【ZBM v1.2 활용】

무선 음성전송 예제 프로젝트

OL마이크로웨이브

http://olmicrowaves.com

목 차

1. 개요

IEEE 802.15.4 및 지그비 기술을 활용한 음성 전송이나 음성 통신에 대한 다양한 분야에서의 요구가 증대되고 있고, 많은 연구가 진행되고 있다. 본 예제 프로젝트에서는 **IEEE 802.15.4** 기술에 기반한 무선 음성전송시스템을 구현해 본다. 무선 음성전송시스템은 메인컴퓨터(PC), 서버노드, 고정노드, 도터모듈로 구성되어, 메인컴퓨터에서 녹음한 음성파일(WAV)로부터 PCM 데이터를 추출하여 원격지에 무선으로 전송하고, 원격지의 도터모듈 내부의 DAC&스피커부에서 음성을 재생하여 출력하는 기능을 수행한다. 따라서 본 예제 프로젝트를 통하여 다음과 같은 기술에 용이하게 접근할 수 있다.

- **IEEE 802.15.4** 및 지그비 기술
- **AVR(Atmega128)** 설계 및 펌웨어 제작
- **비주얼C++ MFC** 활용한 PC GUI 응용프로그래밍 제작
- **비주얼C++ MFC** 쓰레드 기법 및 시리얼통신 기술
- **WAV** 형식의 음성 녹음 및 재생, **PCM** 데이터 추출
- **PCM** 데이터의 **DAC** 처리 및 음성 재생

본 자료를 통해 지그비 무선 음성전송 기술을 기반으로 논문을 준비중이거나 응용시스템을 제작하고자 하는 사용자들에게 “Hello!” 역할과 더불어 지그비 활성화를 위한 다양한 아이디어가 제공되기를 기대한다.

2. 활용 분야

지그비는 기본적으로 저속, 저전력 소모, 근거리 통신을 목표로 한다. 음성과 같은 대량의 정보를 전송하는 데는 상당한 시간이 소요되거나 전력의 손실이 커지기 마련이다. 따라서 지그비를 활용한 무선 음성전송시스템은 저속의 전송속도에도 충분한 효과를 발휘할 수 있는 응용분야를 발굴하여 적용하는 것이 바람직하다.

- **안전관리 및 재해예방**
  - ■ 산악 경보 방송
  - ■ 응급 구조 요청
- **음성안내**
  - ■ 노약자, 장애인 음성 안내
  - ■ 공원, 야외시설 안내 방송
- **컨퍼런스**
  - ■ 회의용 무선 오디오
- **핸즈프리**

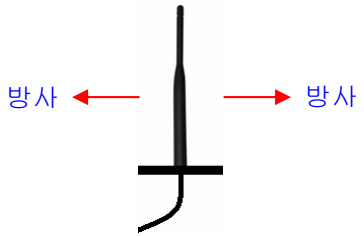
3. 시스템 설치에 관한 고찰

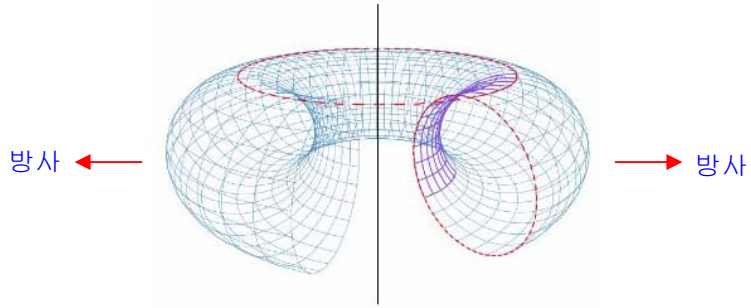
전파를 매개체로하는 무선음성전송시스템은 전달 경로의 구조와 환경 뿐만 아니라 송수신안테나의 방사특성에도 많은 영향을 받는다.
그러므로 신뢰성있는 시스템 구축을 위해서는 이러한 특성을 잘 분석하고 대처해야한다.

3-1. 송수신안테나의 전파 방사특성 파악

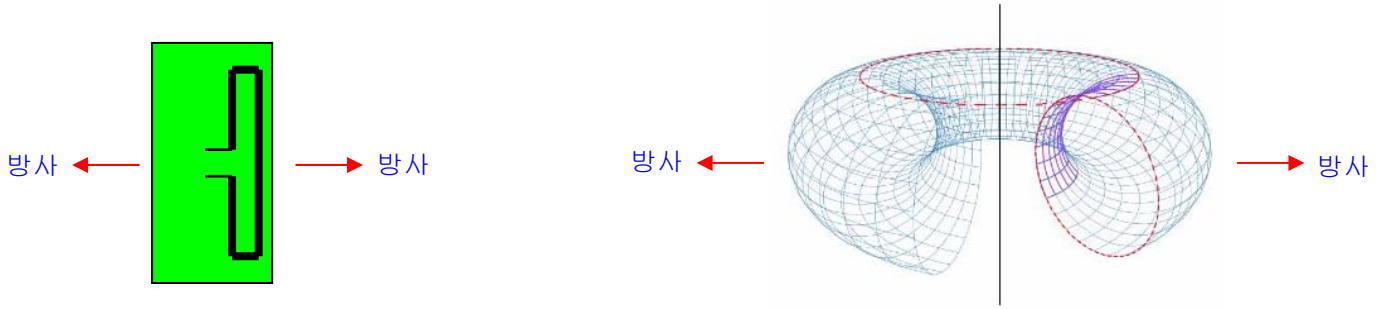
고정노드나 서버노드에 사용되는 안테나의 종류는 형태에 따라 모노폴, 다이폴, 역F형, 칩안테나 등이 있으며, 일반적으로 모노폴은 RF케이블을 통해 회로기판의 외부에 장착하는 경우에 사용하고, 다이폴이나 역F형은 회로기판에 PCB패턴으로 구현하며, 칩안테나는 회로기판에 부착하여 사용한다. 이론적인 관점에서 모노폴의 경우는 안테나의 설치 방향에 따라, PCB형의 경우는 회로기판의 방향에 따라 전파의 세기가 달라지므로 노드의 배치시에 이 점을 고려하여야 한다. 즉, 송수신을 하는 두 노드 간의 안테나의 방향에 따라서 전파의 세기에 영향을 미칠 수 있다. 서로 동일한 평면으로 마주보는 경우에는 전파의 세기가 최대로 되고, 마주보지 않거나 동일한 평면이 아닌 경우에는 상대적으로 전파의 세기가 감소한다. 그러므로 사용하는 노드의 안테나의 종류를 파악하고 방사특성을 염두에두고 설치 및 운용하는 것이 바람직하다.

- **모노폴안테나의 방사특성**

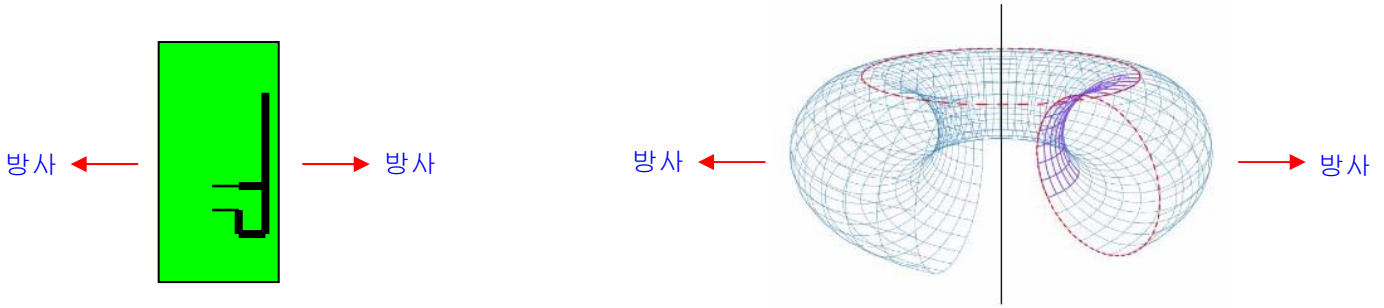




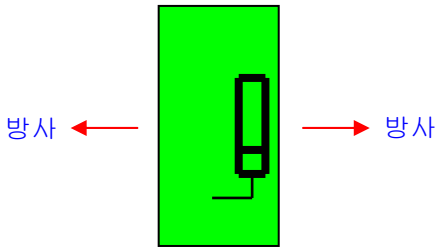
- 다이폴안테나의 방사특성

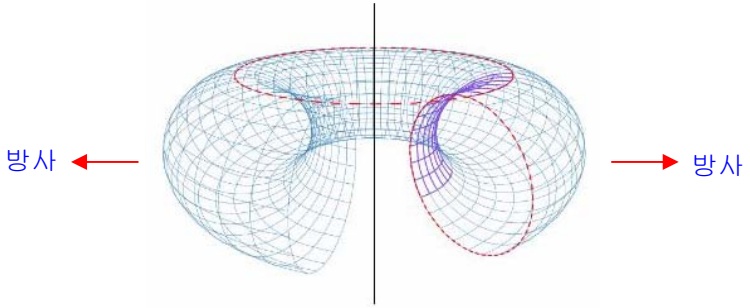


- 역F안테나의 방사특성



- **칩안테나의 방사특성**





위의 그림에 표시된 방사특성은 이상적인 경우이고 실제로는 다소 혹은 상당히 불규칙한 경우가 일반적이다. 특히, PCB형 안테나의 경우 부품이 실장되는 방향(그림에서 왼쪽 방향)은 부품과 기판의 영향으로 상당히 불안정하게 감쇄가 일어나는 경우가 많고, 수직 방향(그림에서 위쪽과 아래쪽 방향)으로는 이론적으로는 통신이 불가능하지만 실제로는 불안정하지만 다소 혹은 상당한 방사 특성을 보이는 경우가 대부분이다. 따라서 노드 배치시에 가급적이면 이러한 불규칙한 방향은 피하는 것이 위치추적의 신뢰도 향상에 도움이 된다.

- **방사 전력**

각 노드의 서비스 범위에 따라서 출력 전력을 적절히 조절한다. 일반적으로 0dBm(1mW) 내외의 출력을 사용하고, 나무나 건물 등의 전파 장애물이 많은 환경에서는 10mW 정도의 보다 높은 출력을 고려해볼 필요도 있다. 10mW 이상의 출력은 지그비의 기본 취지에 맞지 않고 또한 원칙적으로 전파형식검정의 문제가 있어 적절하지 않다.

3-2. 장소의 분석

무선음성전송시스템을 설치하고자 하는 장소의 전파 특성을 분석하는 것은 상당히 중요한 과정 중 하나라고 할 수 있으며, 전파의 전달에 영향을 줄 수 있는 여러가지 요소들을 파악하여 고정노드의 배치에 참조하도록 한다.

- **장애물**

고정 및 이동 장애물의 종류 및 위치, 각 노드 간의 LOS(Line of sight, 가시거리) 확보 여부

- **건물의 구조**

창문 및 출입문의 위치와 크기, 천정의 높이, 통로의 구조, 계단의 위치 등

- **건물의 재질**

벽, 문, 창문의 재질 및 두께

- **전파 간섭**

주변에 무선랜이나 지그비 등, 동일한 주파수 대역(2.4GHz)의 시스템이 운용되고 있으면 무선음성전송시스템과 상호 전파의 간섭은 피할수 없다. 이러한 경우에는 기존에 운용되고 있는 시스템의 재설치나 출력의 감소를 고려해볼 수 있고, 기존의 지그비 시스템이나 설치하고자 하는 무선음성전송시스템의 주파수 채널의 변경을 시도해 볼 수 있다.

3-3. 고정노드의 배치

장소를 분석하는 과정에서 고정노드를 설치할 장소를 선정하고 구체적인 설치 방법을 결정한다.

- **설치 장소**

고정노드의 설치 장소는 다음과 같은 순서로 선정한다.

1) 무선 음성 전송이 필요한 주요지점을 설정한다.
2) 주요지점 간에 통신이 단절되는 경우에는 중간지점을 추가로 설정하여 고정노드들과 서버노드가 Cascade 형태로 네트워크를 형성할 수 있도록 한다.
3) 설정된 지점이 과도하게 많거나 적은 경우에는 적절하게 증감한다.

- **설치 방법**

고정노드의 설치시 가급적 장애물의 교란으로부터 최대한 LOS를 확보하는 것이 유리하다. 또 하나의 고려할 점은 안테나의 방향이다. 앞의 송수신안테나의 전파 방사특성에서 설명한 사항을 염두에 두고 고정노드의 방향을 설정해야 한다.

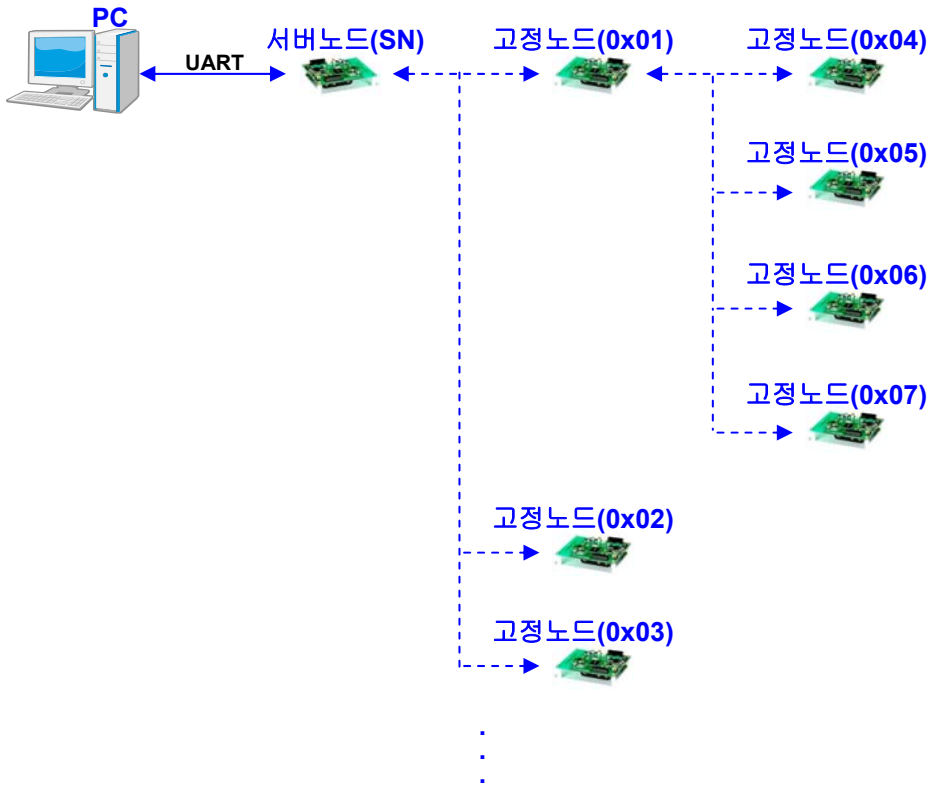
4. 시스템 설치 및 운용

무선음성전송시스템을 운용하기에 앞서, 먼저 활용 분야를 설정하여야 한다. 활용 분야에 따라서 장소의 특징이나 고정노드의 배치 등이 다르기 때문에 해당 분야에 맞는 최적의 시스템을 구성하는 것이 바람직하다.
무선음성전송시스템을 운용하는 절차는, 시스템의 설치 및 기동 단계, 시스템 운용 단계로 구분되어진다.

4-1. 시스템 설치 및 기동 단계

- **네트워크 구성**

3-2절의 장소의 분석과 3-3절의 고정노드의 배치를 참조하여 고정노드의 설치 장소와 네트워크 구성을 결정하고, 전체 시스템의 구성도를 작성해서 시스템이 운용되는 동안 이 구성도를 기본으로 시스템을 관리해야 한다.



[네트워크 구성의 예]

네트워크 구성을 바탕으로 라우팅테이블을 제작한다. 라우팅테이블은 여러 방식이 있겠지만, 본 예제 시스템에서는 간단한 예제로서 다음과 같은 방식을 고안하여 사용한다.

Destination	No. of stages	Stage 1	Stage 2	Stage 3	…
0x01	1	0x01			
0x02	1	0x02			
0x03	1	0x03			
0x04	2	0x01	0x04		
0x05	2	0x01	0x05		
0x06	2	0x01	0x06		
0x07	2	0x01	0x07		
…					

[라우팅테이블의 예]

각 노드에 정보를 전달하기 위해서는 송수신 패킷의 메시지 구조가 정해져야 한다. 본 예제 시스템에서 정보패킷의 흐름은, 메인컴퓨터에서 시작하여 서버노드와 여러 스테이지의 고정노드를 거쳐 최종의 고정노드에 이르는 하향 패킷 흐름과, 반대로 최종의 고정노드에서 시작하여 여러 스테이지의 고정노드와 서버노드를 역순으로 거쳐서 메인컴퓨터에 이르는 상향 패킷 흐름으로 구분된다. 즉, 메인컴퓨터에서 음성데이터를 포함하는 하향 패킷을 내려 보내면 음성 출력 결과가 포함된 상향 패킷의 응답이 올라오게 된다.

하향 패킷은 명령, 스테이지 수(NOS), 스테이지별 고정노드 주소, 패키지 번호, PCM데이터로 구성되어서, 메인컴퓨터로부터 하위 스테이지로 전달되면서 NOS 값이 1씩 감소하고 스테이지별 고정노드 주소가 하나씩 당겨져서 최종의 고정노드에게는 명령, 스테이지 수(NOS), 패키지

번호, PCM데이터만 전달되는 방식으로 구현된다.

하향 패킷 (메인컴퓨터 → 서버노드)

구 분	Byte 0	Byte 1	Byte2	Byte 3~4	Byte 5~6	Byte 7~8	Byte 9~108
PC → SN	L	C	NOS(=2)	(FNst1)	(FNst2)	NOP	PCM 데이터

하향 패킷 (서버노드 → 고정노드1)

구 분	Byte 0	Byte1	Byte 2~3	Byte 4~5	Byte 6~105
SN → FNst1	C	NOS(=1)	(FNst2)	NOP	PCM 데이터

하향 패킷 (고정노드1 → 고정노드2)

구 분	Byte 0	Byte1	Byte 2~3	Byte 4~103
FNst1 → FNst2	C	NOS(=0)	NOP	PCM 데이터

► PC : 메인컴퓨터

► SN (Server Node) : 서버노드

► L (Length) : 자신을 제외한 패킷의 바이트 수

► C (Command) : 명령코드

1 → 음성데이터를 갱신하라

► NOS (Number Of Stage) : 고정노드 단계 수

► (FNst#) (Fixed Node Address) : #스테이지의 고정노드 어드레스

► NOP (Number Of Package) : PCM 데이터 패키지 번호

► PCM DATA : PCM 음성 데이터

상향 패킷의 경우에는, 상위의 고정노드로 응답 코드, 이미지 데이터 Length, 이미지 데이터 총 사이즈, JPEG 이미지 데이터 등으로 구성된 패킷을 전달한다.

상향 패킷 (고정노드 → 서버노드 → 메인컴퓨터)

Byte 0	Byte 1	Byte 2	Byte 3
C	ERR	(FNst#_L)	(FNst#_H)

► C (Command) : 명령에 대한 응답

5 → 음성데이터 갱신 결과를 보고한다

► ERR (Error Code) : 오류 코드

0 → 정상

1 → 통신 오류(무응답)

2 → 통신 오류(패킷 오류)

► (FNst#) (Fixed Node Address) : #스테이지의 고정노드 어드레스

- **시스템 설치**

3-2절의 설치방법을 참조하여 전체 시스템의 구성도에 따라 메인컴퓨터, 서버노드, 고정노드 순으로 각 장치의 정상동작 여부를 확인하면서 시스템을 설치한다.

- **시스템 기동**

시스템 설치가 완료되면 메인컴퓨터, 서버노드, 고정노드 순으로 장치를 기동하고, 각 노드의 정상동작 여부를 확인한다.

4-2. 시스템 운용 단계

무선음성전송시스템의 설치가 완료되고 정상적으로 기동되었으면 이제 음성메시지를 녹음하고 고정노드로 전송하여 스피커로 출력하는 시스템 운용 단계로 들어간다. 사용자가 추구하는 다양한 응용분야에서 맞게 활용하게 되는 것이다.

5. 시스템 유지 및 보수

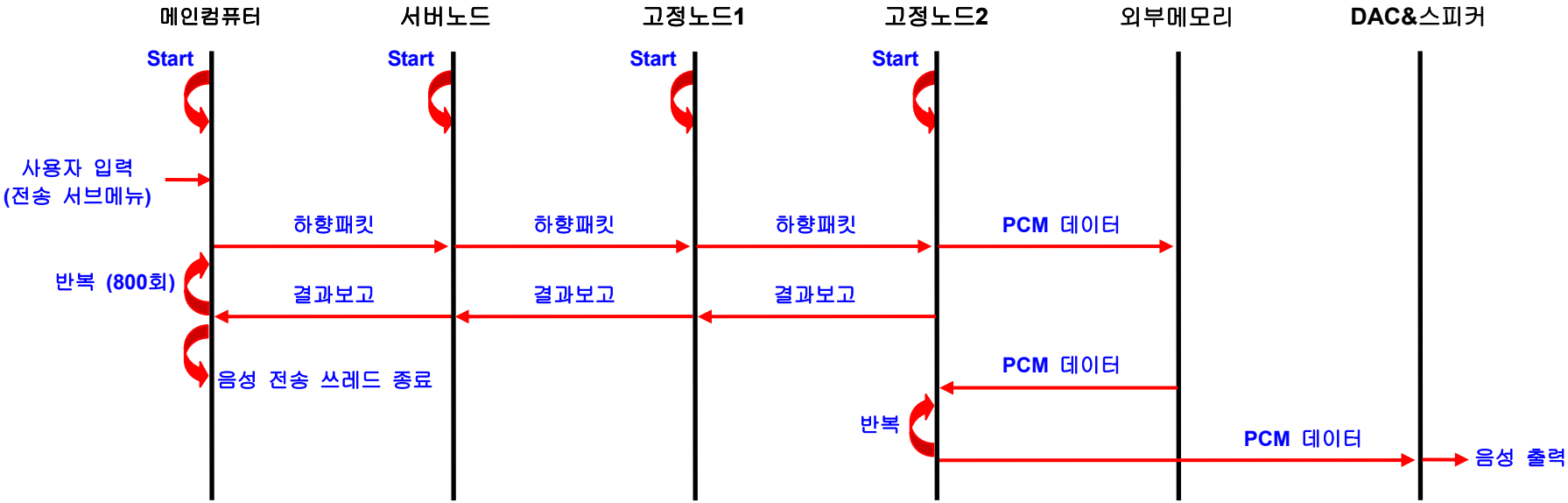
무선음성전송시스템은 운용 시간이 경과함에 따라서 시스템 자체의 노화 현상과 더불어 운용 장소의 전파 특성이 설치 당시와 달라지게 된다. 따라서 신뢰성있는 시스템을 유지하기 위하여 서버노드와 고정노드의 상태를 점검하는 등 주기적인 시스템 유지보수 절차가 필요하다.

