0_parsing_buf);
    }
}
? end else ?
```

< 분석용 어드레스가 00인 경우 진행(어드레스 비교 진행 안 함) >

분석용 데이터 타입과 수신 데이터의 첫 3바이트가 같은지 비교
분석용 최소 값-1 보다 수신 데이터의 값이 큰지 비교
분석용 최대 값+1 보다 수신 데이터의 값이 작은지 비교
분석용 데이터의 조건에 맞는 경우, EXECUTION_OK 설정
동작 카운트 +1 진행
동작 카운트가 60000 이상인 경우 0으로 재 설정
분석용 데이터의 조건에 맞는 경우, "수신 데이터 + DETECT" 출력
분석용 데이터가 아닌 경우, "수신 데이터" 출력

< 분석용 어드레스가 있는 경우 진행 >

분석용 데이터 어드레스와 수신 데이터의 어드레스가 같은지 비교
분석용 데이터 타입과 수신 데이터의 첫 3바이트가 같은지 비교
분석용 최소 값-1 보다 수신 데이터의 값이 큰지 비교
분석용 최대 값+1 보다 수신 데이터의 값이 작은지 비교
분석용 데이터의 조건에 맞는 경우, EXECUTION_OK 설정
동작 카운트 +1 진행
동작 카운트가 60000 이상인 경우 0으로 재 설정
분석용 데이터의 조건에 맞는 경우, "수신 데이터 + DETECT" 출력
분석용 데이터가 아닌 경우, "수신 데이터" 출력

27. BUZZ Process

```
void BUZZ_ON()
{
    sbi(PORTE,PE5);
}
```

```
void BUZZ_OFF()
{
    cbi(PORTE,PE5);
}
```

```
void INIT_BUZZ()
{
    BUZZ_ON();
    WAIT_1MS(20);
    BUZZ_OFF();
    WAIT_1MS(10);
    BUZZ_ON();
    WAIT_1MS(10);
    BUZZ_OFF();
}
```

```
void BUZZ_PROCESS()
{
}
```

현재, BUZZ_PROCESS()는 따로 진행하지 않음
단순히 BUZZ의 ON/OFF만 동작시킴

차후, BUZZ_PROCESS 진행 프로그램 제공 예정

28. LCD Process

```
void LCD_INITIALIZE()
{
    WAIT_1MS(40);
    //
    LCD_WRITE_COMMAND(0x38);
    WAIT_1MS(5);
    //
    WAIT_1MS(20);
    //
    LCD_WRITE_COMMAND(0x38);
    WAIT_1MS(5);
    //
    WAIT_1MS(1);
    //
    LCD_WRITE_COMMAND(0x38);
    WAIT_1MS(5);
    //
    LCD_WRITE_COMMAND(0x38);
    WAIT_1MS(5);
    //
    LCD_WRITE_COMMAND(0x0f);
    WAIT_1MS(5);
    //
    LCD_WRITE_COMMAND(0x06);
    WAIT_1MS(5);
    //
    LCD_WRITE_COMMAND(0x02);
    WAIT_1MS(5);
    //
    LCD_WRITE_COMMAND(0x01);
    WAIT_1MS(5);
} // end LCD_INITIALIZE ?
```

```
void LCD_CLEAR_SCREEN()
{
    LCD_WRITE_COMMAND(0x01);
    WAIT_1MS(5);
    //
    WAIT_1MS(10);
}
```

```
void LCD_CURSOR_OFF()
{
    LCD_WRITE_COMMAND(0x0c);
    WAIT_1MS(5);
    //
    WAIT_1MS(10);
}
```

```
void LCD_WRITE_COMMAND(unsigned char lcd_command)
{
    LCD_EN_0;
    LCD_RW_0;
    LCD_RS_0;
    //
    LCD_EN_1;
    LCD_RW_0;
    LCD_RS_0;
    //
    PORTF = lcd_command;
    //
    LCD_EN_0;
    LCD_RW_0;
    LCD_RS_0;
}
```

```
void LCD_WRITE_DATA(unsigned char lcd_data)
{
    LCD_EN_0;
    LCD_RW_0;
    LCD_RS_1;
    //
    LCD_EN_1;
    LCD_RW_0;
    LCD_RS_1;
    //
    PORTF = lcd_data;
    //
    LCD_EN_0;
    LCD_RW_0;
    LCD_RS_1;
}
```