- **서버노드 및 고정노드 점검**

운용 장소의 전파 특성 변화를 점검하고 필요하면 시스템을 재설치한다.
서버노드 및 고정노드의 설치 상태, 동작 상태, 전원 상태를 점검한다.

6. 무선 음성전송 예제 시스템의 펌웨어 프로그램 구조

무선 음성전송 예제 시스템의 프로그램은 3 종류로 제작되는데, 서버노드 프로그램, 고정노드 프로그램, 메인컴퓨터 프로그램이다. 다음은 시스템의 전체적인 동작 시나리오이다.

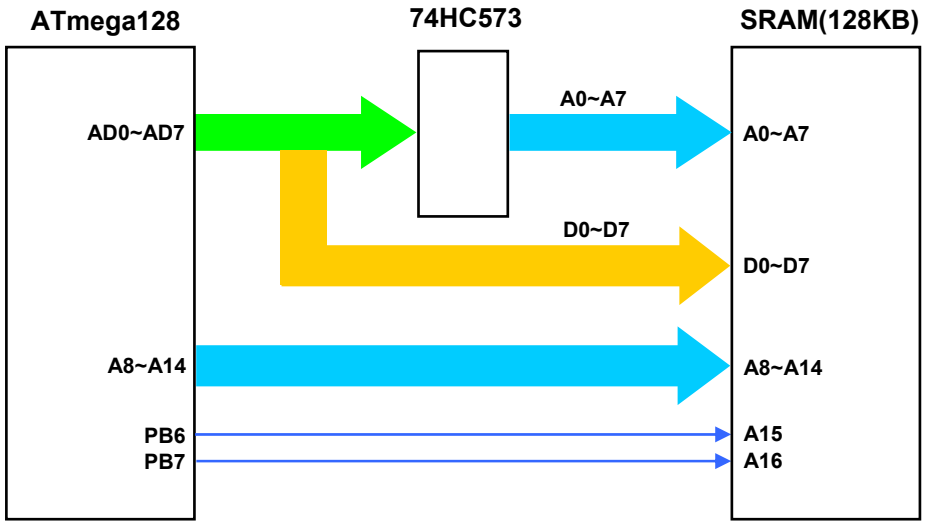


6-1. 외부메모리 제어

메인컴퓨터로부터 지정된 고정노드에 전송된 **PCM** 데이터는 해당 고정노드에 장착된 도터모듈의 외부메모리에 저장된다. **ATmega128**이 지정할 수 있는 외부메모리 주소 영역은 0x1100 ~ 0xFFFF이다. 즉, 최대 64K 바이트 용량의 외부메모리를 사용할 수 있다. 그러나 본 예제에서 필요한 버퍼 용량은 80000바이트 즉, 80K 바이트가 요구되므로, 128K 바이트 용량의 외부메모리를 제어할 수 있도록 외부메모리 회로와 펌웨어 프로그램을 제작하도록 한다.

6-1-1. 기본 회로 구성

ATmega128에서 128KB 외부메모리를 제어하기 위해 PORTB의 PB6과 PB7을 각각 A15와 A16으로 작동하도록 프로그램으로 처리할 수 있도록 하드웨어적으로 다음과 같이 구성한다.



6-1-2. 소프트웨어 처리

외부메모리 접근 영역별로 어드레스 핀의 적절한 출력을 검토하고 매크로에 적용한다.

6-1-2-1. 0x1100~0xFFFF 영역 접근

ATmega128의 A0~A15 핀으로 작동되는 외부메모리 기본영역으로서, 위의 '**기본 회로 구성**'에서와 같이, **ATmega128**의 A15 핀을 사용하지 않고 대신에 PB6을 제어하여 A15의 역할을 하도록 프로그램을 구현해야 한다.

6-1-2-2. 0x10000~0x1FFFF 영역 접근

ATmega128에서 인식되는 어드레스 값은 최대 0xFFFF이므로 0x10000~0x1FFFF 값은 0x0000~0xFFFF로 인식되어 내부메모리 영역과 외부메모리 기본영역으로 오동작을 유발하게 된다. 그래서, PB7을 A16의 역할을 하도록 추가로 할당하여 PB6과 PB7을 외부메모리에 접속하고, **ATmega128**이 0x10000~0x1FFFF 값을 0x0000~0xFFFF로 인식하여 내부메모리를 접근하는 오류를 방지하기 위하여, 프로그램이 외부메모리 영역을 접근할 때는 프로그램 내부적으로 A15를 항상 '1'의 값이 되도록하여 **ATmega128** 컨트롤러가 외부메모리 제어(AD0~A15, /RD, /WR, ALE)를 유발하도록 해놓고, 더불어서 프로그램에서 GPIO핀인 PB6과 PB7을 표준적인 A15와 A16으로 사용하면 된다.

6-1-2-3. A15, A16 출력용 GPIO 설정

예제 프로젝트 파일 중에서 **hal_zbm.h**의 '**I/O 포트 정의**' 부분과 '**I/O 포트 초기화 매크로 정의**' 부분에 PORTB의 PB6과 PB7을 A15와 A16으로 사용하기 위한 출력 포트로 설정한다.

```
//-----------------------------------------------------------------------------------------------
// Port B
//-----------------------------------------------------------------------------------------------
…
…
#define A15        6   // PB.6 – Output: to SRAM
#define A16        7   // PB.7 – Output: to SRAM

//-----------------------------------------------------------------------------------------------
// I/O 포트 초기화 매크로 정의
//-----------------------------------------------------------------------------------------------
#define PORT_INIT() \
…
…
DDRB = BM(MOSI) | BM(SCK) | BM(CSN) | BM(A15) | BM(A16); \
…
```

6-1-2-4. 외부메모리 접근 명령 매크로

앞에서 설명한 영역별 접근 방법을 통합하여 적용하면 다음과 같은 매크로 명령어를 정의할 수 있다. 펌웨어 프로그램에서 이와 같은 매크로를 정의하면 외부메모리 공간을 편리하게 사용할 수 있다.

- **외부메모리 쓰기 명령 매크로**

```
#define wemem(elogicaladdr,data) \
        do { \
                PORTB &= 0x3F; PORTB |= (((elogicaladdr) >> 9) & 0xC0); \
                *((unsigned char *)((unsigned short)((elogicaladdr & 0xFFFF) | 0x8000))) = (data); \
```

```
} while (0)
```

- 외부메모리 읽기 명령 매크로

```
#define remem(elogicaladdr,data) \
        do { \
                PORTB &= 0x3F; PORTB |= (((elogicaladdr) >> 9) & 0xC0); \
                data = *((unsigned char *)((unsigned short)((elogicaladdr & 0xFFFF) | 0x8000))); \
} while (0)
```

- 외부메모리 쓰기 명령어

```
wemem(elogicaladdr,wdata);
```

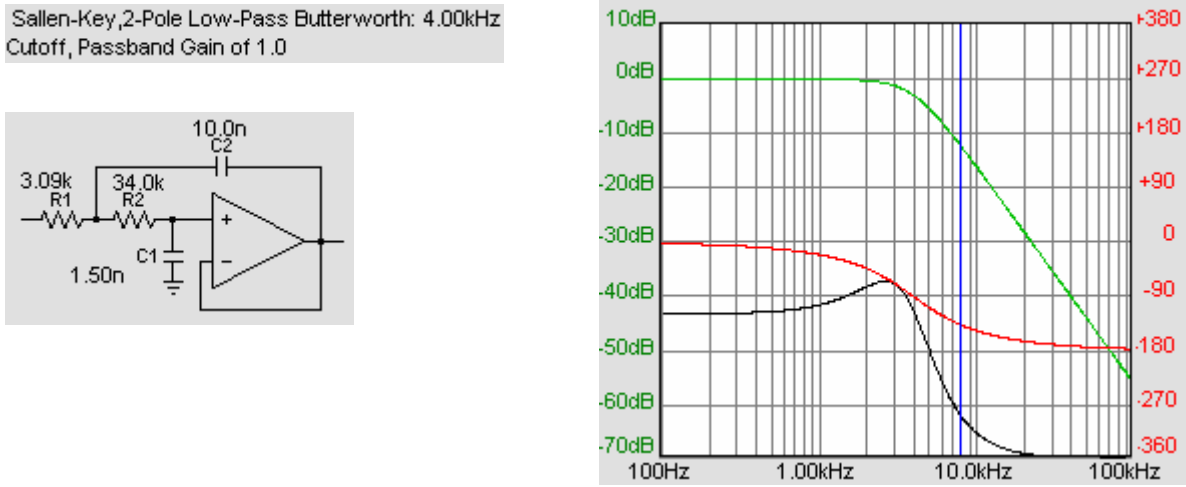
- 외부메모리 읽기 명령어

```
remem(elogicaladdr,rdata);
```

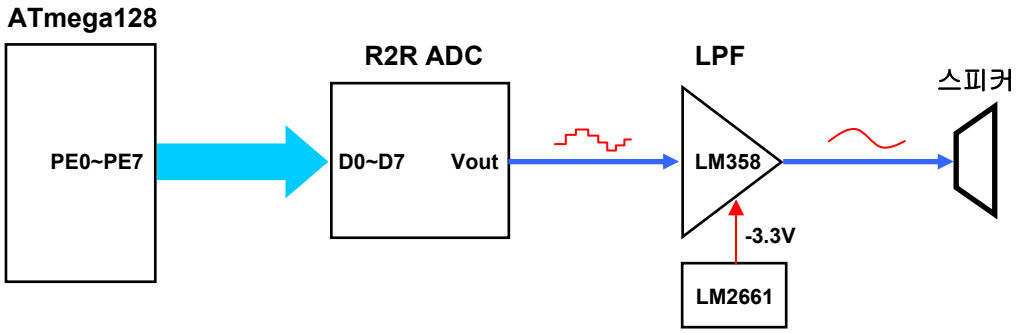
6-2. DAC & 스피커 제어

PCM 음성 데이터를 재생하려면 기본적으로 **DAC**(Digital to Analog Converter)를 사용해야 한다. 다양한 **DAC** 방법이 있지만 본 예제에서는 간단하면서도 실용적으로 음성을 재생할 수 있는 **R2R** 방식의 **DAC** 회로를 구현하도록 한다. **R2R DAC**는 순전히 저항(Resistor)으로만 구성되어서, R옴과 2R옴의 저항을 사다리 형태로 구성하여 8비트의 디지털 신호를 전압으로 변환한다.
DAC 회로를 거친 음성신호 전압은 불연속적인 신호로서, 불연속 부분에서 고주파 하모닉 노이즈가 발생하게 된다. 따라서 노이즈를 제거하기 위해 LPF(Low Pass Filter) 회로가 추가되어야 한다. 본 예제에서는 **LM358** OP앰프를 사용하여 2차 Active LPF를 구현한다. **LM358**을 3.3V 단전원으로 구동하기에는 출력 스윙이 너무 작다. 그래서 **LM2661** Voltage Inverter를 사용하여 3.3V로부터 -3.3V를 만들어내어 **LM358**에 +3.3V와 -3.3V를 공급하는 방식으로 제작한다.

LPF는 **TI**에서 공개하는 **FilterPro V2.0**로 설계했으면 제원과 특성은 다음과 같다.



6-2-1. 기본 회로 구성



6-2-2. 소프트웨어 처리

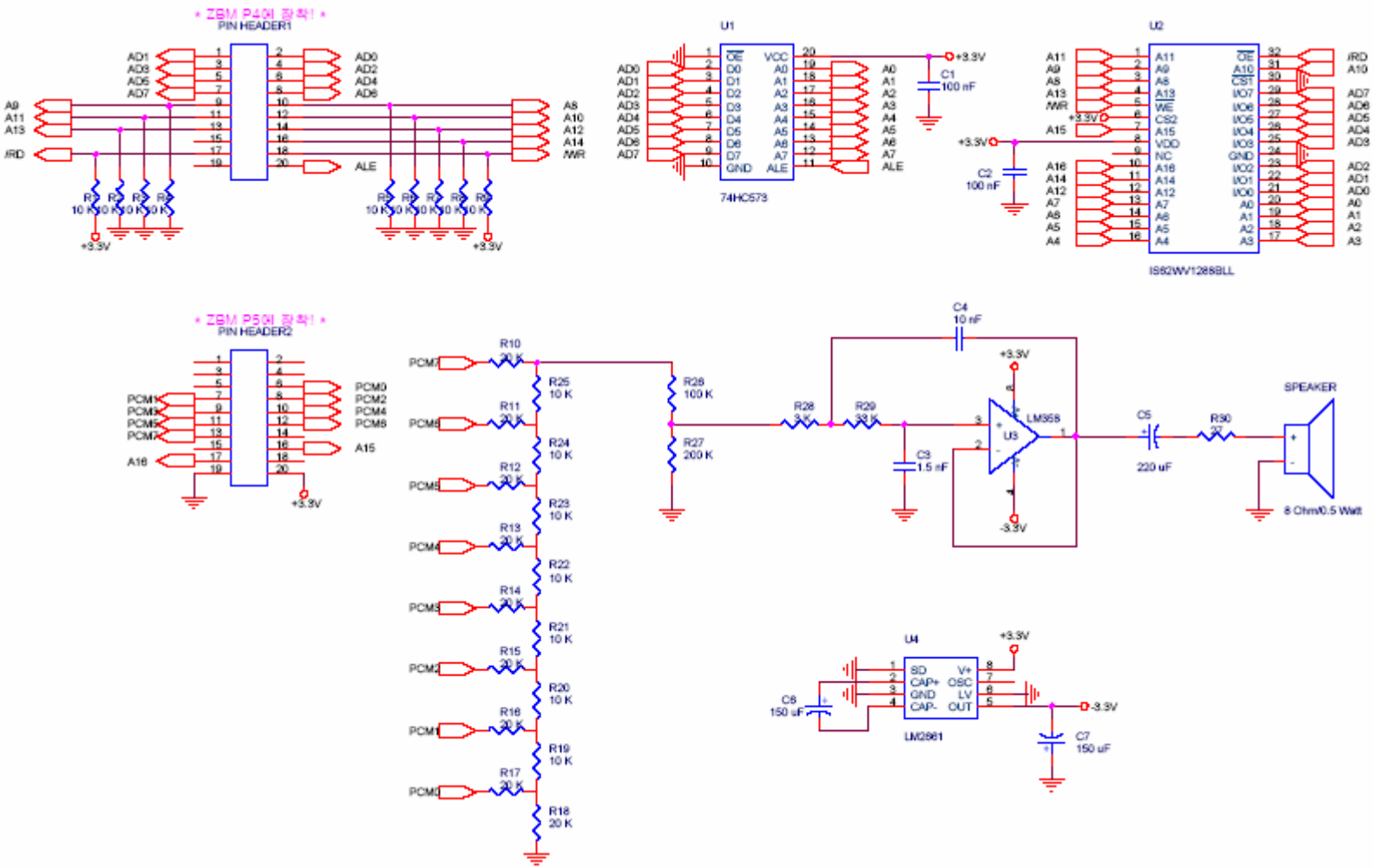
예제 프로젝트 파일 중에서 **hal_zbm.h**의 '**I/O 포트 정의**' 부분과 '**I/O 포트 초기화 매크로 정의**' 부분에 PORTE를 **PCM** 데이터 출력 포트로 설정한다.

```
//-----------------------------------------------------------------------------------------------
// Port E
//-----------------------------------------------------------------------------------------------
#define PCM0    0   // PE.0 – Output: to R2R DAC
#define PCM1    1   // PE.1 – Output: to R2R DAC
#define PCM2    2   // PE.2 – Output: to R2R DAC
#define PCM3    3   // PE.3 – Output: to R2R DAC
#define PCM4    4   // PE.4 – Output: to R2R DAC
#define PCM5    5   // PE.5 – Output: to R2R DAC
#define PCM6    6   // PE.6 – Output: to R2R DAC
#define PCM7    7   // PE.7 – Output: to R2R DAC

//-----------------------------------------------------------------------------------------------
// I/O 포트 초기화 매크로 정의
//-----------------------------------------------------------------------------------------------
#define PORT_INIT() \
…
…
DDRE = 0xFF; \
…
```

6-3.도터모듈 제작

도터모듈은 **ZBM v1.2**의 확장커넥터에 장착되는 형태로 제작한다.

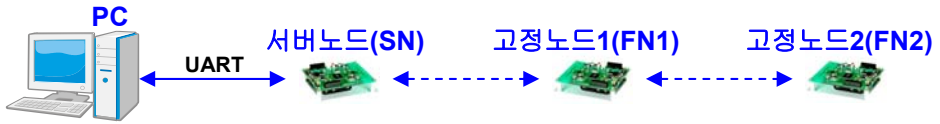
ZBM v1.2의 내부 회로는 3.3V 전원으로 동작되므로 메모리와 기타 부품은 3.3V 전원 타입으로 선정한다.



[회로도 : 당사 홈페이지 Downloads 메뉴에 **DTBD.pdf** 참조]

6-4. 라우팅 동작

메인컴퓨터로부터 최종단의 고정노드에 이르기까지 데이터가 전달되기 위해서는 라우팅 정보가 필요하다. 라우팅 방법과 정보는 여러가지가 있겠지만 본 예제에서는 메인컴퓨터에서 라우팅테이블을 관리하고 정보패킷에 라우팅정보를 실어서 하위노드로 내려 보내는 방식을 사용한다. 라우팅테이블을 제작하기 위해서는 다음과 같이 시스템구성도에 기초한 라우팅정보가 필요하다.



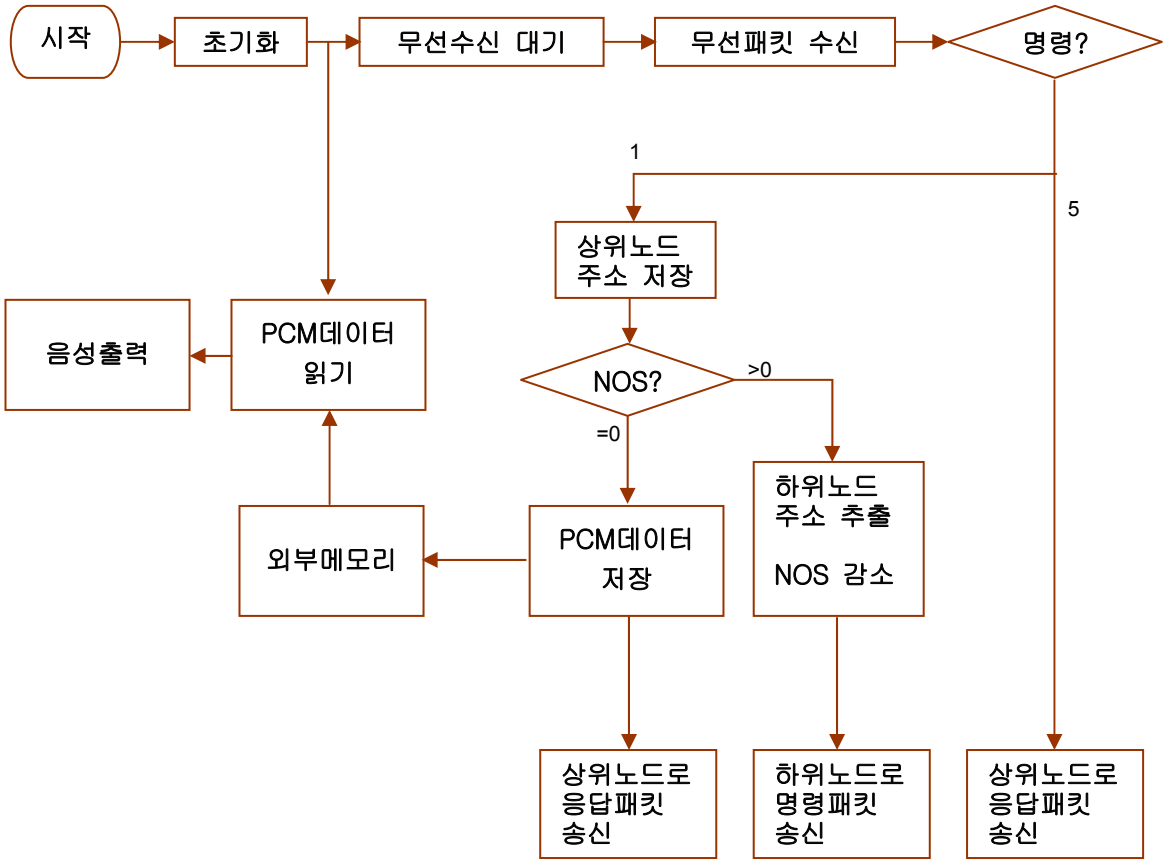
[예제 프로젝트의 네트워크 구성도]

Destination	No. of stages	Stage 1	Stage 2
0x01	1	0x01	
0x02	2	0x01	0x02

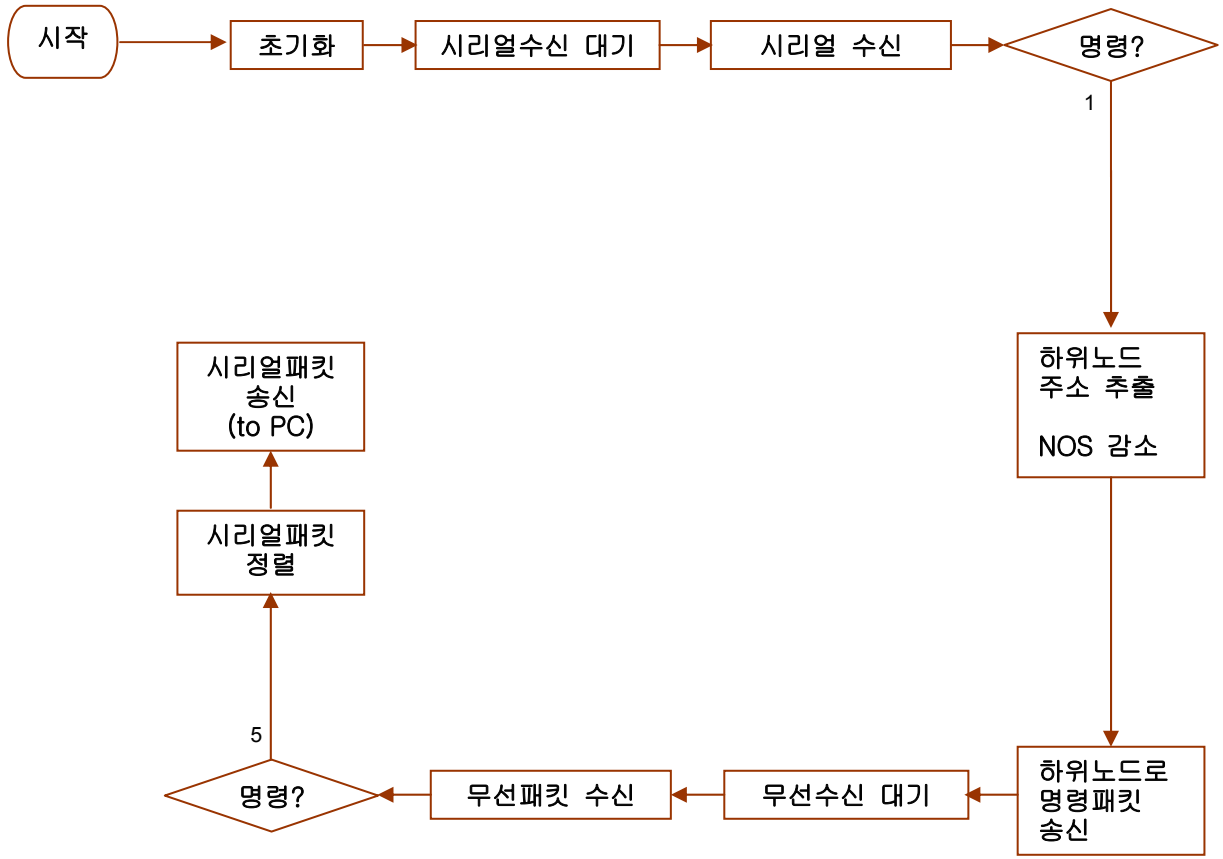
[라우팅테이블]

6-5. 노드별 펌웨어 프로그램 순서도

6-5-1. 고정노드 프로그램



6-5-2. 서버노드 프로그램



7. 노드별 펌웨어 및 메인컴퓨터 응용프로그램 제작

6절에서 설명한 내용을 바탕으로, 각 노드의 펌웨어 프로그램과 메인컴퓨터를 위한 응용프로그램을 제작해보기로 한다.
각 노드의 펌웨어를 제작하기 위해, **ZBM v1.2** 응용을 위한 범용 프로그램 소스인 **PROTO**를 활용한다.
메인컴퓨터 응용프로그램은 **Microsoft Visual C++6.0**을 사용하여 구현하도록 한다.

7-1. 고정노드 프로그램 제작

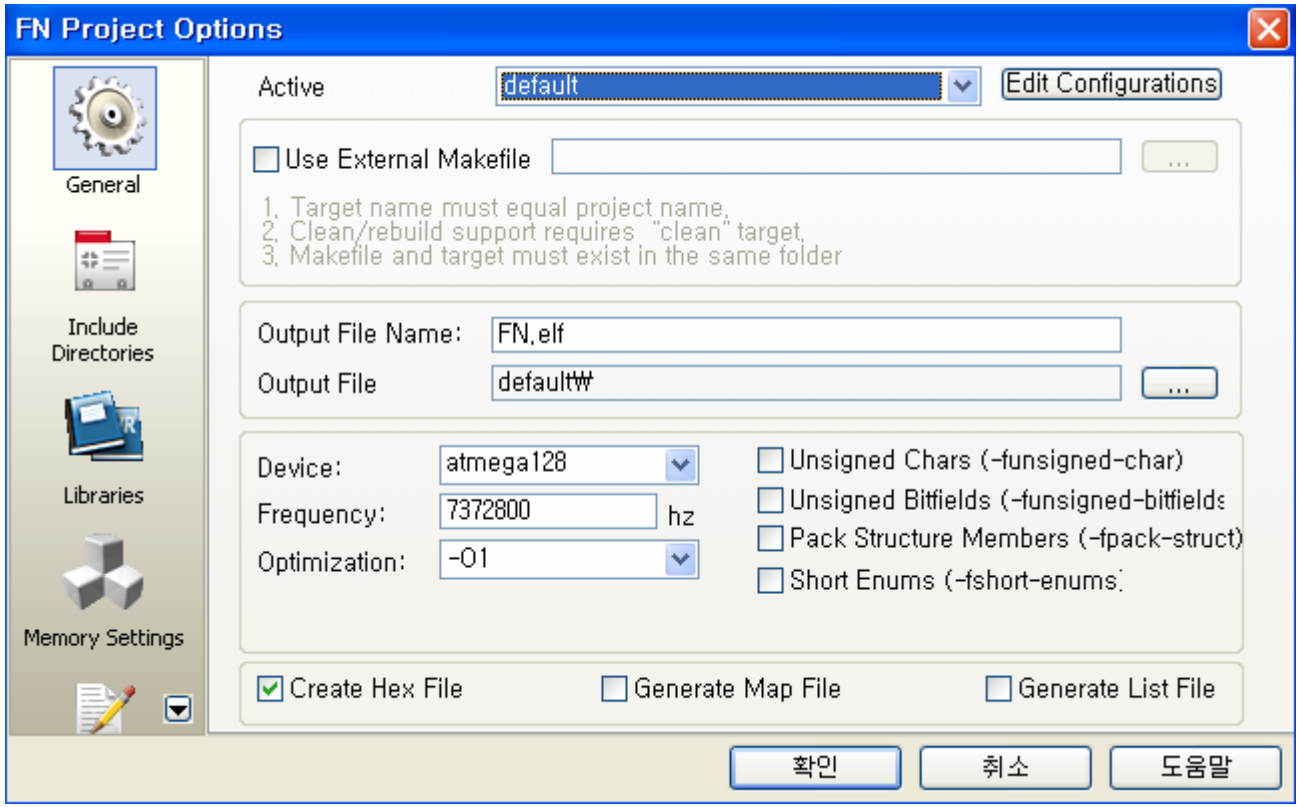
[Step.1] 고정노드 프로젝트 생성

① 프로젝트를 저장할 디렉토리를 하나 만들고(예, **FN**) **AVR Studio 4**를 실행한 후 **Project** 메뉴에 있는 **New Project**를 클릭한다.
(※ 디렉토리와 프로젝트 이름에는 한글을 사용하지 않도록 한다.)

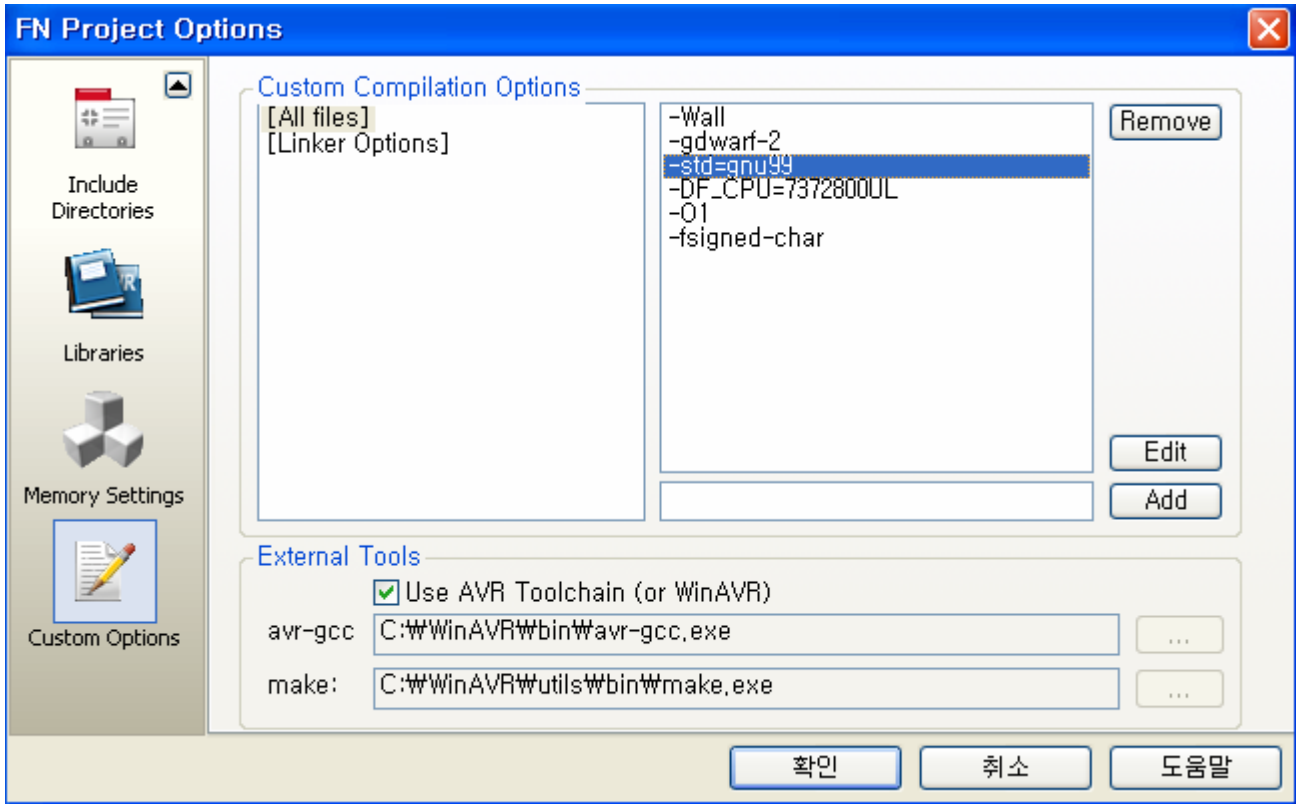
② **Project type**에서 **AVR GCC**를 선택하고, **Project name**에 적당한 이름을 넣고(예, **FN**), **Create initial file**을 체크 해제하고, **Location**에 앞에서 만든 프로젝트 디렉토리를 지정해주고, **Next**를 클릭한다.

③ **Debug platform**을 **JTAG ICE**로 선택하고, **Device**를 **ATmega128**로 선택하고 **Finish**를 클릭한다. 그러면 새로운 프로젝트가 생성된다.

④ **Project** 메뉴에 있는 **Configuration Options**를 선택하여 열고, 좌측의 아이콘 창에서 **General**을 선택하여 열고 아래 화면과 같이 입력 및 체크한다.



아이콘 창을 아래로 스크롤하여 **Custom Options**를 선택하여 열고 **-std=gnu99**가 등록되어 있지 않으면 입력하고 **ADD**를 클릭하여 등록한 후 **확인**을 클릭한다.



⑤ 프로젝트 디렉토리 아래에, **PROTO** 디렉토리에 있는 3개의 디렉토리를 복사해 넣는다.(**apps**, **include**, **lib**)

⑥ **Project** 메뉴의 **Configuration Options**를 다시 선택하여 열고, 좌측의 아이콘 창에서 **Include Directories** 를 선택하여 열고, 프로젝트 디렉토리를 선택하여 등록한다.

⑦ 화면 좌측의 프로젝트매니저 창에서 **Source Files**를 선택하고, 마우스 우측 버튼을 클릭하고, **Add Existing File(s)...**를 클릭한 후 **lib** 디렉토리 및 그 하위의 디렉토리에 있는 모든 **.c** 파일을 선택하여 등록한다.

⑧ 화면 좌측의 프로젝트매니저 창에서 **Source Files**를 선택하고, 마우스 우측 버튼을 클릭하고, **Add Existing File(s)...**를 클릭한 후 **apps** 디렉토리에 있는 **master.c** 파일을 선택하여 등록한 후 **Rename File...**하여 고정노드1(FN1)을 위해 파일명을 바꾼다.(예, **fn1.c**)

⑨ **Build** 메뉴에서 **Rebuild All**을 클릭하여 컴파일 및 **hex**파일을 생성한다. (※ 오류가 발생하지 않아야 한다.)

[Step.2] hal_zbm.h 수정

hal_zbm.h는 **ATmega128(A,L)**의 I/O핀과 외부장치와의 핀 연결에 관한 초기화 매크로나 간단한 명령 매크로를 정의한다.
외부장치는 외부메모리, LCD, 각종 센서등, **ZBM v1.2**의 확장커넥터와 접속되는 모든 장치를 말한다.
따라서 사용자는 외부확장 커넥터(P4, P5)를 통해서 응용보드를 제어하려면 해당 I/O핀을 **hal_zbm.h**에 등록하고 초기화할 수 있도록 해야한다.

본 예제에서는 **6-1** 절과 **6-2** 절에서 설명한 바와 같이 외부 도터모듈과의 접속을 위한 사항을 적용한다.

① 프로젝트매니저 창에서 **External Dependancies**를 열고 **hal_zbm.h**를 더블클릭하여 편집창에 띄운다.

② **I/O 포트 정의** 부분에 사용하고자 하는 핀을 추가로 정의한다.

```
//-------------------------------------------------------------------------------------------------
// Port B
//-------------------------------------------------------------------------------------------------
#define CSN                 0   // PB.0 - Output: SPI Chip Select (CS_N)
#define SCK                 1   // PB.1 - Output: SPI Serial Clock (SCLK)
#define MOSI                2   // PB.2 - Output: SPI Master out - slave in (MOSI)
#define MISO                3   // PB.3 - Input:   SPI Master in - slave out (MISO)
//#define                   4   // PB.4 - 사용가능
//#define                   5   // PB.5 - 사용가능
#define A15                 6   // PB.6 - Output: to SRAM
#define A16                 7   // PB.7 - Output: to SRAM

…

//-------------------------------------------------------------------------------------------------
// Port E
//-------------------------------------------------------------------------------------------------
#define PCM0                0   // PE.0 - Output: to R2R DAC
#define PCM1                1   // PE.1 - Output: to R2R DAC
#define PCM2                2   // PE.2 - Output: to R2R DAC
#define PCM3                3   // PE.3 - Output: to R2R DAC
#define PCM4                4   // PE.4 - Output: to R2R DAC
#define PCM5                5   // PE.5 - Output: to R2R DAC
#define PCM6                6   // PE.6 - Output: to R2R DAC
#define PCM7                7   // PE.7 - Output: to R2R DAC

…
```

③ **초기화 매크로 정의** 부분의 **I/O 포트 초기화 매크로 정의** 부분에 위에서 추가로 정의한 핀을 초기화한다.

```
#define PORT_INIT() \
        do { \
                SFIOR |= BM(PUD); \
                DDRB   = BM(MOSI) | BM(SCK) | BM(CSN) | BM(A15) | BM(A16); \
                PORTB = BM(MOSI) | BM(SCK) | BM(CSN); \
                DDRD   = BM(UART1_RTS); \
                PORTD = BM(UART1_RTS) | BM(UART1_CTS); \
                DDRE   = 0xFF; \
                DDRG   = BM(RESET_N) | BM(VREG_EN); \
                PORTG = BM(RESET_N); \
        } while (0)
```

④ LCD모듈 등 사용자가 장착하는 응용보드에 대한 초기화나 명령 매크로가 필요한 경우 **사용자 추가 매크로** 부분에 추가한다.

외부메모리 쓰기 및 읽기 명령 매크로를 추가한다.

```
//-------------------------------------------------------------------------------------------------
// 외부메모리 쓰기 명령
//-------------------------------------------------------------------------------------------------
#define wemem(elogicaladdr,data) \
        do { \
                unsigned long mappingaddr = elogicaladdr + 0x1100; \
                PORTB &= 0x3F; PORTB |= (((mappingaddr) >> 9) & 0xC0); \
                if(mappingaddr > 0xFFFF) \
                        *((unsigned char *)((unsigned short)(( mappingaddr & 0xFFFF) | 0x8000))) = (data); \
                else if(mappingaddr <= 0xFFFF) \
                        *((unsigned char *)((unsigned short)( mappingaddr))) = (data); \
} while (0)

//-------------------------------------------------------------------------------------------------
// 외부메모리 읽기 명령
//-------------------------------------------------------------------------------------------------
#define remem(elogicaladdr,data) \
        do { \
```

```
                unsigned long mappingaddr = elogicaladdr + 0x1100; \
                PORTB &= 0x3F; PORTB |= (((mappingaddr) >> 9) & 0xC0); \
                if(mappingaddr > 0xFFFF) \
                        data = *((unsigned char *)((unsigned short)(( mappingaddr & 0xFFFF) | 0x8000))); \
                else if(mappingaddr <= 0xFFFF) \
                        data = *((unsigned char *)((unsigned short)( mappingaddr))); \
} while (0)
```

[Step.3] hal.h 수정

hal.h는 **ATmega128(A,L)**의 내부 리소스 사용에 관한 초기화 매크로나 간단한 명령 매크로를 정의한다.
현재 기본적으로 정의되어 있는 매크로는 다음과 같다. **(hal.h** 참조)

- SPI_INIT() 매크로
- SPI를 통한 **CC2420** 제어 매크로
- 1 usec 딜레이 함수 선언 (halWait)
- Global interrupt 제어 매크로 (sei, cli)
- UART0, UART1 초기화 및 사용과 관련된 매크로
- ADC 초기화 및 사용과 관련된 매크로

SPI는 **CC2420** 제어 전용으로 사용되므로 사용자가 SPI를 추가적으로 사용할 경우에는 세심한 주의를 요한다.
위에서 SPI 관련 항목을 제외하고 4가지 항목이 정의되어 있으므로 사용자는 내용을 충분히 파악해서 프로그램 개발에 활용하도록 한다.
위에서 언급되지 않은 항목을 사용하려면 **hal.h**에 관련된 매크로를 추가해준다. 예를 들어 다음과 같은 내부 리소스를 사용하려면 사용자가 초기화 및 사용과 관련된 매크로를 추가해 주어야 한다.

- 타이머/카운터

- 외부인터럽트
- 내부EEPROM

① 프로젝트매니저 창에서 **External Dependancies**를 열고 **hal.h**를 더블클릭하여 편집창에 띄운다.

② 기존에 등록되어 있지 않은 **ATmega128(A,L)** 내부 리소스를 사용하려면 관련된 초기화 및 명령 매크로를 **사용자 추가 매크로** 부분에 추가한다.

※ 본 예제의 고정노드 프로그램에서는 타이머/카운터2를 사용하지만 응용프로그램 소스에서 바로 정의하여 사용하기로 한다.

[Step.4] fn1.c 작성

① **fn1.c**를 더블클릭하여 편집창에 띄우고 고정노드에 대한 소스를 작성한다. (소스에 표시된 (1) ~ (9) 순)

② 작성이 완료되면 **Build** 메뉴에서 **Rebuild All**을 클릭하여 컴파일 및 hex파일을 생성한다.
(※ 오류가 발생하면 디버깅한다.)