```
void LCD_DISPLAY_STRING(unsigned char *str)
{
    while(*str)
    {
        LCD_WRITE_DATA(*str++);
        WAIT_1MS(5);
    }
}
```

```
void INIT_LCD()
{
    LCD_INITIALIZE();
    LCD_CLEAR_SCREEN();
    LCD_WRITE_COMMAND(0x80);
    WAIT_1MS(5);
    LCD_DISPLAY_STRING("Firmtech.co.,LTD");
    LCD_WRITE_COMMAND(0xc0);
    WAIT_1MS(5);
    LCD_DISPLAY_STRING("FZ800ED_KEY_V010");
    LCD_CURSOR_OFF();
}
```

```
void LCD_PROCESS()
{
}
```

현재, LCD_PROCESS()는 따로 진행하지 않음
단순히 초기 문자만 출력시킴
차후, LCD_PROCESS 진행 프로그램 제공 예정

29. FND Process

```
void INIT_FND()  
{  
    PORTA = FND_ON_0;  
    WAIT_1MS(70);  
    PORTA = FND_ON_1;  
    WAIT_1MS(70);  
    PORTA = FND_ON_2;  
    WAIT_1MS(70);  
    PORTA = FND_ON_3;  
    WAIT_1MS(70);  
    PORTA = FND_ON_4;  
    WAIT_1MS(70);  
    PORTA = FND_ON_5;  
    WAIT_1MS(70);  
    PORTA = FND_ON_6;  
    WAIT_1MS(70);  
    PORTA = FND_ON_7;  
    WAIT_1MS(70);  
    PORTA = FND_ON_8;  
    WAIT_1MS(70);  
    PORTA = FND_ON_9;  
    WAIT_1MS(70);  
    PORTA = FND_ON_0;  
} ? end INIT_FND ?
```

```
void FND_PROCESS()  
{  
  
}
```

현재, FND_PROCESS()는 따로 진행하지 않음
단순히 초기 카운트 동작만 진행

차후, FND_PROCESS 진행 프로그램 제공 예정

30. LED Process

```
void INIT_LED()
{
  PORTB = LED_ON_0;
  WAIT_1MS(70);
  PORTB = LED_ON_1;
  WAIT_1MS(70);
  PORTB = LED_ON_2;
  WAIT_1MS(70);
  PORTB = LED_ON_3;
  WAIT_1MS(70);
  PORTB = LED_ON_4;
  WAIT_1MS(70);
  PORTB = LED_ON_5;
  WAIT_1MS(70);
  PORTB = LED_ON_6;
  WAIT_1MS(70);
  PORTB = LED_ON_7;
  WAIT_1MS(70);
  PORTB = LED_OFF;
} ? end INIT_LED ?
```

```
void LED_PROCESS()
{
}
```

현재, LED_PROCESS()는 따로 진행하지 않음
단순히 초기 순차적 ON/OFF 동작

차후, LED_PROCESS 진행 프로그램 제공 예정

31. Utility Process : UART 0 Function – 1th

SIGNAL(SIG_UART0_RECVD)

```
{  
    rx0_buf[p_rx0_wr++] = UDR0;  
    if(rx0_buf[p_rx0_wr-1] == CR_VALUE)cr0_check_flag++;  
    if(rx0_buf[p_rx0_wr-1] == LF_VALUE)lf_check_flag++;  
    if (p_rx0_wr > UART0_BUF_SIZE-1)p_rx0_wr = 0;  
}
```

```
//
```

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*/  
    UCSRB = (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 interrupt vector >

1. UART0으로 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. 수신 데이터가 CR(0x0d)인 경우 플래그 체크
5. 수신 데이터가 LF(0x0a)인 경우 플래그 체크

< ATmega128 UART0초기화 >

1. UBRR0L 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

32. Utility Process : UART 0 Function – 2th