◆ **fn1.c**의 내용 (청색 글씨의 내용만 필요에 따라 수정) ◆

```
/*****************************************************************************************************
*           - PAN ID: 0x2176 for example                                                             *
*           - Short address: 0x0000 ~ 0xFFFF changeable                                              *
*****************************************************************************************************/

#include "include\include.h"

//-----------------------------------------------------------------------------------------------
```

```
// Basic RF transmission and reception structures
//------------------------------------------------------------------------------------------------
BASIC_RF_RX_INFO rfRxInfo;
BASIC_RF_TX_INFO rfTxInfo;
BYTE pTxBuffer[BASIC_RF_MAX_PAYLOAD_SIZE];
BYTE pRxBuffer[BASIC_RF_MAX_PAYLOAD_SIZE];

//------------------------------------------------------------------------------------------------
// (7) 사용자가 필요에 따라 만든 외부변수
//------------------------------------------------------------------------------------------------
//
// 사용자 외부변수 추가…
//
volatile unsigned int Upperaddr, Loweraddr;

//------------------------------------------------------------------------------------------------
// (8) 사용자가 필요에 따라 만든 함수는 여기에 정의하거나 별도의 .c 파일에 기술한다.
//------------------------------------------------------------------------------------------------
//
// 사용자 함수 추가…
//

//------------------------------------------------------------------------------------------------
// 상위노드 혹은 하위노드로부터의 무선데이타 수신후 처리 함수(인터럽트 핸들러)
//------------------------------------------------------------------------------------------------
BASIC_RF_RX_INFO* basicRfReceivePacket(BASIC_RF_RX_INFO *pRRI)
{
//------------------------------------------------------------------------------------------------
// (9) pRRI->pPayload[] : 상위노드 혹은 하위노드로부터 수신한 데이타 처리
//------------------------------------------------------------------------------------------------
        unsigned char Command, NOS, i;
        unsigned int Fromaddr, NOP;
        unsigned long INDEX;

        // 패킷을 보낸 고정노드의 주소
```

```
Fromaddr = pRRI->srcAddr;

// 명령 코드
Command = pRRI->pPayload[0];

// 명령 분석
switch(Command)
{
        case 1: // 음성데이터를 갱신하라 : 상위노드로부터
        {
                Upperaddr = Fromaddr;
                NOS = pRRI->pPayload[1];

                if(NOS == 0) // 여기가 목적지 노드! 음성데이터 갱신 작업..
                {
                        NOP = ((unsigned int)pRRI->pPayload[2]) | ((unsigned int)(pRRI->pPayload[3])<<8);

                        INDEX = (unsigned long)NOP * 100;

                        // 외부메모리에 저장
                        for(unsigned long eaddr = INDEX; eaddr < INDEX + 100; eaddr++)
                        {
                                wemem(eaddr,pRRI->pPayload[eaddr%100+4]);
                        }

                        // 상위노드에 응답할 패킷 구성
                        pTxBuffer[0] = 5; // 음성데이터 갱신 결과를 보고한다
                        pTxBuffer[1] = 0; // 오류코드 : 정상!
                        pTxBuffer[2] = rfSettings.myAddr & 0xFF; // 현재노드 주소 하위바이트
                        pTxBuffer[3] = ((rfSettings.myAddr & 0xFFFF) >> 8) & 0xFF; // 현재노드 주소 상위바이트

                        // 패킷 포맷(C, ERR, FNst#_L,FNst#_H)에 따라 4 바이트 길이
                        rfTxInfo.length = 4;

                        // 패킷을 수신할 상위노드 주소
```

```
		rfTxInfo.destAddr = Upperaddr;

		// 발사!!
		basicRfSendPacket(&rfTxInfo);
	}
	else if(NOS > 0) // 여기는 중간노드! 하위노드로 메시지 전달..
	{
		NOS--; // NOS 갱신

		Loweraddr = (unsigned int)(pRRI->pPayload[2]); // 하위 고정노드 주소 하위바이트
		Loweraddr |= ((unsigned int)(pRRI->pPayload[3]))<<8; // 하위 고정노드 주소 상위바이트

		// 하위 고정노드에 송신할 패킷 구성
		pTxBuffer[0] = Command; // 명령
		pTxBuffer[1] = NOS;

		for(i = 0 ; i < NOS*2 ; i += 2) // 하위 고정노드 주소들
		{
			pTxBuffer[2+i] = pRRI->pPayload[4+i];
			pTxBuffer[3+i] = pRRI->pPayload[5+i];
		}

		// 나머지 바이트 무선 송신 버퍼에 저장
		for(i=NOS*2; i<NOS*2+102; i++)
		{
			pTxBuffer[2+i] = pRRI->pPayload[4+i];
		}

		// 패킷 포맷에 따라 NOS*2 + 104 바이트 길이
		rfTxInfo.length = NOS*2 + 104;

		// 패킷을 수신할 하위 고정노드 주소
		rfTxInfo.destAddr = Loweraddr;

		// 발사!!
```

```
                basicRfSendPacket(&rfTxInfo);
        }

        break; // switch(Command)문 탈출

} // case 1

case 5: // 음성데이터 갱신 결과를 보고한다 : 하위노드로부터
{
        // 상위 노드에 송신할 패킷 구성
        pTxBuffer[0] = 5; // 음성데이터 갱신 결과를 보고한다
        pTxBuffer[1] = pRRI->pPayload[1]; // 오류코드
        pTxBuffer[2] = pRRI->pPayload[2]; // 고정노드 주소 하위바이트
        pTxBuffer[3] = pRRI->pPayload[3]; // 고정노드 주소 상위바이트

        // 패킷 포맷(C, ERR, FNst#_L,FNst#_H)에 따라 4 바이트 길이
        rfTxInfo.length = 4;

        // 패킷을 수신할 상위노드 주소
        rfTxInfo.destAddr = Upperaddr;

        // 발사!!
        basicRfSendPacket(&rfTxInfo);

        break; // switch(Command)문 탈출

} // case 5

default: // 수신 패킷 오류
{
        // 하위노드로부터 수신한 메시지이면..
        if(Fromaddr == Loweraddr)
        {
                rfTxInfo.destAddr = Upperaddr; // 상위노드에 보고
```

```
                        }
                        else rfTxInfo.destAddr = Fromaddr; // 상위노드로부터 수신한 메시지, 상위노드에 보고

                        // 상위노드에 송신할 패킷 구성
                        pTxBuffer[0] = 5; // 음성데이터 갱신 결과를 보고한다
                        pTxBuffer[1] = 2; // 오류코드 : 통신 오류 (패킷 오류)
                        pTxBuffer[2] = rfSettings.myAddr & 0xFF; // 현재노드 주소 하위바이트
                        pTxBuffer[3] = ((rfSettings.myAddr & 0xFFFF) >> 8) & 0xFF; // 현재노드 주소 상위바이트

                        // 패킷 포맷에 따라 4 바이트 길이
                        rfTxInfo.length = 4;

                        // 발사!!
                        basicRfSendPacket(&rfTxInfo);

                        break; // switch(Command)문 탈출
                }

        } // switch(Command)

        return pRRI; // 수신대기 상태로 복귀

} // basicRfReceivePacket

//-------------------------------------------------------------------------------------------------
// main 함수
//-------------------------------------------------------------------------------------------------
int main(void)
{
//-------------------------------------------------------------------------------------------------
// (1) I/O포트, 변수, ATmega128 내부장치 및 주변장치 초기화
//-------------------------------------------------------------------------------------------------
        unsigned int srcaddr;
        unsigned char x;
```

```
        PORT_INIT();
        SPI_INIT();
        ENABLE_EXT_RAM(); // 외부메모리 사용 매크로, MCUCR |= 0x80;
//      INIT_UART1(UART_BAUDRATE_115K2,UART_OPT_8_BITS_PER_CHAR); // UART1 초기화(115.2 Kpbs)
//      ENABLE_UART1();

//-------------------------------------------------------------------------------------------------
// (2) source address 설정
//-------------------------------------------------------------------------------------------------
        srcaddr = 0x0001; // 라우팅 정보에 따라 고정노드1의 어드레스를 0x0001로 설정

//-------------------------------------------------------------------------------------------------
// (3) 채널 및 Pan ID 설정
//-------------------------------------------------------------------------------------------------
        basicRfInit(&rfRxInfo, 26, 0x2176, srcaddr); // 26번 채널 설정, Pan ID = 0x2176(예를 들어..)

//-------------------------------------------------------------------------------------------------
// (4) 전송데이타 크기 설정
//-------------------------------------------------------------------------------------------------
        rfTxInfo.length = 104; // 104 바이트 전송(예를 들어..) C, NOS, NOP_L, NOP_H, PCM DATA

//-------------------------------------------------------------------------------------------------
// ACK 수신 여부 설정
//-------------------------------------------------------------------------------------------------
        rfTxInfo.ackRequest = TRUE;

        rfTxInfo.pPayload = pTxBuffer;
        rfRxInfo.pPayload = pRxBuffer;

//-------------------------------------------------------------------------------------------------
// (5) 전송데이타 초기화
//-------------------------------------------------------------------------------------------------
        for (unsigned char n = 0; n < 104; n++)
        {
                pTxBuffer[n] = 0; // 송신 버퍼 클리어
```

```
        }

        // Turn on RX mode
        basicRfReceiveOn();

//-----------------------------------------------------------------------------------------------
// (6) 메인 루틴
//-----------------------------------------------------------------------------------------------
        while(1) // 무한 반복
        {
                for(unsigned long eaddr = 0; eaddr < 80000; eaddr++)
                {
                        remem(eaddr,x); // 외부메모리에서 읽어낸다
                        PORTE = x;      // R2R DAC로 출력한다
                        halWait(100);   // PCM 샘플링 주기 조절 (대략 125 uSec)
                }

        } // while(1)

} // main()
```

[Step.5] fn2.c 작성

① **apps** 디렉토리에서 **fn1.c**를 복하사여 **fn2.c**를 만든다.

② 화면 좌측의 프로젝트매니저 창에서 **Source Files**를 열고, **fn1.c**에 마우스 우측 버턴을 클릭하고, **Remove File from Project**를 클릭하여 현재 프로젝트에서 제외한다.

③ 프로젝트매니저 창에서 **Source Files**를 선택하고, 마우스 우측 버턴을 클릭하고, **Add Existing File(s)...**를 클릭한 후 **apps** 디렉토리에 있는 **fn2.c** 파일을 선택하여 등록한다.

④ **fn2.c** 파일을 더블클릭하여 편집창에 띄우고 고정노드2에 대한 소스 어드레스 부분을 다음과 같이 수정한다.

```
//-------------------------------------------------------------------------------------------------
// (2) source address 설정
//-------------------------------------------------------------------------------------------------
        srcaddr = 0x0002; // 라우팅 정보에 따라 고정노드2의 어드레스를 0x0002로 설정
```

7-2. 서버노드 프로그램 제작

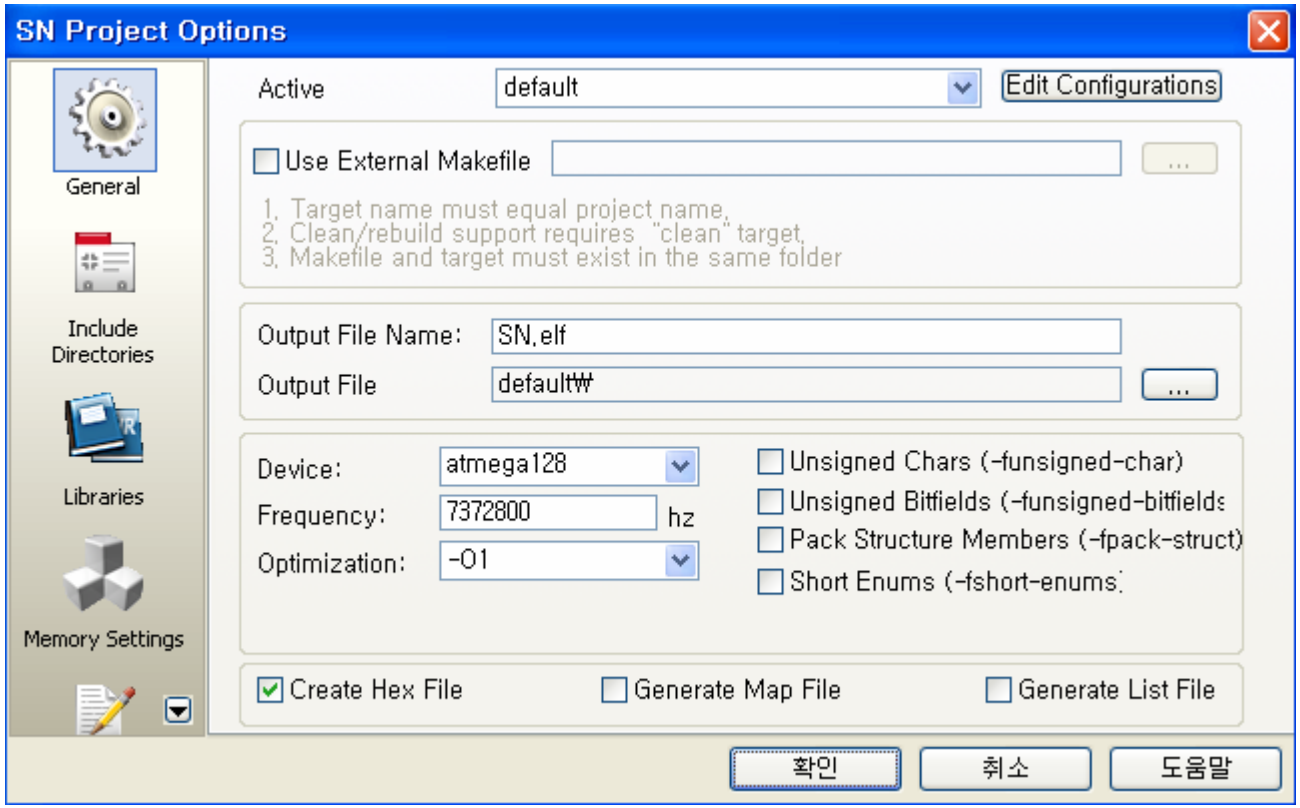
[Step.1] 서버노드 프로젝트 생성

① 프로젝트를 저장할 디렉토리를 하나 만들고(예, **SN**) **AVR Studio 4**를 실행한 후 **Project** 메뉴에 있는 **New Project**를 클릭한다.
(※ 디렉토리와 프로젝트 이름에는 한글을 사용하지 않도록 한다.)

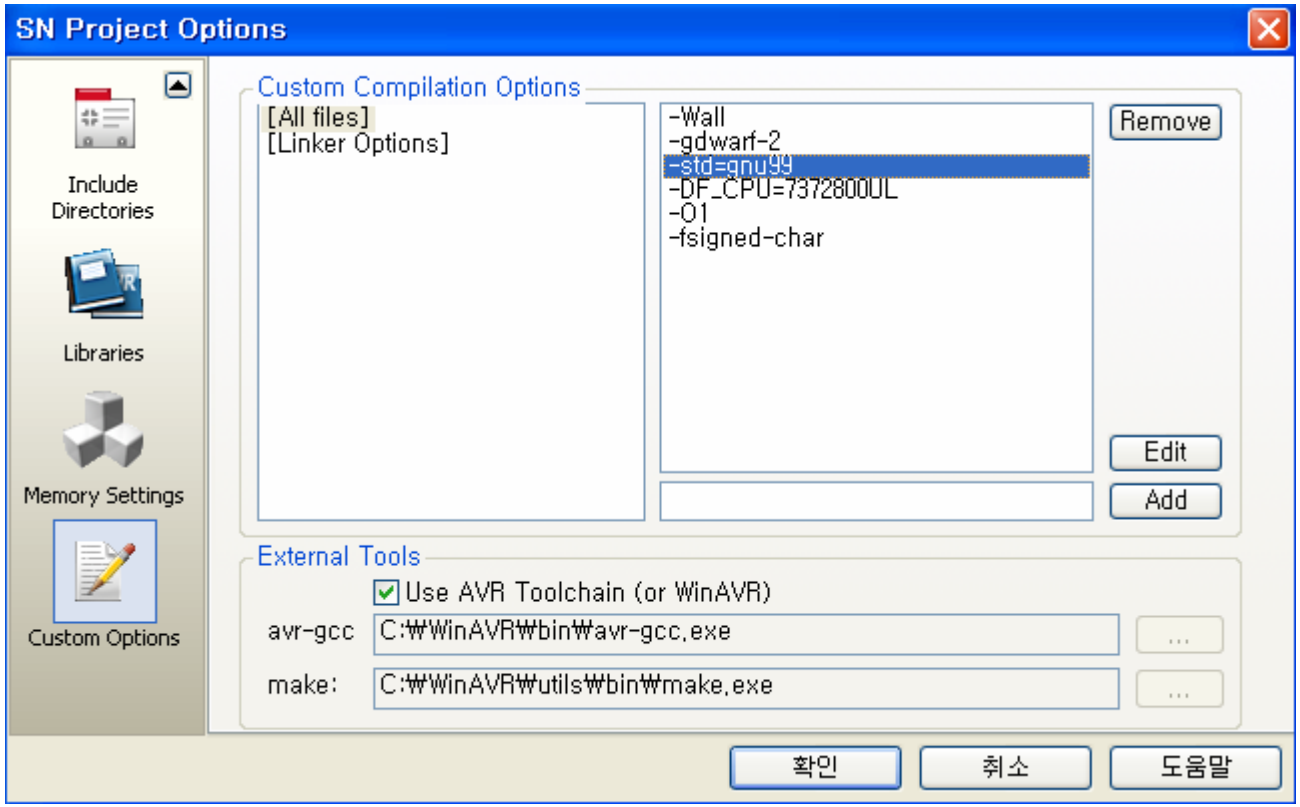
② **Project type**에서 **AVR GCC**를 선택하고, **Project name**에 적당한 이름을 넣고(예, **SN**), **Create initial file**을 체크 해제하고, **Location**에 앞에서 만든 프로젝트 디렉토리를 지정해주고, **Next**를 클릭한다.

③ **Debug platform**을 **JTAG ICE**로 선택하고, **Device**를 **ATmega128**로 선택하고 **Finish**를 클릭한다. 그러면 새로운 프로젝트가 생성된다.

④ **Project** 메뉴에 있는 **Configuration Options**를 선택하여 열고, 좌측의 아이콘 창에서 **General**을 선택하여 열고 아래 화면과 같이 입력 및 체크한다.



아이콘 창을 아래로 스크롤하여 **Custom Options**를 선택하여 열고 **-std=gnu99**가 등록되어 있지 않으면 입력하고 **ADD**를 클릭하여 등록한 후 **확인**을 클릭한다.



⑤ 프로젝트 디렉토리 아래에, **PROTO** 디렉토리에 있는 3개의 디렉토리를 복사해 넣는다.(**apps**, **include**, **lib**)

⑥ **Project** 메뉴의 **Configuration Options**를 다시 선택하여 열고, 좌측의 아이콘 창에서 **Include Directories** 를 선택하여 열고, 프로젝트 디렉토리를 선택하여 등록한다.

⑦ 화면 좌측의 프로젝트매니저 창에서 **Source Files**를 선택하고, 마우스 우측 버튼을 클릭하고, **Add Existing Source File(s)...**를 클릭한 후

lib 디렉토리 및 그 하위의 디렉토리에 있는 모든 **.c** 파일을 선택하여 등록한다.

⑧ 화면 좌측의 프로젝트매니저 창에서 **Source Files**를 선택하고, 마우스 우측 버튼을 클릭하고, **Add Existing Source File(s)...**를 클릭한 후 **apps** 디렉토리에 있는 **master.c** 파일을 선택하여 등록한 후 **Rename File...**하여 파일명을 바꾼다.(예, **sn.c**)

⑨ **Build** 메뉴에서 **Rebuild All**을 클릭하여 컴파일 및 **hex**파일을 생성한다.
(※ 오류가 발생하지 않아야 한다.)

[Step.2] hal_zbm.h 수정

hal_zbm.h는 **ATmega128(A,L)**의 I/O핀과 외부장치와의 핀 연결에 관한 초기화 매크로나 간단한 명령 매크로를 정의한다.
외부장치는 외부메모리, LCD, 각종 센서등, **ZBM v1.2**의 확장커넥터와 접속되는 모든 장치를 말한다.
따라서 사용자는 외부확장 커넥터(P4, P5)를 통해서 응용보드를 제어하려면 해당 I/O핀을 **hal_zbm.h**에 등록하고 초기화할 수 있도록 해야한다

① 프로젝트매니저 창에서 **External Dependancies**를 열고 **hal_zbm.h**를 더블클릭하여 편집창에 띄운다.

② **I/O 포트 정의** 부분에 사용하고자 하는 핀을 추가로 정의한다.

※ 본 예제의 서버노드는 외부장치를 장착하지 않으므로 필요하지 않음

③ **초기화 매크로 정의** 부분의 **I/O 포트 초기화 매크로 정의** 부분에 위에서 추가로 정의한 핀을 초기화한다.

※ 본 예제의 서버노드는 외부장치를 장착하지 않으므로 필요하지 않음

④ LCD모듈 등 사용자가 장착하는 응용보드에 대한 초기화나 명령 매크로가 필요한 경우 **사용자 추가 매크로** 부분에 추가한다.

※ 본 예제의 서버노드는 외부장치를 장착하지 않으므로 필요하지 않음

[Step.3] hal.h 수정

hal.h는 **ATmega128(A,L)**의 내부 리소스 사용에 관한 초기화 매크로나 간단한 명령 매크로를 정의한다.
현재 기본적으로 정의되어 있는 매크로는 다음과 같다. **(hal.h** 참조)

- SPI_INIT() 매크로
- SPI를 통한 **CC2420** 제어 매크로
- 1 usec 딜레이 함수 선언 (halWait)
- Global interrupt 제어 매크로 (sei, cli)
- UART0, UART1 초기화 및 사용과 관련된 매크로
- ADC 초기화 및 사용과 관련된 매크로

SPI는 **CC2420** 제어 전용으로 사용되므로 사용자가 SPI를 추가적으로 사용할 경우에는 세심한 주의를 요한다.
위에서 SPI 관련 항목을 제외하고 4가지 항목이 정의되어 있으므로 사용자는 내용을 충분히 파악해서 프로그램 개발에 활용하도록 한다.
위에서 언급되지 않은 항목을 사용하려면 **hal.h**에 관련된 매크로를 추가해준다. 예를 들어 다음과 같은 내부 리소스를 사용하려면 사용자가 초기화 및 사용과 관련된 매크로를 추가해 주어야 한다.

- 타이머/카운터
- 외부인터럽트

- 내부EEPROM

① 프로젝트매니저 창에서 **External Dependancies**를 열고 **hal.h**를 더블클릭하여 편집창에 띄운다.

② 기존에 등록되어 있지 않은 **ATmega128(A,L)** 내부 리소스를 사용하려면 관련된 초기화 및 명령 매크로를 **사용자 추가 매크로** 부분에 추가한다.

※ 본 예제의 서버노드 프로그램에서는 내부 리소스 사용에 관한 사용자 추가 매크로가 필요하지 않음

[Step.4] sn.c 작성

① **sn.c**를 더블클릭하여 편집창에 띄우고 서버노드에 대한 소스를 작성한다. (소스에 표시된 (1) ~ (9) 순)

② 작성이 완료되면 **Build** 메뉴에서 **Rebuild All**을 클릭하여 컴파일 및 **hex**파일을 생성한다.
(※ 오류가 발생하면 디버깅한다.)

◆ **sn.c**의 내용 (청색 글씨의 내용만 필요에 따라 수정) ◆

```
/*****************************************************************************************************
*           - PAN ID: 0x2176 for example                                                            *
*           - Short address: 0x0000 ~ 0xFFFF changeable                                             *
*****************************************************************************************************/

#include "include\include.h"

//-----------------------------------------------------------------------------------------------
// Basic RF transmission and reception structures
//-----------------------------------------------------------------------------------------------
```

```
BASIC_RF_RX_INFO rfRxInfo;
BASIC_RF_TX_INFO rfTxInfo;
BYTE pTxBuffer[BASIC_RF_MAX_PAYLOAD_SIZE];
BYTE pRxBuffer[BASIC_RF_MAX_PAYLOAD_SIZE];

//-------------------------------------------------------------------------------------------------
// (7) 사용자가 필요에 따라 만든 외부변수
//-------------------------------------------------------------------------------------------------
//
// 사용자 외부변수 추가…
//

//-------------------------------------------------------------------------------------------------
// (8) 사용자가 필요에 따라 만든 함수는 여기에 정의하거나 별도의 .c 파일에 기술한다.
//-------------------------------------------------------------------------------------------------
//
// 사용자 함수 추가…
//

//-------------------------------------------------------------------------------------------------
// 하위노드가 응답한 무선데이타 수신후 처리 함수(인터럽트 핸들러)
//-------------------------------------------------------------------------------------------------
BASIC_RF_RX_INFO* basicRfReceivePacket(BASIC_RF_RX_INFO *pRRI)
{
//-------------------------------------------------------------------------------------------------
// (9) pRRI->pPayload[] : 하위노드로부터 수신한 데이타 처리
//-------------------------------------------------------------------------------------------------
        unsigned char Command;
        unsigned int Fromaddr;

        // 패킷을 보낸 고정노드의 주소
        Fromaddr = pRRI->srcAddr;

        if(Fromaddr == rfTxInfo.destAddr) // 지금 수신하는 메시지의 발신자 주소가 직전에 내가 보냈던 곳의
        {                                 // 주소이면..하위 고정노드로부터의 응답 메시지임!
```

```
		Command = pRRI->pPayload[0];

		// 명령 분석
		switch(Command)
		{
			case 5: // 음성데이터 갱신 결과 보고가 올라옴!
			{
				// 메인컴퓨터(PC)로 상향패킷 송신
				UART1_WAIT_AND_SEND(5); // 음성데이터 갱신 결과를 보고한다
				UART1_WAIT_AND_SEND(pRRI->pPayload[1]); // 오류코드
				UART1_WAIT_AND_SEND(pRRI->pPayload[2]); // 고정노드 주소 하위바이트
				UART1_WAIT_AND_SEND(pRRI->pPayload[3]); // 고정노드 주소 상위바이트

				break;
			}
			default: // 하위 고정노드와 통신 오류 (패킷 오류)
			{
				// 메인컴퓨터(PC)로 상향패킷 송신
				UART1_WAIT_AND_SEND(5); // 음성데이터 갱신 결과를 보고한다
				UART1_WAIT_AND_SEND(2); // 오류코드 : 통신 오류 (패킷 오류)
				UART1_WAIT_AND_SEND(Fromaddr&0xFF); // 하위 고정노드 주소 하위바이트
				UART1_WAIT_AND_SEND((Fromaddr>>8)&0xFF); // 하위 고정노드 주소 상위바이트

				break;
			}

		} // switch(Command)

	} // if(Fromaddr == rfTxInfo.destAddr)

	return pRRI; // 수신대기 상태로 복귀

} // basicRfReceivePacket

//-----------------------------------------------------------------------------------------------
```

```
// main 함수
//-------------------------------------------------------------------------------------------------
int main(void)
{
//-------------------------------------------------------------------------------------------------
// (1) I/O포트, 변수, ATmega128 내부장치 및 주변장치 초기화
//-------------------------------------------------------------------------------------------------
        unsigned int srcaddr, destaddr;
        unsigned char i, length, dest_L, dest_H;

        PORT_INIT();
        SPI_INIT();
        INIT_UART1(UART_BAUDRATE_921K6,UART_OPT_8_BITS_PER_CHAR); // UART1 초기화(921.6Kbps)
        ENABLE_UART1();

//-------------------------------------------------------------------------------------------------
// (2) source address 설정
//-------------------------------------------------------------------------------------------------
        srcaddr = 0x0023; // 서버노드의 어드레스를 0x0023로 설정

//-------------------------------------------------------------------------------------------------
// (3) 채널 및 Pan ID 설정
//-------------------------------------------------------------------------------------------------
        basicRfInit(&rfRxInfo, 26, 0x2176, srcaddr); // 26번 채널 설정, Pan ID = 0x2176(예를 들어..)

//-------------------------------------------------------------------------------------------------
// (4) 전송데이타 크기 설정
//-------------------------------------------------------------------------------------------------
        rfTxInfo.length = 106; // 하향 패킷, 본 예제에서 최대 106 바이트(C,NOS,FNst2_L,FNst2_H,NOP_L,NOP_H,PCMDATA)

//-------------------------------------------------------------------------------------------------
// ACK 수신 여부 설정
//-------------------------------------------------------------------------------------------------
        rfTxInfo.ackRequest = TRUE;
```

```
        rfTxInfo.pPayload = pTxBuffer;
        rfRxInfo.pPayload = pRxBuffer;

//------------------------------------------------------------------------------------------------
// (5) 전송데이타 초기화
//------------------------------------------------------------------------------------------------
        for (unsigned char n = 0; n < 106; n++)
        {
                pTxBuffer[n] = 0; // 송신 버퍼 클리어
        }

        // Turn on RX mode
        basicRfReceiveOn();

//------------------------------------------------------------------------------------------------
// (6) 메인 루틴
//------------------------------------------------------------------------------------------------
        while(1) // 무한 반복
        {
//------------------------------------------------------------------------------------------------
// (6-1) 하위 첫번째 스테이지의 고정노드로 송신할 패킷 구성
//------------------------------------------------------------------------------------------------
                // PC로부터 첫번째 바이트 수신 : Length
                UART1_WAIT_AND_RECEIVE(length);

                // PC로부터 C 수신하여 무선 송신 버퍼에 저장
                UART1_WAIT_AND_RECEIVE(pTxBuffer[0]);

                // PC로부터 NOS 수신하여 무선 송신 버퍼에 저장
                UART1_WAIT_AND_RECEIVE(pTxBuffer[1]);

                // PC로부터 하위 첫번째 스테이지의 고정노드 주소 수신
                UART1_WAIT_AND_RECEIVE(dest_L); // 하위 고정노드 주소(low byte)
                UART1_WAIT_AND_RECEIVE(dest_H); //하위 고정노드 주소(high byte)
```

```
                // PC로부터 나머지 바이트 수신하여 무선 송신 버퍼에 저장
                for(i=2; i<length-2; i++)
                {
                        UART1_WAIT_AND_RECEIVE(pTxBuffer[i]);
                }

                // NOS 값 1 감소
                pTxBuffer[1] = pTxBuffer[1] - 1;

//-------------------------------------------------------------------------------------------------
// (6-2) destination address 설정
//-------------------------------------------------------------------------------------------------
                // 하위 첫번째 스테이지의 고정노드 주소
                destaddr = (unsigned int)dest_L;
                destaddr |= (unsigned int)(dest_H)<<8;
                rfTxInfo.destAddr = destaddr;

//-------------------------------------------------------------------------------------------------
// (6-3) 무선 송신
//-------------------------------------------------------------------------------------------------

                rfTxInfo.length = length - 2;

                // 하위 고정노드로 하향 패킷 발사!
                basicRfSendPacket(&rfTxInfo);

        } // while(1)

} // main()
```

7-3. 펌웨어 다운로드

[Step.1] 하드웨어 준비

① **ZBM v1.2** 모듈에, **AVR**용 JTAG장치의 커넥터와 전원을 장착한다. (※ 전원 OFF상태, JTAG장치와 PC간 케이블은 장착하지 않은 상태)

※ JTAG장치 주의사항

- JTAG장치에 전원 선택 스위치가 있는 경우, 타겟(**ZBM v1.2** 모듈)으로부터 전원을 공급받도록 스위치를 설정한다. 만약 PC로부터 전원을 공급받도록 설정하게 되면 JTAG장치의 종류에 따라서 타겟이나 JTAG장치 자체에 손상을 야기할 수 있다.
- **ZBM v1.2** 모듈은 JTAG장치에 +3.3V의 전원을 공급하므로, JTAG장치는 3.3V의 전원으로 구동이 가능해야 한다.

② **ZBM v1.2** 모듈의 전원을 ON한 후, JTAG장치와 PC간 케이블을 장착한다. 이것은 JTAG장치가 정상적으로 셋업된 후에 PC에서 정상적으로 JTAG장치를 인식할 수 있기 때문이다.

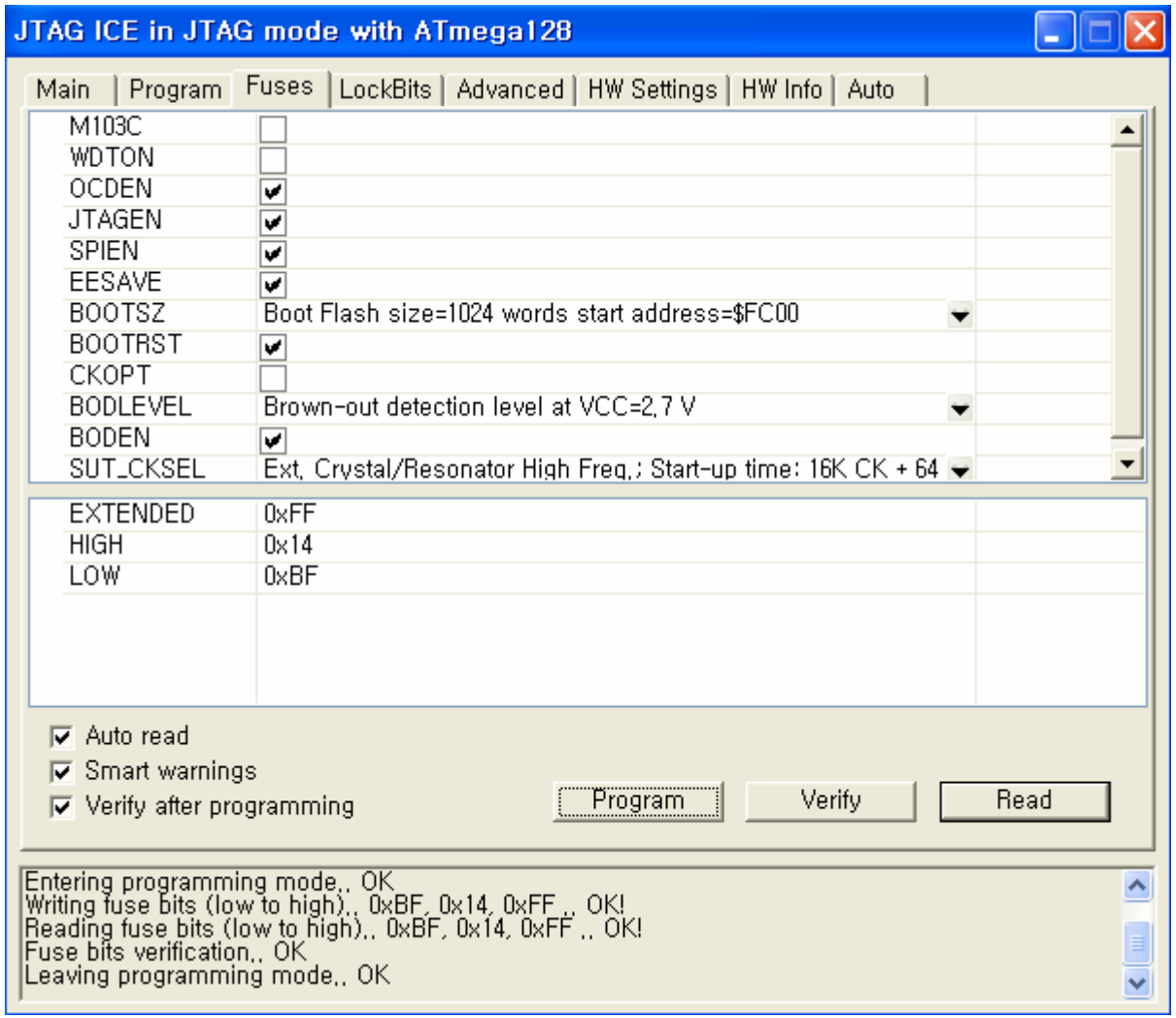
[Step.2] AVR Studio에서 다운로드

① **AVR Studio**에서 **Tools** 메뉴 아래에 **Program AVR -> Connect...**를 클릭하여 **Select AVR Programmer** 창을 띄운다.

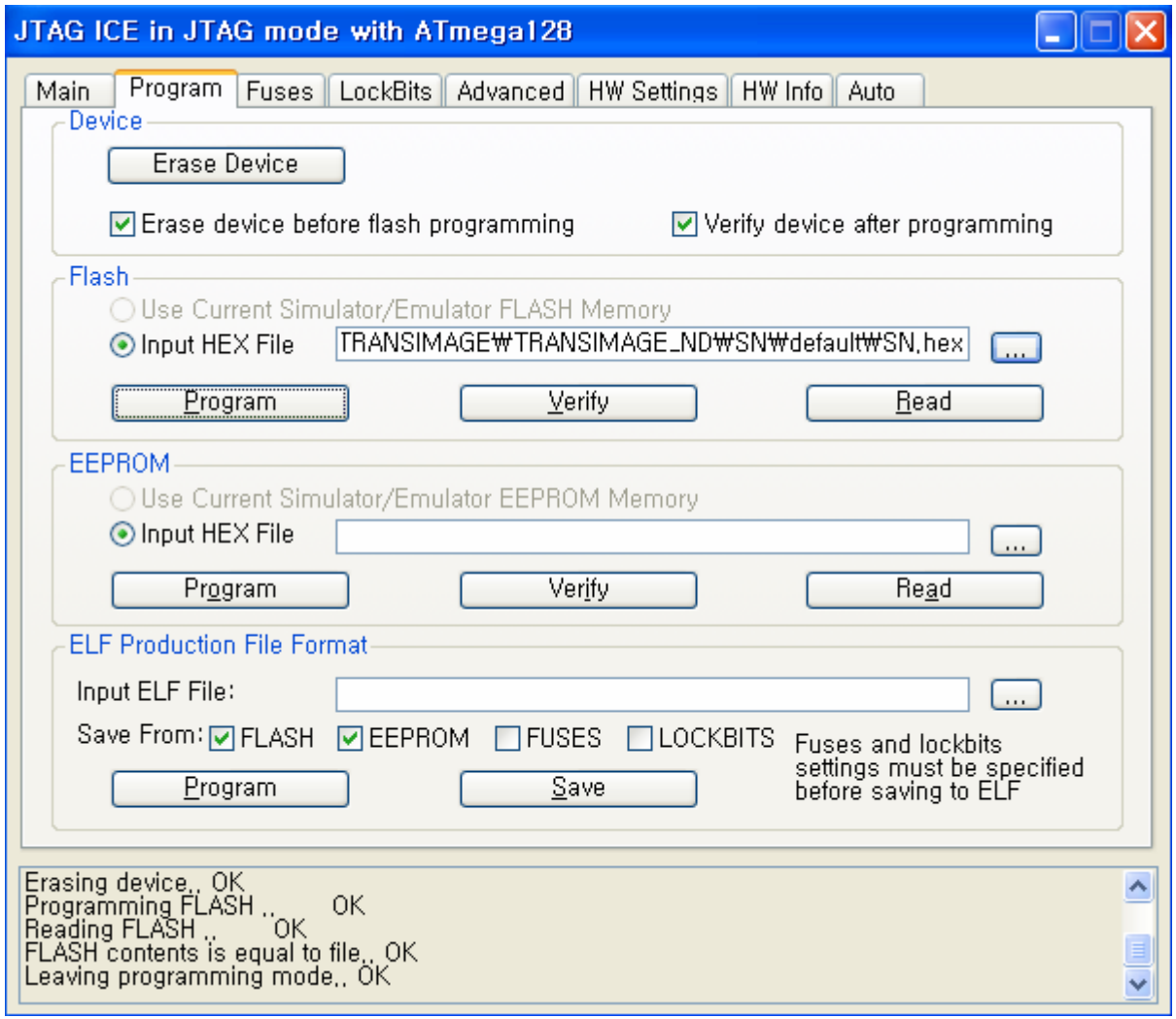
② **Select AVR Programmer** 창에서, **Platform: JTAG ICE, Port: COM3**(예, COM Port를 모르면 **Auto** 선택)을 선택하고 **Connect**를 클릭하여 **JTAG ICE** 창을 띄운다.

③ **JTAG ICE** 창에서 **Main** 탭을 선택하고, **Device**를 **ATmega128**로 선택한다.

④ **JTAG ICE** 창에서 **Fuses** 탭을 선택하고, 다음과 같이 퓨즈 비트를 체크해제 및 체크하고 **Program**을 클릭하여 퓨즈 비트를 설정한다.



⑤ **JTAG ICE** 창에서 **Program** 탭을 선택하고, 다운로드할 hex 파일의 위치를 지정하고 **Program**을 클릭하여 펌웨어를 다운로드한다.

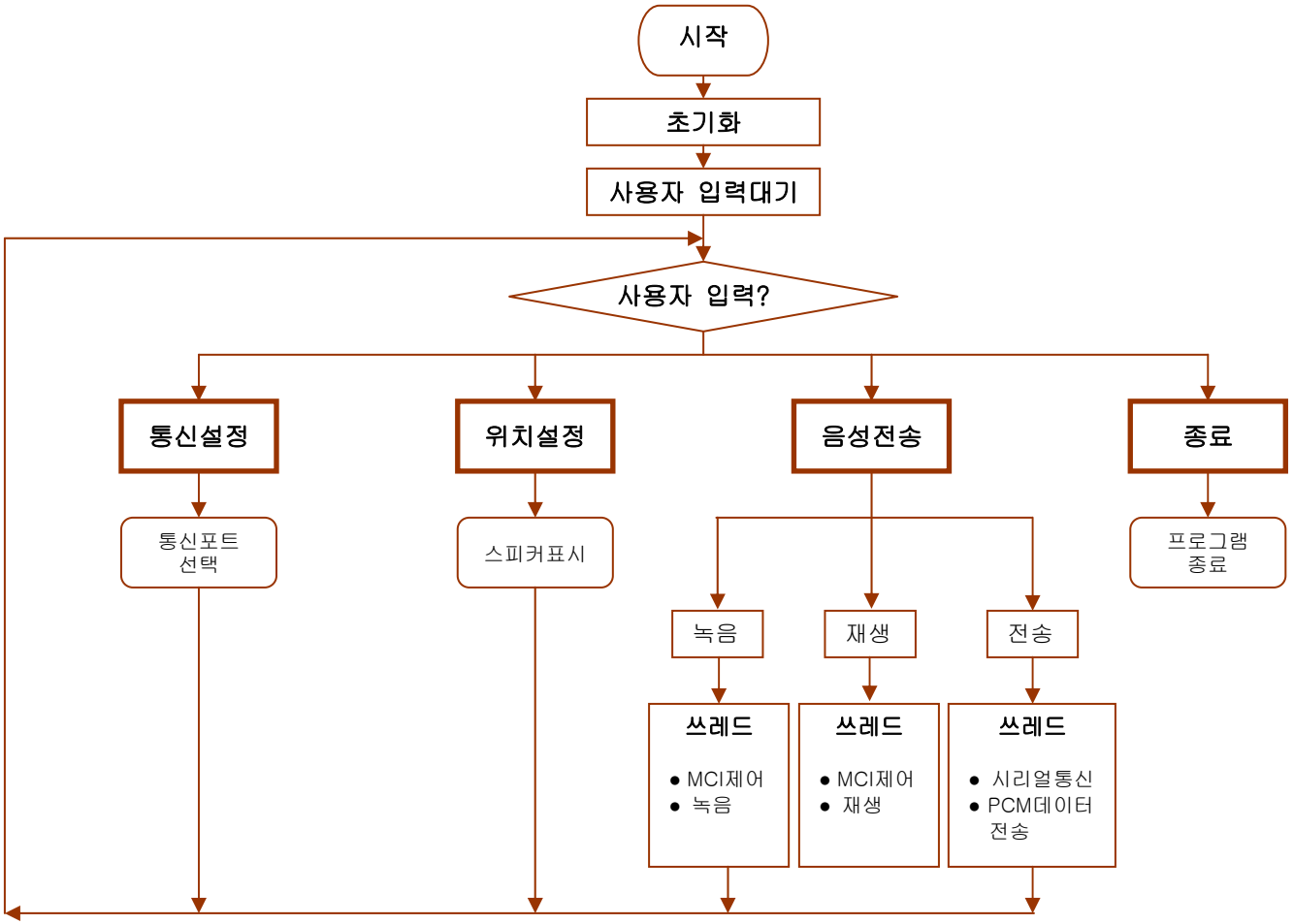


7-4. 메인컴퓨터 응용프로그램 제작

본 예제에서는 메인컴퓨터 응용프로그램을 **Visual C++ 6.0**을 사용하여 제작해 본다. **Visual C++** Tool은 최근에 **2008** 버젼이나 **2010** 버전 등이 사용되어지고 있지만, 공개 버전인 **Express** 버전에서는 **MFC**를 지원하지 않기 때문에 초보자가 GUI 응용프로그램을 제작하는 것은 거의 불가능하다고 할 수 있다. **Visual C++ 6.0**은 오랜 기간 동안 광범위하게 사용되어 왔고 교재나 예제가 풍부하며 **MFC**를 사용하여 고성능의 GUI를 제작하는데 전혀 손색이 없고, 주변에 이미 설치된 곳이 많아 접근이 용이하기 때문에 아직도 초보자 뿐만 아니라 중급자에게도 유용하다고 할 수 있다. 그러나 **윈도우XP**에서만 안정적으로 사용할 수 있다는 제약이 따른다.

여기서 제작할 메인컴퓨터 응용프로그램의 기능을 요약하면 다음과 같다.