```
unsigned char PUTCHAR_UART0 (unsigned char c)
{
    while (!(UCSR0A & 0x20)) ;
    UDR0 = c;
    return 0;
}
//
void DISPSTR_UART0 (unsigned char *s)
{
    while (*s != '\0')
        PUTCHAR_UART0(*s++);
}
//
unsigned int CHECK_RX_BUF_UART0(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 GETCHAR_UART0 (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;
}
//
void DISPLAY_CR_LF_UART0()
{
    PUTCHAR_UART0(0x0d);
    PUTCHAR_UART0(0x0a);
}
```

< **PUTCHAR_UART0**() >

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

< **DISPSTR_UART0**() >

1. 문자열로 된 시리얼 데이터를 UART0 포트로 출력

< **CHECK_RX_BUF_UART0**() >

1. UART0로 입력된 데이터는 rx0_buf[] 버퍼에 저장
2. CHECK_RX_BUF_UART0() 함수는 rx0_buf[] 버퍼에 저장되어 있는 데이터의 개수를 체크하는 함수

< **GETCHAR_UART0**() >

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

< **DISPLAY_CR_LF_UART0**() >

1. UART0으로 CR(0Xd)와 LF(0x0a)를 출력하는 함수

33. Utility Process : UART 1 Function – 1th

```
SIGNAL(SIG_UART1_RECV)
{
    rx1_buf[p_rx1_wr++] = UDR1;
    if (p_rx1_wr > UART1_BUF_SIZE-1) p_rx1_wr = 0;
}
//
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 UART0 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. 수신 데이터가 CR(0x0d)인 경우 플래그 체크

< ATmega128 UART1초기화 >

1. UBRR1L 설정에 따른 시리얼 데이터 통신 속도 설정
2. 수신 완료 인터럽트 허용 (RXCIE)
3. UART1 수신 부 동작 허용 (RXEN)
4. UART1 송신 부 동작 허용 (TXEN)
5. 수신 데이터 저장 버퍼 및 수신 데이터 포인터 초기값 설정

34. Utility Process : UART 1 Function – 2th

```
unsigned char PUTCHAR_UART1 (unsigned char c)
{
    while (!(UCSR1A & 0x20));
    UDR1 = c;
    return 0;
}
//
void DISPSTR_UART1 (unsigned char *s)
{
    while (*s != '\0')
        PUTCHAR_UART1(*s++);
}
//
unsigned int CHECK_RX_BUF_UART1(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 GETCHAR_UART1 (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;
}
//
void DISPLAY_CR_LF_UART1()
{
    PUTCHAR_UART1(0x0d);
    PUTCHAR_UART1(0x0a);
}
```

< **PUTCHAR_UART1**() >

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

< **DISPSTR_UART1**() >

1. 문자열로 된 시리얼 데이터를 UART1 포트로 출력

< **CHECK_RX_BUF_UART1**() >

1. UART1로 입력된 데이터는 rx1_buf[] 버퍼에 저장
2. CHECK_RX_BUF_UART1() 함수는 rx1_buf[] 버퍼에 저장되어 있는 데이터의 개수를 체크하는 함수

< **GETCHAR_UART1**() >

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

< **DISPLAY_CR_LF_UART1**() >

1. UART1으로 CR(0x0d)와 LF(0x0a)를 출력하는 함수

35. Utility Process : Timer 0 Function

```
SIGNAL(SIG_OVERFLOW0)
{
    timer0_counter++;
    if (timer0_counter > 200 )
    {
        timer0_counter = 0;
    }
}
//
void INIT_TIMER0(void)
{
    TIMSK |= 1 << TOIE0;
    TCNT0 = 0;
    TCCR0 = 5;
    timer0_counter = 0;
}
```

< ATmega128 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

< ATmega128 Timer0 초기화 >

1. Timer0의 인터럽트 허용 설정 (TIMSK |=1<<TOIE0)
2. Timer0의 카운터는 0부터 시작 (TCNT0 = 0)
3. Timer0의 분주 비 128 설정 (TCCR0 = 5 = 0x05 = 0000 0101)
4. Application에서 사용하기 위한 timer0_count = 0 설정

36. Utility Process : Other Function

```
unsigned char HEX2CHAR(unsigned char c)
{
    if ((c == 0 || c > 0) && (c < 10))
        return '0' + c;
    if (c >= 10 && c <= 15)
        return 'A' + c - 10;
    //
    return c;
}
```