- **GUI** 화면을 통하여 사용자와 인터페이스 처리
- **쓰레드** 기법을 이용하여 무선 음성전송 기능 구현
- **시리얼통신** 기능을 구현하여 서버노드와 패킷을 송수신
- **GUI** 상에서 **WAV** 형식의 음성을 녹음하고 재생
- 녹음된 **WAV** 파일로부터 **PCM** 데이터를 추출

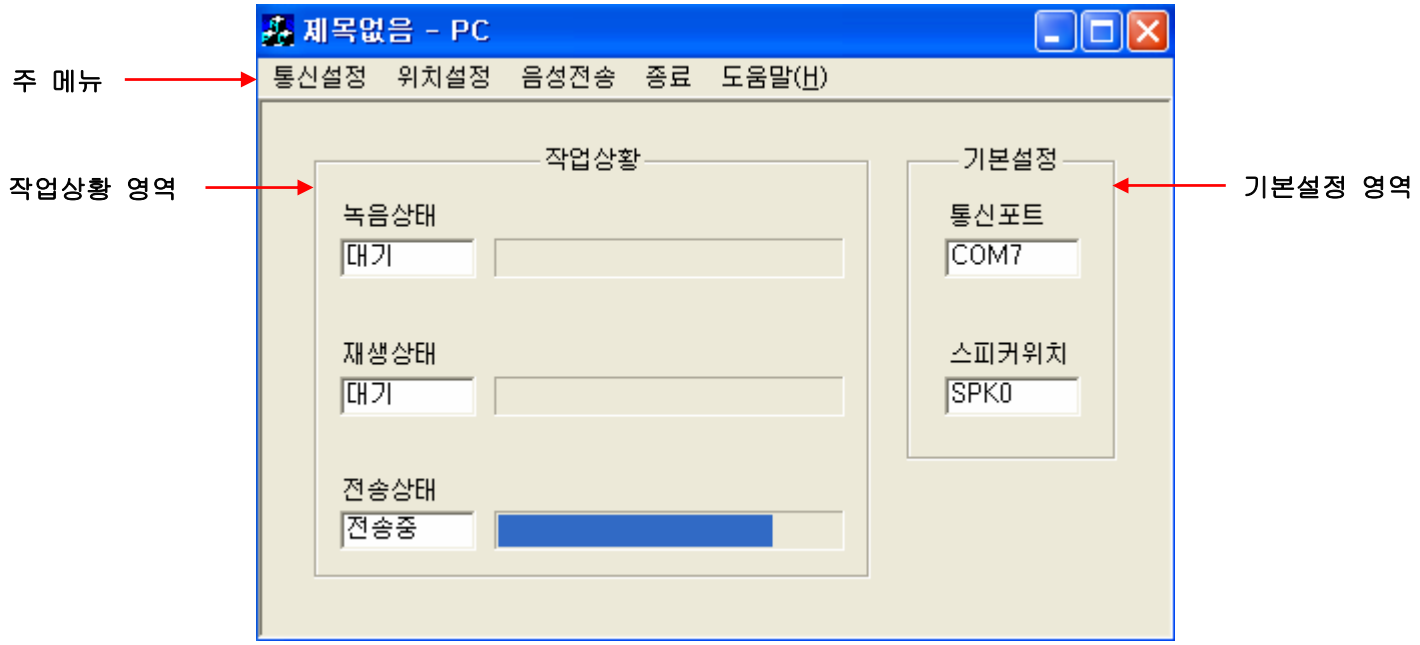


[프로그램 순서도]

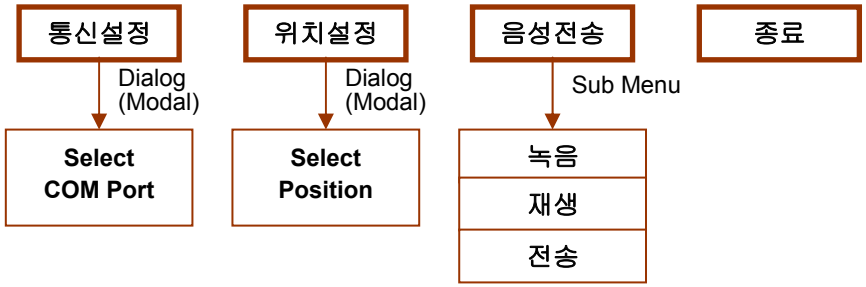
7-4-1. 프로그램 상위설계

메인컴퓨터 응용프로그램은 **MFC AppWizard(exe)**를 사용하여 **Single Document** 타입으로 제작하기로 한다. **Single Document** 타입은 화면의 상단에 주 메뉴가 가로로 나열되어 있고 그 하단에 뷰 영역으로 이루어진다. 본 예제에서는 주 메뉴를 통해 무선 음성전송 예제 시스템의 각종 설정과 제어를 하고, 뷰 영역에 음성 녹음, 재생, 전송 상황이 출력되는 작업상황 영역과, 현재의 설정상태를 보여주는 기본설정 영역을 배치한다.

7-4-1-1. 화면 구성

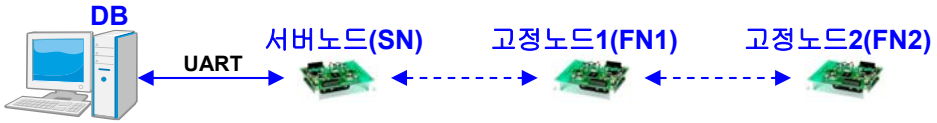


7-4-1-2. 메뉴 흐름



7-4-1-3. 시스템 파라메타 설정

본 예제의 네트워크 구조는 서버노드 아래에 2 단계(Stage)의 고정노드로 구성되는 형태로서, 네트워크 구성도와 라우팅테이블은 다음과 같다.



[예제 프로젝트의 네트워크 구성도]

Destination	No. of stages	Stage 1	Stage 2
0x01	1	0x01	
0x02	2	0x01	0x02

※ 어드레스는 고정노드 프로그램에서 임의로 지정한 값을 사용

[라우팅테이블]

프로그램 운용시에, 라우팅테이블을 메모리(버퍼)에 저장하고 하향 패킷을 구성할 때마다 수시로 참조하는 방식으로 프로그램을 제작한다. 이때 사용되는 버퍼는 2차원 배열을 사용하고 첫번째 열에는 최종단 목표 고정노드의 어드레스를, 두번째 열에는 단계수(NOS), 세번째 열부터는 단계별 고정노드 어드레스를 저장한다. 본 예제에서는 2단계 구조에 고정노드가 2개 이므로 2 x 4 요소의 2차원 배열이다.

m_RT_TABLE[2] [4] : 라우팅테이블 값을 저장하기 위한 배열

	Destination (고정노드) 0	**단계수 (NOS) 1**	**1단계 (Stage 1) 2**	**2단계 (Stage 2) 3**
0	0x0001	1	0x0001	
1	0x0002	2	0x0001	0x0002

[배열 구조]

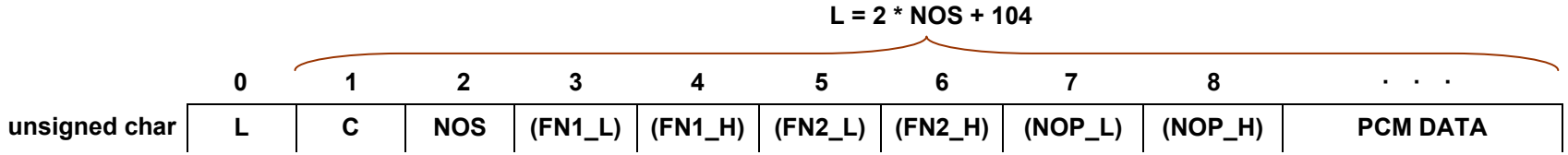
7-4-1-4. 송.수신 패킷 제작

메인컴퓨터와 서버노드 간에 시리얼포트로 주고 받는 송.수신 데이터 패킷과 상위의 고정노드와 하위의 고정노드 사이에 무선으로 주고 받는 송.수신 데이터 패킷을 제작한다.

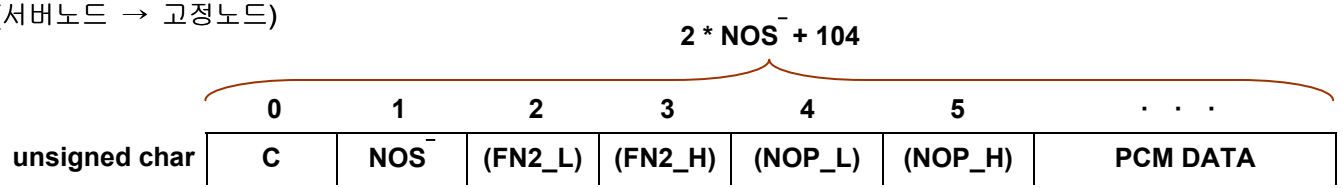
① 하향 패킷

◆ 음성데이터 갱신 명령

(메인컴퓨터 → 서버노드)



(서버노드 → 고정노드)



▶ L (Length) : 자신을 제외한 패킷의 바이트 수

▶ C (Command) : 명령코드

1 → 음성데이터를 갱신하라

▶ NOS (Number Of Stage) : 고정노드 단계 수

▶ NOS¯ : NOS - 1

▶ (FNst#) (Fixed Node Address) : #스테이지의 고정노드 어드레스

▶ (NOP_L) / (NOP_H) (Number Of Package Low / High byte) : PCM 데이터 패키지 번호 하위 및 상위 바이트

▶ PCM DATA : PCM 음성 데이터

② 상향 패킷

◆ 음성데이터 갱신 결과 보고

	0	1	2	3
unsigned char	C	ERR	(FNst#_L)	(FNst#_H)

► C (Command) : 응답코드

5 → 음성데이터 갱신 결과를 보고한다

► ERR (Error Code) : 오류 코드

0 → 정상

1 → 통신 오류(무응답)

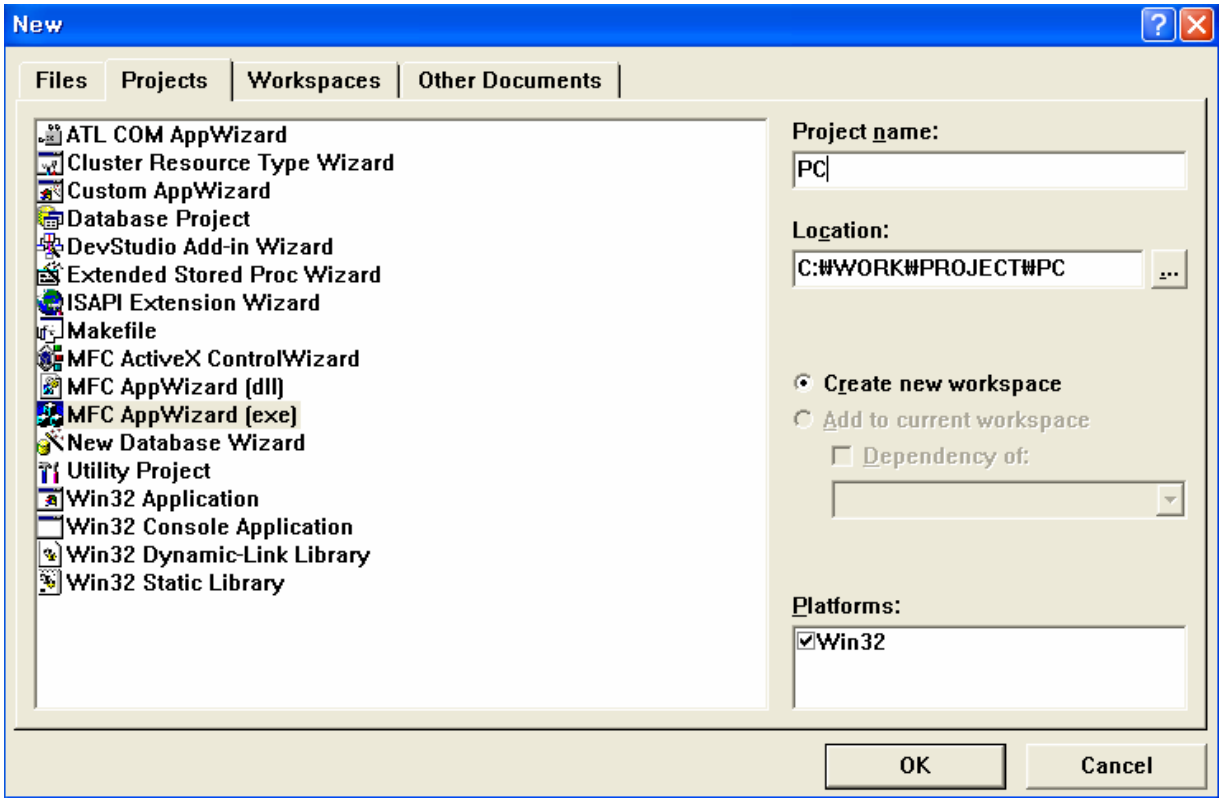
2 → 통신 오류(패킷 오류)

► (FNst#) (Fixed Node Address) : #스테이지의 고정노드 어드레스

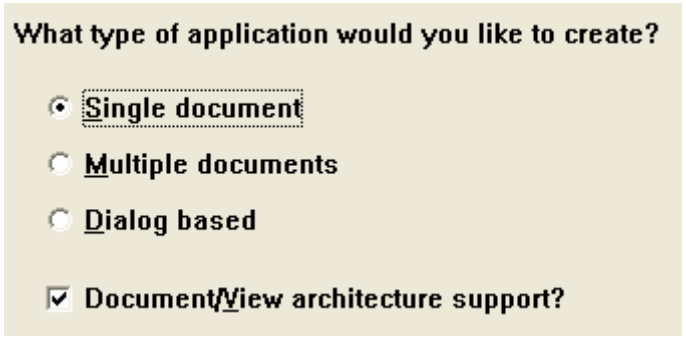
7-4-2. 프로그램 상세설계

7-4-2-1. 프로젝트 생성

• **Visual C++ 6.0**을 실행한 후, **File** 메뉴에서 **New**를 클릭하여 다음과 같이 **MFC AppWizard(exe)** 프로젝트를 생성한다.

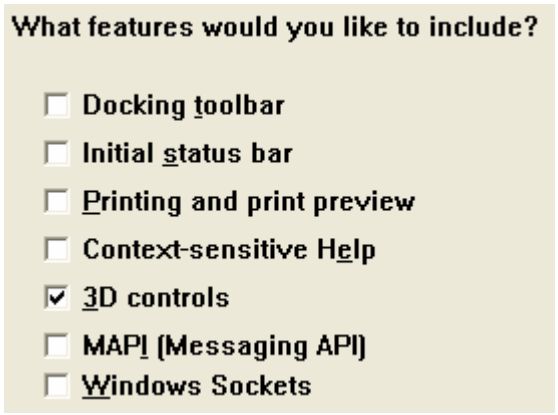


• MFC AppWizard-Step 1 화면에서 Single Document를 선택한다.

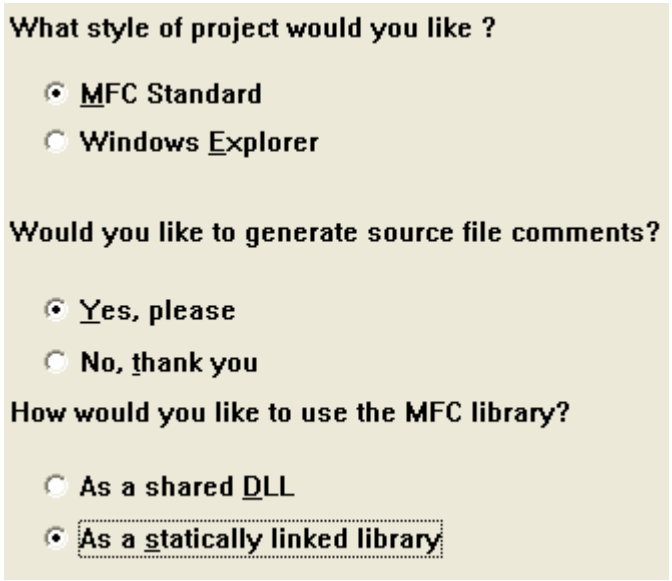


• Next를 2회 클릭하여 MFC AppWizard-Step 3 of 6 화면에서 ActiveX Control을 체크 해제한다.

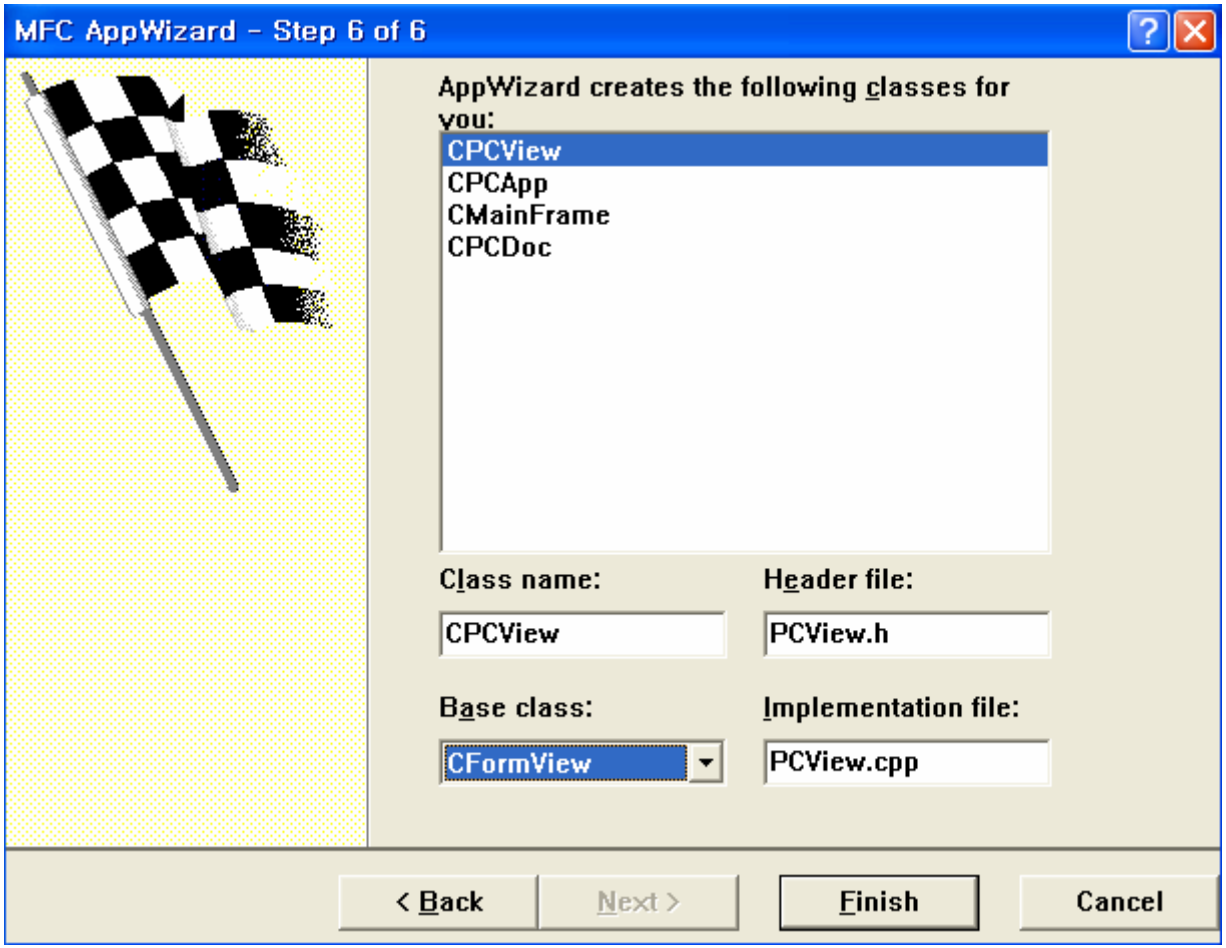
• Next를 1회 클릭하여 MFC AppWizard-Step 4 of 6 화면에서 3D controls를 제외하고 모두 체크 해제한다.



• Next를 1회 클릭하여 MFC AppWizard-Step 5 of 6 화면에서 다음과 같이 체크한다.



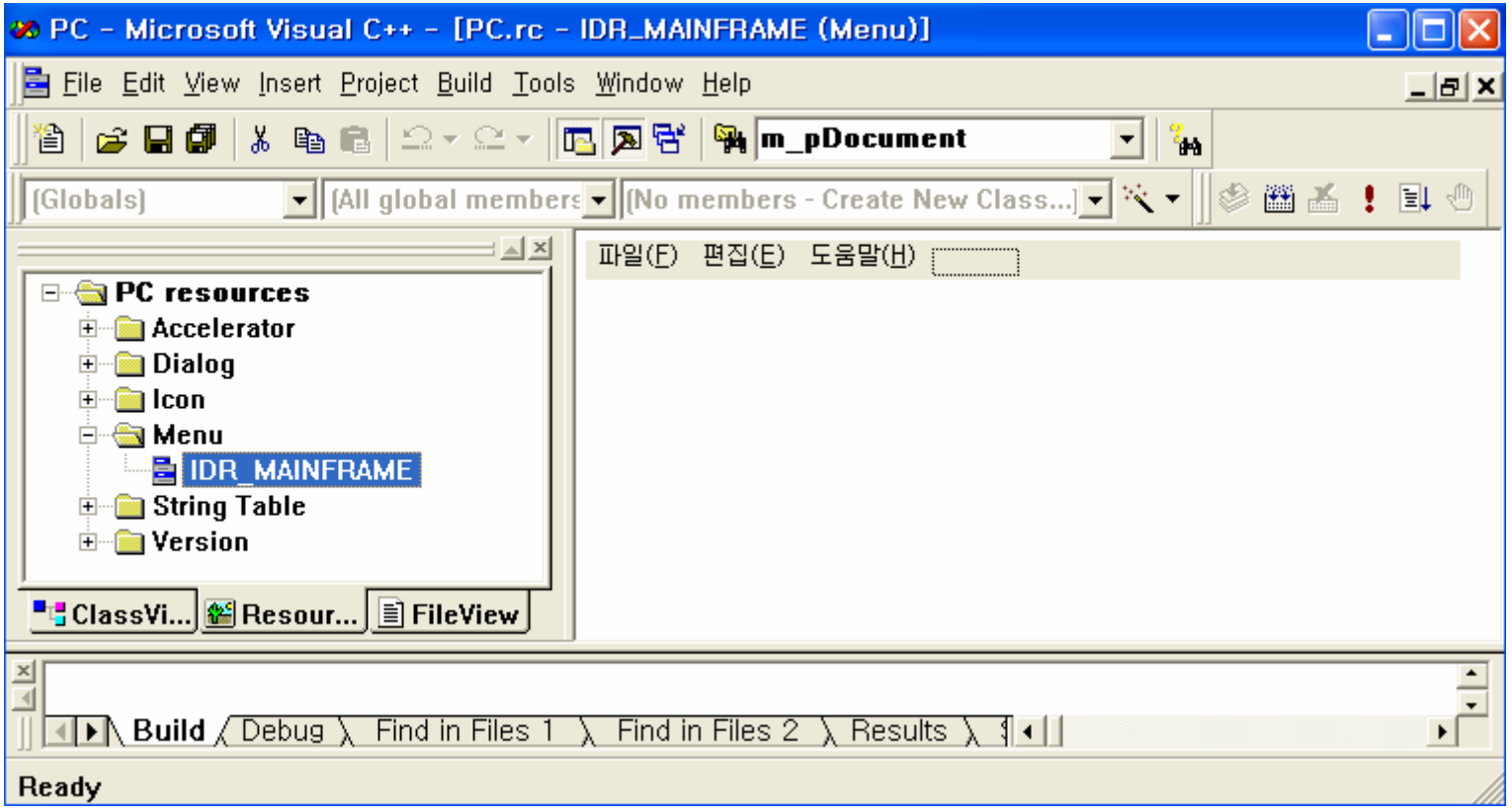
• Next를 1회 클릭하여 MFC AppWizard-Step 6 of 6 화면에서 CPCView 클래스의 Base class를 CFormView로 선택하고 Finish, OK를 클릭한다.