< 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 0x1은 Char 타입의 1(HEX값으로 0x31)로 변경됨

```
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++) ;
}
```

< WAIT_1MS() >

1. 약 1ms동안 대기하는 함수

< WAIT_1US() >

1. 약 1us동안 대기하는 함수

FZ800ED_PARSING_GPIO

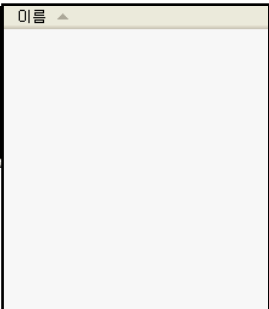
프로그램 컴파일

37. 소스 컴파일은 WINAVR 사용

```
C:\WINDOWS\system32\cmd.exe
C:\FIRMTECH\FZ800ED\#Application\FZ800ED_PARSING_GPIO_U0.1.0>make clean
-----
begin
rm -f RESULT/*.*
-----
end
-----
C:\FIRMTECH\FZ800ED\#Application\FZ800ED_PARSING_GPIO_U0.1.0>make
-----
begin
avr-gcc --version
avr-gcc (GCC) 4.2.2 (WinAVR 20071221)
Copyright (C) 2007 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

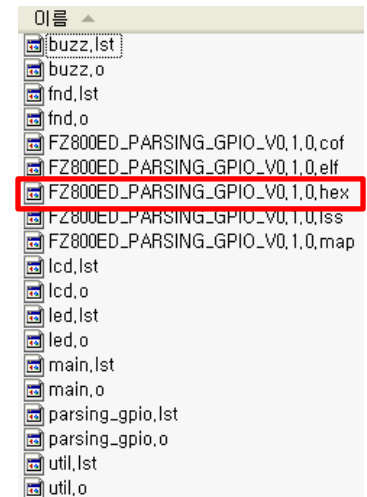
#@-avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O ihex FZ800ED_PARSING_GPIO_U0.1.0.elf FZ800ED_PARSING_GPIO_U0.1.0.eep
avr-objcopy --debugging --change-section-address .data-0x800000 --change-section-address .bss-0x800000 --change-section-address .noinit-0x800000 --change-section-address .eeprom-0x810000 -O coff-avr FZ800ED_PARSING_GPIO_U0.1.0.elf FZ800ED_PARSING_GPIO_U0.1.0.cof
Warning: file C:/WINDOWS/TEMP/ccNQGKrd.s not found in symbol table, ignoring
Warning: ignoring function __vectors() outside any compilation unit
Warning: ignoring function __bad_interrupt() outside any compilation unit
avr-objcopy: --change-section-vma .eeprom+0xff7f0000 never used
avr-objcopy: --change-section-lma .eeprom+0xff7f0000 never used
avr-objcopy: --change-section-vma .noinit+0xff800000 never used
avr-objcopy: --change-section-lma .noinit+0xff800000 never used
cp FZ800ED_PARSING_GPIO_U0.1.0.cof RESULT/FZ800ED_PARSING_GPIO_U0.1.0.cof
Size after:
text    data    bss     dec     hex filename
0       7408    0       7408    1cf0 FZ800ED_PARSING_GPIO_U0.1.0.hex
-----
end
C:\FIRMTECH\FZ800ED\#Application\FZ800ED_PARSING_GPIO_U0.1.0>
```

make clean 실행 이후 RESULT 폴더



1. 사용 컴파일러
 - 사용하는 컴파일러는 WINAVR 사용
 - <http://sourceforge.net> 에서 다운로드 하여 설치
2. 컴파일에 사용하는 파일(PARSING_KEY)
 - main.c, main.h, parsing_gpio.c, parsing_gpio.h, lcd.c, lcd.h, fnd.c, fnd.h, buzz.c, buzz.h, util.c, util.h
3. 컴파일 방법
 - Window Command 창을 이용하여 컴파일에 사용할 파일이 있는 위치로 이동
 - 소스를 수정한 경우, make clean 입력 후 Enter Key 입력
 - make 입력 후 Enter Key 입력
 - ERROR 없이 진행된 경우, HEX 파일과 기타 파일 생성

make 실행 이후 RESULT 폴더



Memo