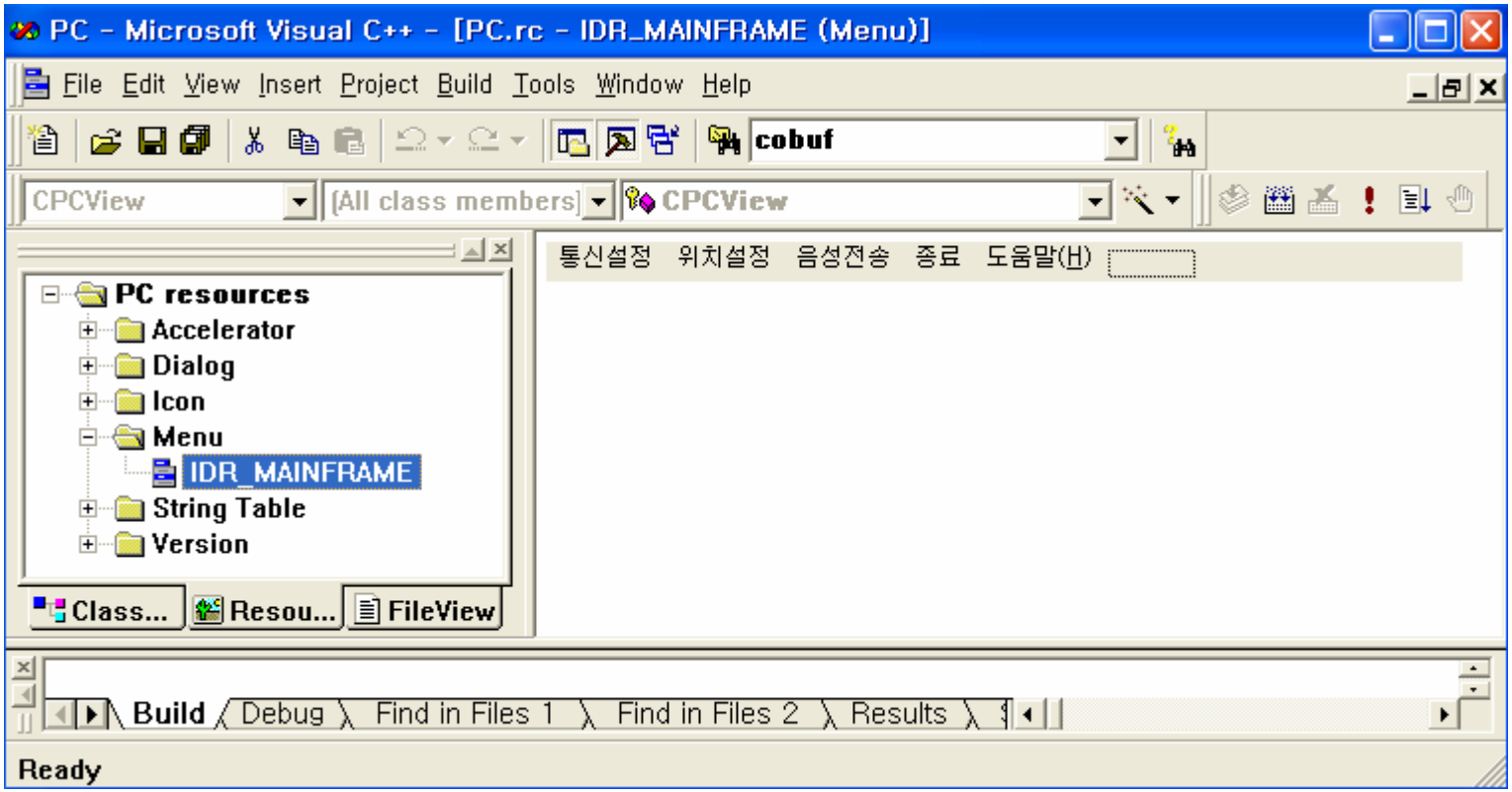
7-4-2-2. 기본 화면 제작

[Step.1] 주메뉴 생성

• **ResourceView** 창을 열고 **Menu** 아래에 있는 **IDR_MAINFRAME**을 더블클릭하여 메뉴 리소스 편집화면을 연다.



• **파일**과 **편집** 메뉴를 삭제하고, 네모난 점선상자를 더블클릭하여 **Caption**에 각각 **통신설정**, **위치설정**, **음성전송**, **종료**를 입력하여 메뉴를 추가한다.



[Step.2] 화면 제작

• **ResourceView** 창을 열고 **Dialog** 아래에 있는 **IDD_PC_FORM**을 더블클릭하여 메인 대화상자 리소스를 연다. (TODO: ... 텍스트 삭제)

• 메인 대화상자에 **Group Box**를 추가한다. 속성(**Properties**) 창을 띄운 후, **General** 탭 아래 **Caption**에 **작업상황**로 입력하고 **Styles** 탭 아래 **Horizontal alignment**를 **Center**로 지정한다.

• **Group Box** 안에 **Static Text** 3개와 **Edit Box** 3개를 세로 방향으로 짝을 맞추어 추가하고 속성(**Properties**) 창을 띄워서 속성을 다음과 같이 지정한다.

ID:	Caption:	(기타사항)
IDC_STATIC	녹음상태	
IDC_EDIT1		**Tap stop** 체크 해제
IDC_STATIC	재생상태	
IDC_EDIT2		**Tap stop** 체크 해제
IDC_STATIC	전송상태	
IDC_EDIT3		**Tap stop** 체크 해제

• **IDC_EDIT1**, **IDC_EDIT2**, **IDC_EDIT3** 우측에 각각 Progress 컨트롤을 추가하고 크기를 적당히 조절한다.

• 메인 대화상자의 우측에 **Group Box**를 추가한다. 속성(**Properties**) 창을 띄운 후, **General** 탭 아래 **Caption**에 **기본설정**으로 입력하고 **Styles** 탭 아래 **Horizontal alignment**를 **Center**로 지정한다.

• **Group Box** 안에 **Static Text** 3개와 **Edit Box** 2개를 세로 방향으로 짝을 맞추어 추가하고 속성(**Properties**) 창을 띄워서 속성을 다음과 같이 지정한다.

ID:	Caption:	(기타사항)
IDC_STATIC	통신포트	
IDC_EDIT4		**Tap stop** 체크 해제
IDC_STATIC	스피커위치	
IDC_EDIT5		**Tap stop** 체크 해제

7-4-2-3. 종료 메뉴 구현

• **ResourceView** 창을 열고 **Menu** 아래에 있는 **IDR_MAINFRAME**을 더블클릭하여 메뉴 리소스 편집화면을 연다. **종료** 메뉴를 더블클릭하여 속성(**Properties**) 창을 열고, **ID:**에 **ID_APP_EXIT**를 찾아서 지정한다.

• **ClassWizard**를 열고, **Message Maps** 탭을 클릭하고, **Class name**에 **CPCDoc**, **Object IDs**에 **ID_APP_EXIT**, **Messages**에 **COMMAND**를 선택하고, **Add Function…** 버튼을 클릭, 팝업 창에 **OK**를 클릭하고 **Edit Code** 버튼을 클릭하면 명령 핸들러인 **OnAppExit( )**가 만들어진다. **OnAppExit( )** 안에 다음과 같이 **ExitProcess(TRUE);**를 추가하여 프로그램을 종료하는 기능을 구현하고, 빌드후 테스트 해본다.

```
void CPCDoc::OnAppExit()
{
	// TODO: Add your command handler code here

	ExitProcess(TRUE);
}
```

7-4-2-4. 통신설정 메뉴 구현

실질적인 시리얼통신 기능은 범용으로 미리 제작되어 있는 클래스(**CSERIAL**)에서 수행되는데, 본 예제프로그램에서는 이 시리얼통신 클래스를 활용하도록 한다. **CSERIAL** 클래스는 **Serial.h**와 **Serial.cpp**로 이루어져 있다.
통신설정 메뉴를 클릭하면 작은 다이얼로그 창(**IDD_COMDLG**)이 뜨고 이 창에서 COM 포트를 선택하도록 구현한다.
일반적으로 통신설정이란 통신포트, 통신속도, Data bits, Stop bit, Parity 등등의 파라메타를 설정하는 것을 의미하지만, 본 예제에서는 프로그램의 단순화를 위하여 통신포트만 설정하도록 하고 나머지 파라메타는 **921.6Kbps, 8 Data bits, 1 Stop bit, No parity** 로 고정한다.
참고로, 921.6Kbps를 선택한 이유는, **ZBM v1.2** 모듈에서 사용하는 **AVR**이 7.3728 MHz의 X-Tal로 구동될 때, 시리얼통신 오차율 0%로 구현할 수 있는 최대의 속도이기 때문이다.

[Step.1] 시리얼통신 관련 파일 다운로드

당사 홈페이지(**http://www.olmicrowaves.com**)의 **Downloads** 메뉴 아래 **ZigBee** 카테고리에서 다음의 파일을 다운로드하여 프로젝트 디렉토리(**PC**)에 저장한다.

- **Serial.h**
- **Serial.cpp**

[Step.2] 시리얼통신 클래스 삽입

FileView 창을 열고 **PC files** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Files to Project**를 클릭하여 **Serial.h**와 **Serial.cpp**를 프로젝트에 삽입한다.

[Step.3] 메뉴 ID 설정

• **ResourceView** 창을 열고 **Menu** 아래에 있는 **IDR_MAINFRAME**을 더블클릭하여 메뉴 리소스 편집화면을 연다.

• **통신설정** 메뉴를 더블클릭, 속성(**Properties**) 창에서 **Pop-up**을 체크해제하고 **ID:**를 **IDM_COM_SELECT**으로 설정한다.

[Step.4] 메뉴 핸들러 생성

• **ClassWizard**를 열고, **Message Maps** 탭을 클릭하고, **Class name**에 **CPCDoc**, **Object IDs**에 **IDM_COM_SELECT**, **Messages**에 **COMMAND**를 선택하고, **Add Function…** 버튼을 클릭, 팝업 창에 **OK**를 클릭하고 **Edit Code** 버튼을 클릭하면 명령 핸들러인 **OnComSelect( )**가 만들어진다.

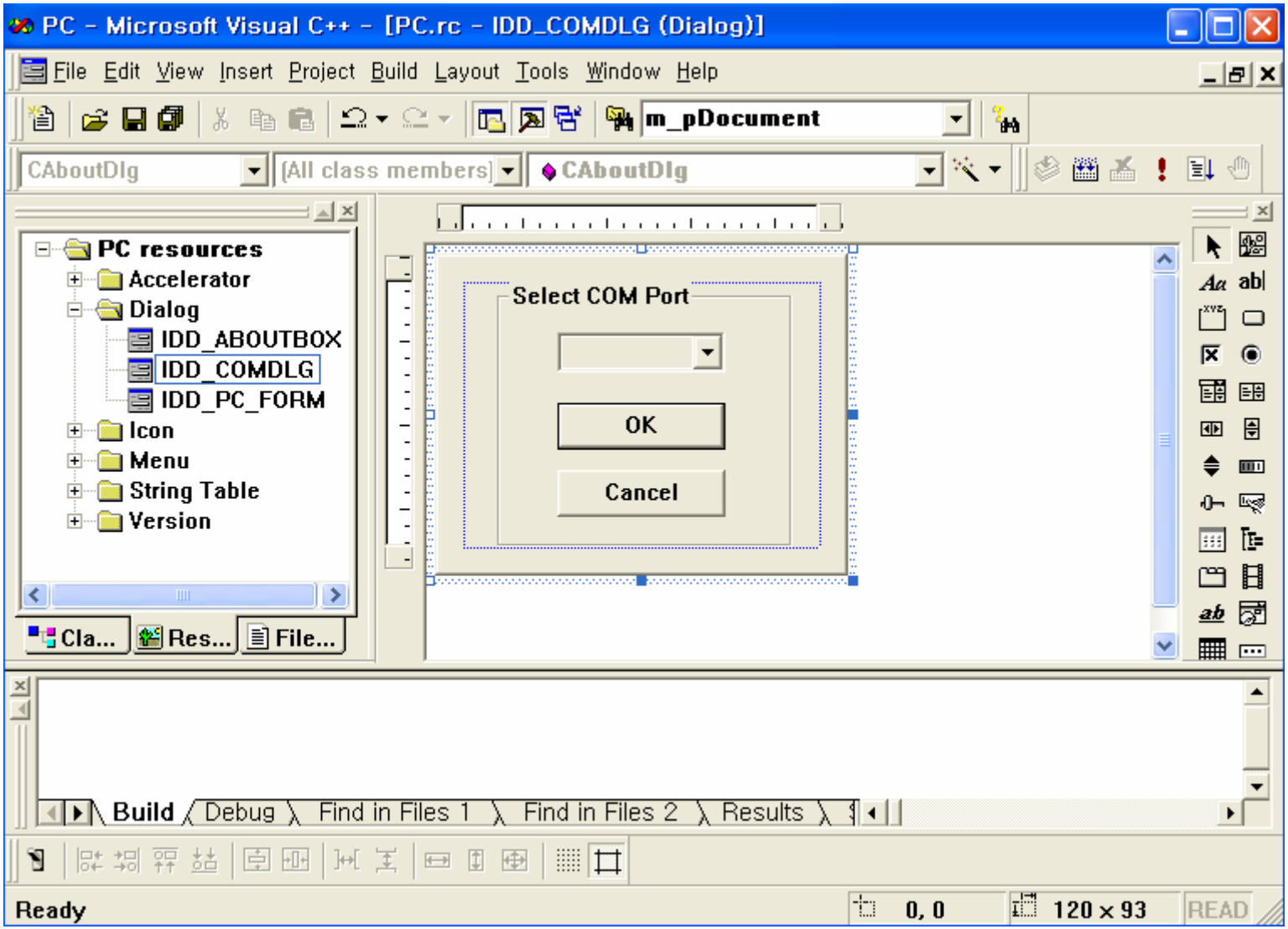
[Step.5] Select COM Port 대화상자 생성

• **ResourceView** 창을 열고 **Dialog** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Insert Dialog**를 클릭하여 새로운 대화상자를 만들고 속성(**Properties**) 창을 열어 **General** 탭에서 ID를 **IDD_COMDLG**로 설정하고, **Styles** 탭에서 **Title bar**를 체크 해제한다.

• **Combo Box**를 1개 추가하고 속성(**Properties**) 창을 열어 **General** 탭에서 ID를 **IDC_COMPORT**로 설정하고, **Data** 탭에서 COM1~COM9까지 입력 (※ 줄바꿈: **Ctrl+Enter**)하고, **Styles** 탭에서 **Sort**를 체크 해제한다.

• **Combo Box**의 ▼를 클릭하고 점선상자를 아래로 드래그하여 가시영역을 설정한다.

• **Combo Box**와 **OK** 및 **Cancel** 버튼을 적절히 위치시킨 후 **Group Box**로 둘러싸고 **Group Box**의 **Caption**을 **Select COM Port**로 입력한다.



[Step.6] IDD_COMDLG를 위한 CComDlg 클래스 생성

• **ClassWizard**를 연다.

• **IDD_COMDLG**을 위한 새로운 클래스의 생성 여부를 묻는 창이 뜨면 **OK**를 클릭하고, 그렇지 않으면 **Add Class**, **New**를 클릭한다.

• **Name**에 **CComDlg**로 입력하고, **Base class**를 **CDialog**로 설정하고, **Dialog ID**를 **IDD_COMDLG**로 설정하고 **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.7] CComDlg 클래스에 다이얼로그 초기화를 위한 메시지 핸들러 추가

• **ClassWizard**를 열고, **Class name**에 **CComDlg**, **Object IDs**에 **CComDlg**, **Messages**에 **WM_INITDIALOG**를 선택하고 **Add Function**을 클릭하면 **OnInitDialog** 함수가 생성된다. **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.8] CComDlg 클래스에 통신포트 설정을 위한 메시지 핸들러 추가

• **ClassWizard**를 열고, **Class name**에 **CComDlg**, **Object IDs**에 **IDC_COMPORT**, **Messages**에 **CBN_SELCHANGE**를 선택하고 **Add Function**을 클릭하면 **OnSelchangeComport** 함수가 생성된다. **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.9] CComDlg 클래스에 필요한 변수 추가

• **m_com** : **Combo Box**로부터 통신포트 값을 전달하기 위한 변수.

ClassWizard를 열고, **Member Variables** 탭을 클릭하고, **Class name**에 **CComDlg**, **Control IDs**에 **IDC_COMPORT**를 선택하고 **Add Variable**을 클릭한다. **Member variable name**에 **m_com**을 입력하고, **Category**에 **Control**을 선택하고, **Variable Type**에 **CComboBox**를 선택하고 **OK**를 클릭하여 **ClassWizard**를 닫는다.

• **com** : 통신포트 값을 **CComDlg** 클래스로부터 다른 클래스로 전달하기 위한 변수.

ClassView 창을 열고 **CComDlg** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **int**, **Varible Name**에 **com**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

[Step.10] OnSelchangeComport 함수 작성

CComDlg 클래스 내에 있는 **OnSelchangeComport()**를 더블클릭하여 열고 다음의 소스를 코딩해 넣는다. 이 소스는 콤보박스에서 통신포트를 선택했을 때 선택된 포트값을 전역변수 **com**에 저장하는 기능을 수행한다.

```
int tmp = m_com.GetCurSel();
if(tmp != CB_ERR)
{
        CString tmpr;
        m_com.GetLBText(tmp, tmpr);
        tmpr.Delete(0, 3);
        com = atoi(tmpr);
        if(com<=0) com = 0;
}
```

[Step.11] CPCDoc 클래스에 필요한 변수 추가

• **m_pSerial** : **CSERIAL** 클래스 타입의 객체로서 시리얼통신 관련 여러 함수를 제공한다.

ClassView 창을 열고 **CPCDoc** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **CSERIAL**, **Varible Name**에 **m_pSerial**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

• **m_hWnd** : **CSERIAL** 객체에서 수신한 시리얼 데이터를 **CPCDoc** 클래스에 전달하기 위한 윈도우 핸들

ClassView 창을 열고 **CPCDoc** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **HWND**, **Varible Name**에 **m_hWnd**를 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

• **m_strComport** : **CPCDoc** 클래스 내에 있는 **OnComSelect( )**에 의해 설정된 통신포트 값을 **CPCView** 클래스로 전달한다.

ClassView 창을 열고 **CPCDoc** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **CString**, **Varible Name**에 **m_strComport**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

[Step.12] 통신설정 메뉴 및 종료 메뉴 핸들러 작성

• **CPCDoc** 클래스의 멤버 함수 하나를 더블클릭하여 **PCDoc.cpp**를 열고 상단에 **ComDlg.h**, **PCView.h, MainFrm.h**를 include 한다.

```
// PCDoc.cpp : implementation of the CPCDoc class
//
```

```
#include "stdafx.h"
#include "PC.h"

#include "PCDoc.h"
#include "ComDlg.h"
#include "PCView.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

• **CPCDoc** 클래스 내에 있는 **OnComSelect( )**를 더블클릭하여 열고 다음의 소스를 코딩해 넣는다. 이 소스는 메뉴에서 **통신설정**을 클릭했을 때 **Select COM Port** 다이얼로그 박스가 팝업되고, 통신포트를 선택할 수 있도록 하는 기능을 수행한다.

```
void CPCDoc::OnComSelect()
{
	// TODO: Add your command handler code here

	CComDlg dlg;

	if(dlg.DoModal() == IDOK)
	{
		if((dlg.com > 0) && (dlg.com < 125))
		{
			m_pSerial.Stopcom();

			// Data bits(4-8), Stop bits(0,1,2:1,1.5,2), Parity(0-4=no,odd,even,mark,space)
			m_pSerial.Setting(dlg.com,921600,8,0,0); // 921.6Kbps, 8 Data bits, 1 Stop bit, No parity
			m_pSerial.Init();
```

```
                    CPCView* pView = (CPCView*)((CMainFrame*)(AfxGetApp()->m_pMainWnd))->GetActiveView();

                    if(!m_pSerial.Startcom())
                    {
                        m_strComport.Format("");
                        pView->GetDlgItem(IDC_EDIT4)->SetWindowText(m_strComport);
                        AfxMessageBox("Not Connected !");
                    }
                    else
                    {
                        m_strComport.Format("COM%d",dlg.com);
                        pView->GetDlgItem(IDC_EDIT4)->SetWindowText(m_strComport);
                    }

                    UpdateAllViews(NULL);
                    m_pSerial.SetHwnd(pView->m_hWnd); // WM_RECEIVEDATA 메시지를 CPCView 클래스로 보내도록 설정
                }
                else AfxMessageBox("Not Selected !");
            }
}
```

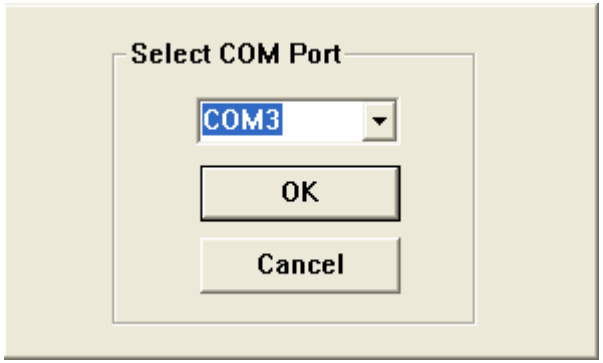
• **CPCDoc** 클래스 내에 있는 **OnAppExit( )**를 더블클릭하여 열고 다음의 소스를 코딩해 넣는다. 이 소스는 메뉴에서 **종료**를 클릭했을 때 설정된 통신포트를 해제하고 메인 응용프로그램을 종료하는 기능을 수행한다.

```
    m_pSerial.Release();
    ExitProcess(TRUE);
```

[Step.13] 통신포트 설정 기능 확인

지금까지의 작업을 저장하고 **Rebuild All**을 수행하여 **Error**와 **Warning**이 없는 것을 확인후 프로그램을 실행해본다.

통신설정 메뉴를 클릭하여 통신포트를 설정해본다.



[Step.14] 시리얼통신 데이터 수신메시지 핸들러 생성

• **CPCView** 클래스를 더블클릭하여 **PCView.h**를 열고 시리얼통신 데이터 수신메시지 핸들을 위한 AFX_MSG 함수 선언을 추가한다.

```
// Generated message map functions
protected:
        //{{AFX_MSG(CPCView)
                // NOTE - the ClassWizard will add and remove member functions here.
                //      DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        afx_msg BOOL OnSerialreceive(UINT WParam , LONG LParam);
        DECLARE_MESSAGE_MAP()
};
```

• **PCView.cpp**를 열고 상단의 Message Map부분에 시리얼통신 데이터 수신메시지 핸들러를 위한 Message Map을 추가한다.

```
BEGIN_MESSAGE_MAP(CPCView, CFormView)
        //{{AFX_MSG_MAP(CPCView)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //      DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        ON_MESSAGE(WM_RECEIVEDATA, OnSerialreceive)
END_MESSAGE_MAP()
```

[Step.15] 시리얼통신 데이터 수신메시지 핸들러 함수 작성

고정노드로부터 시리얼 데이타 수신시 임시 저장용 버퍼와 Flag를 **CPCDoc** 클래스에 만들어 두고, **CPCView** 클래스에서 참조하도록 한다.

• **m_RX_DATA[ ]** : 수신한 시리얼 데이타를 임시 저장해둘 배열변수

ClassView 창을 열고 **CPCDoc** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **unsigned char**, **Varible Name**에 **m_RX_DATA[100]**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

• **m_RxComplete** : 시리얼 데이터 수신 완료 Flag

ClassView 창을 열고 **CPCDoc** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **int**, **Varible Name**에 **m_RxComplete**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

시리얼통신 데이터 수신 메시지 핸들러를 작성한다.

• **CPCView** 클래스의 멤버 함수 하나를 더블클릭하여 **PCView.cpp**를 열고, 하단에 **OnSerialreceive( )** 함수를 다음과 같이 코딩해 넣는다. 이 소스는 서브노드로부터 시리얼 데이터가 수신되었다는 윈도우 메시지가 발생하면 수행된다.

```
BOOL CPCView::OnSerialreceive(UINT WParam, LONG LParam)
{
	int length, tmp;
	unsigned char tmpc;

	CPCDoc *pDoc = GetDocument();

	length = (int)WParam;

	tmpc = 0;

	while ((tmpc == 0) && (length>0)) // 앞쪽에 0x00으로 입력된 더미 제거
	{
		pDoc->m_pSerial.bufread.Rxchar(&tmpc);
		length--;
	}

	if (length>=0) pDoc->m_RX_DATA[0] = tmpc;

	for(tmp=1; tmp<=length; tmp++)
	{
		pDoc->m_pSerial.bufread.Rxchar(&tmpc); // 시리얼 수신버퍼에서 1바이트 읽는다.
		pDoc->m_RX_DATA[tmp] = tmpc;
	}

	pDoc->m_RxComplete = 1;

	return TRUE;
}
```

지금까지의 작업을 저장하고 **Rebuild All**을 수행하여 **Error**와 **Warning**이 없는 것을 확인후 프로그램을 실행해본다.

7-4-2-5. 위치설정 메뉴 구현

위치설정 메뉴를 클릭하면 작은 다이얼로그 창(**IDD_POSDLG**)이 뜨고 이 창에서 카메라 위치를 선택하도록 구현한다.

[Step.1] 메뉴 ID 설정

• **ResourceView** 창을 열고 **Menu** 아래에 있는 **IDR_MAINFRAME**을 더블클릭하여 메뉴 리소스 편집화면을 연다.

• **위치설정** 메뉴를 더블클릭, 속성(**Properties**) 창에서 **Pop-up**을 체크해제하고 **ID:**를 **IDM_POS_SELECT**으로 설정한다.

[Step.2] 메뉴 핸들러 생성

• **ClassWizard**를 열고, **Message Maps** 탭을 클릭하고, **Class name**에 **CPCDoc**, **Object IDs**에 **IDM_POS_SELECT**, **Messages**에 **COMMAND**를 선택하고, **Add Function…** 버튼을 클릭, 팝업 창에 **OK**를 클릭하고 **Edit Code** 버튼을 클릭하면 명령 핸들러인 **OnPosSelect( )**가 만들어진다.

[Step.3] Select Speaker Position 대화상자 생성

• **ResourceView** 창을 열고 **Dialog** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Insert Dialog**를 클릭하여 새로운 대화상자를 만들고 속성(**Properties**) 창을 열어 **General** 탭에서 ID를 **IDD_POSDLG**로 설정하고, **Styles** 탭에서 **Title bar**를 체크 해제한다.

• **Combo Box**를 1개 추가하고 속성(**Properties**) 창을 열어 **General** 탭에서 ID를 **IDC_SPKPOS**로 설정하고, **Data** 탭에서 **SPK0**, **SPK1**를 입력

(※ 줄바꿈: **Ctrl+Enter)**하고, **Styles** 탭에서 **Sort**를 체크 해제한다.

• **Combo Box**의 ▼를 클릭하고 점선상자를 아래로 드래그하여 가시영역을 설정한다.

• **Combo Box**와 **OK** 및 **Cancel** 버턴을 적절히 위치시킨 후 **Group Box**로 둘러싸고 **Group Box**의 **Caption**을 **Select Speaker Position**로 입력한다.

[Step.4] IDD_POSDLG를 위한 CPosDlg 클래스 생성

• **ClassWizard**를 연다.

• **IDD_POSDLG**을 위한 새로운 클래스의 생성 여부를 묻는 창이 뜨면 **OK**를 클릭하고, 그렇지 않으면 **Add Class**, **New**를 클릭한다.

• **Name**에 **CPosDlg**로 입력하고, **Base class**를 **CDialog**로 설정하고, **Dialog ID**를 **IDD_POSDLG**로 설정하고 **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.5] CPosDlg 클래스에 다이얼로그 초기화를 위한 메시지 핸들러 추가

• **ClassWizard**를 열고, **Class name**에 **CPosDlg**, **Object IDs**에 **CPosDlg**, **Messages**에 **WM_INITDIALOG**를 선택하고 **Add Function**을 클릭하면 **OnInitDialog** 함수가 생성된다. **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.6] CPosDlg 클래스에 카메라 위치 설정을 위한 메시지 핸들러 추가

• **ClassWizard**를 열고, **Class name**에 **CPosDlg**, **Object IDs**에 **IDC_SPKPOS**, **Messages**에 **CBN_SELCHANGE**를 선택하고 **Add Function**을

클릭하면 **OnSelchangeCampos** 함수가 생성된다. **OK**를 클릭하여 **ClassWizard**를 닫는다.

[Step.7] CPosDlg 클래스에 필요한 변수 추가

• **m_pos** : **Combo Box**로부터 스피커 위치 값을 전달하기 위한 변수.

ClassWizard를 열고, **Member Variables** 탭을 클릭하고, **Class name**에 **CPosDlg**, **Control IDs**에 **IDC_SPKPOS**를 선택하고 **Add Variable**을 클릭한다. **Member variable name**에 **m_pos**을 입력하고, **Category**에 **Control**을 선택하고, **Variable Type**에 **CComboBox**를 선택하고 **OK**를 클릭하여 **ClassWizard**를 닫는다.

• **pos** : 스피커의 위치 값을 **CPosDlg** 클래스로부터 다른 클래스로 전달하기 위한 변수.

ClassView 창을 열고 **CPosDlg** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **int**, **Varible Name**에 **pos**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

[Step.8] OnSelchangeSpkpos 함수 작성

CPosDlg 클래스 내에 있는 **OnSelchangeSpkpos()**를 더블클릭하여 열고 다음의 소스를 코딩해 넣는다. 이 소스는 콤보박스에서 스피커의 위치를 선택했을 때 선택된 위치값을 전역변수 **pos**에 저장하는 기능을 수행한다.

```
int tmp = m_pos.GetCurSel();
if(tmp != CB_ERR)
{
        CString tmpr;
        m_pos.GetLBText(tmp, tmpr);
```

```
            tmpr.Delete(0, 3);
            pos = atoi(tmpr);
            if(pos<=0) pos = 0;
    }
```

[Step.9] CPCDoc 클래스에 필요한 변수 추가

• **m_SpkPos** : **CPCDoc** 클래스 내에 있는 **OnPosSelect( )**에 의해 설정된 스피커의 위치 값.

ClassView 창을 열고 **CPCDoc** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **int**, **Varible Name**에 **m_Spkpos**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

• **m_strSpkpos** : **m_Spkpos** 값의 앞에 “SPK”을 붙여 스트링 타입으로 **CPCView** 클래스로 전달한다.

ClassView 창을 열고 **CPCDoc** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에 **CString**, **Varible Name**에 **m_strSpkpos**을 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

[Step.10] 위치설정 메뉴 핸들러 작성

• **CPCDoc** 클래스의 멤버 함수 하나를 더블클릭하여 **PCDoc.cpp**를 열고 상단에 **PosDlg.h**를 include 한다.

```
// PCDoc.cpp : implementation of the CPCDoc class
//

#include "stdafx.h"
#include "PC.h"
```

```
#include "PCDoc.h"
#include "ComDlg.h"
#include "PCView.h"
#include "MainFrm.h"
#include "PosDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

• **CPCDoc** 클래스 내에 있는 **OnPosSelect( )**를 더블클릭하여 열고 다음의 소스를 코딩해 넣는다. 이 소스는 메뉴에서 **위치설정**을 클릭했을 때 **Select Speaker Position** 다이얼로그 박스가 팝업되고, 스피커의 위치를 선택할 수 있도록 하는 기능을 수행한다.

```
void CPCDoc::OnPosSelect()
{
	// TODO: Add your command handler code here

	CPosDlg dlg;

	if(dlg.DoModal() == IDOK)
	{
		m_CamPos = dlg.pos;
		m_strCampos.Format("CAM%d",m_CamPos);


		CPCView* pView = (CPCView*)((CMainFrame*)(AfxGetApp()->m_pMainWnd))->GetActiveView();
		pView->GetDlgItem(IDC_EDIT5)->SetWindowText(m_strCampos);

		UpdateAllViews(NULL);
	}
}
```

[Step.11] 스피커 위치 설정 기능 확인

지금까지의 작업을 저장하고 **Rebuild All**을 수행하여 **Error**와 **Warning**이 없는 것을 확인후 프로그램을 실행해본다.

위치설정 메뉴를 클릭하여 스피커의 위치를 설정해본다.

7-4-2-6. 음성전송 메뉴 구현

음성전송 메뉴를 클릭하면 **녹음/재생/전송** 부메뉴가 팝업되고, **녹음** 부메뉴를 클릭하면 음성 메시지 녹음 작업이 수행되며 **재생** 부메뉴를 클릭하면 녹음된 음성 메시지의 재생 작업이 수행된다. 녹음, 재생, 전송 부메뉴를 클릭하면 해당 작업을 처리하는 동안에 메인컴퓨터 응용프로그램이 먹통이 되는 것을 방지하기 위해 각각 별도의 쓰레드를 생성하여 작업을 처리하도록 한다.

음성을 녹음하고 재생하는 동작을 구현하기 위해서 윈도우즈에서 제공하는 **MCI**(Multimedia Control Interface) 기능을 이용한다. **MCI**는 음성데이터를 고수준에서 간편하게 제어할 수 있도록 **OPEN**, **RECORD**, **SAVE**, **PLAY**, **CLOSE** 등 몇 가지의 명령을 제공한다.
MCI를 사용하기 위해서는 **mmsystem.h**를 include 하고 **Project Settings**에 **winmm.lib**를 Link해 주어야 한다. 혹은, **Windows Multimedia library** 컴포넌트를 **Add To Project** 해도 동일한 역할을 수행한다.
MCI 명령을 수행하는 함수는 **mciSendCommand** 함수로서, 함수 인자는 디바이스 ID, 명령 메시지, 메시지 플래그, 메시지 포인터로 구성된다.

본 예제에서는 **WAV** 형식으로 8kHz 샘플링, 샘플당 8비트, 모노 방식, 10초간 음성을 녹음하고, 파일로 저장하고, 재생하도록 한다. 저장된 **WAV** 파일을 다시 읽어들여서 **PCM** 데이터를 추출하여 고정노드로 전송하고, 고정노드에서는 이 **PCM** 데이터를 **R2R DAC**를 이용하여 음성신호로 복원한 후 스피커로 출력하도록 한다.

WAV 파일 포맷 = **44**바이트 헤드 + 데이터 바이트

샘플링 주파수 : 8 kHz
샘플당 비트수 : 8
모노 방식
녹음 시간 : 10초

총 PCM 데이터 비트 = 8000 x 8 x 10 = 640000
총 PCM 데이터 바이트 = 640000 / 8 = **80000**
WAV 파일 용량 = 80000 + 44 = 80044

[Step.1] 메뉴 ID 설정

• **ResourceView** 창을 열고 **Menu** 아래에 있는 **IDR_MAINFRAME**을 더블클릭하여 메뉴 리소스 편집화면을 연다.
• **음성전송** 메뉴를 더블클릭, 속성(**Properties**) 창에서 **Pop-up**을 체크한다. 하위 메뉴를 만들기 위해 점선 박스를 더블클릭하고, **ID:**를 **IDM_RECORD**로 **Caption:**을 **녹음**으로 설정한다. **재생** 메뉴와 **전송** 메뉴도 동일한 방법으로 제작한다.
(**ID: IDM_PLAY / Caption: 재생**, **ID: IDM_TRANSMIT / Caption: 전송**)

[Step.2] CPCDoc 클래스에 필요한 변수 추가

• **m_PcBusy** : 응용프로그램이 현재 음성 녹음이나 재생 혹은 전송 작업 상태에 있음을 나타내는 Flag

ClassView 창을 열고 **CPCDoc** 위에서 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴를 열고 **Add Member Variable**을 클릭하여 **Variable Type**에

int, **Varible Name**에 **m_PcBusy**를 입력하고 **Access**는 **Public**으로 설정한 후 **OK**를 클릭한다.

• **m_FnInuse** : 응용프로그램이 현재 고정노드 접근 상태에 있음을 나타내는 Flag

m_PcBusy와 동일한 과정으로 반복한다.(**Variable Type**: **int**, **Varible Name**: **m_FnInuse**, **Access**: **Public**)

• **m_RT_TABLE[2] [4]** : 라우팅테이블 값을 저장하는 시스템 파라메타.

m_PcBusy와 동일한 과정으로 반복한다.(**Variable Type**: **unsigned int**, **Varible Name**: **m_RT_TABLE[2] [4]**, **Access**: **Public**)

• **m_Timeout** : 시리얼포트 수신부에서, 서버노드로부터 무응답에 의한 타임아웃 여부를 나타내는 Flag

m_PcBusy와 동일한 과정으로 반복한다.(**Variable Type**: **BOOL**, **Varible Name**: **m_Timeout**, **Access**: **Public**)

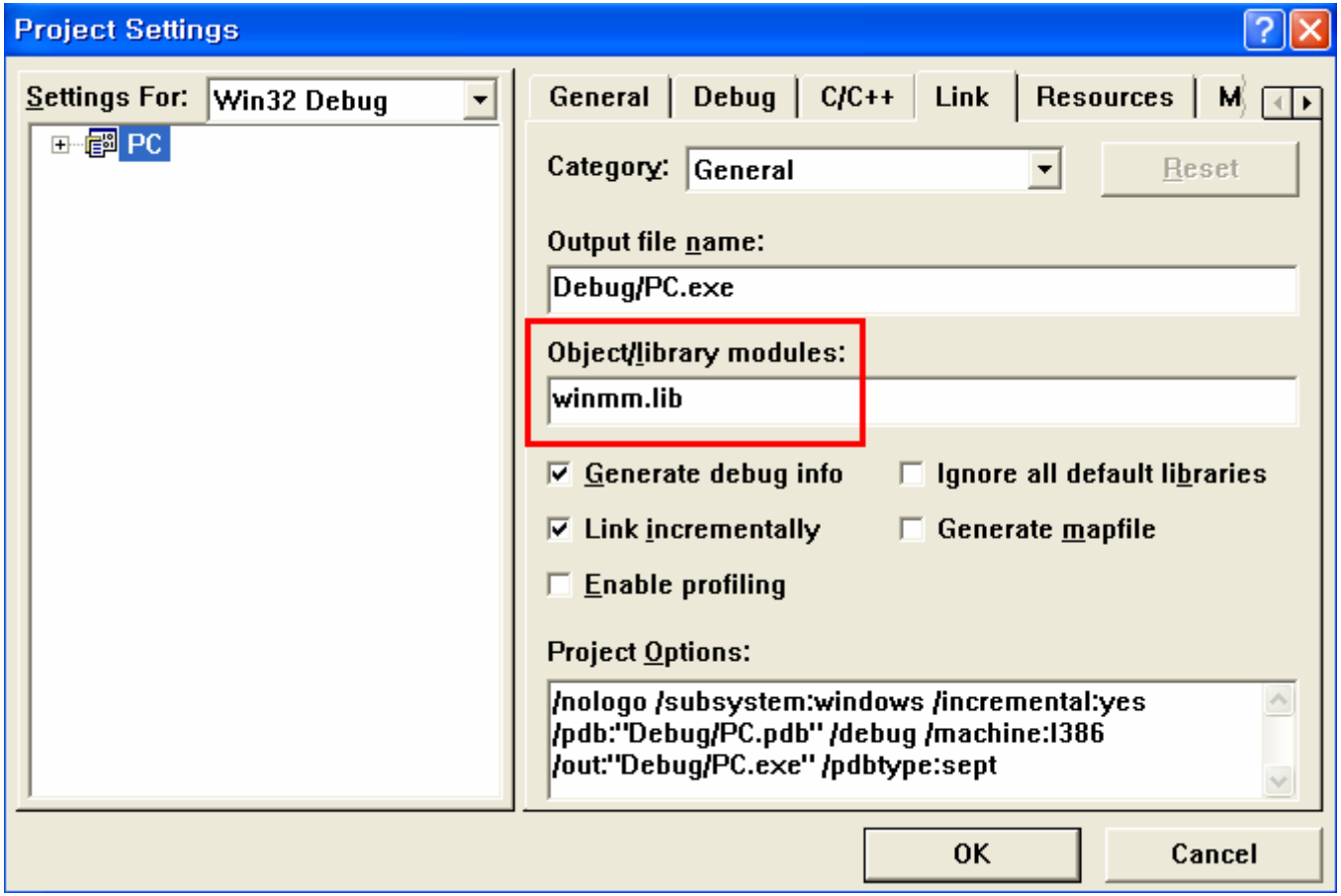
[Step.3] 메뉴 핸들러 생성

• **CPCDoc** 클래스의 멤버 함수 하나를 더블클릭하여 **PCDoc.cpp**를 열고 상단에 **mmsystem.h**를 include 한다.

```
// PCDoc.cpp : implementation of the CPCDoc class
//

#include "stdafx.h"
#include "PC.h"

#include "PCDoc.h"
#include "ComDlg.h"
```

```
#include "PCView.h"
#include "MainFrm.h"
#include "PosDlg.h"

#include "mmsystem.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

• **Project** 메뉴에서 **Settings**를 클릭하고 **Link**탭을 선택하여 다음과 같이 **.lib** 파일명을 지정해 준다.



• **ClassWizard**를 열고, **Member Variables** 탭을 클릭하고, **Class name**에 **CPCView**, **Control IDs**에 **IDC_PROGRESS1**를 선택하고 **Add Variable**을 클릭한다. **Member variable name**에 **m_record**을 입력하고, **Category**에 **Control**을 선택하고, **Variable Type**에 **CProgressCtrl**를 선택하고 **OK**를 클릭하여 **ClassWizard**를 닫는다. **m_record**은 **IDC_PROGRESS1**을 제어하기 위한 컨트롤 변수이다.

• 위의 과정을 **IDC_PROGRESS2**와 **IDC_PROGRESS3**에 대해서도 반복하여, 각각의 프로그레스를 제어하기 위한 멤버변수 **m_play**와 **m_transmit**를 생성한다.

• **ClassWizard**를 열고, **Message Maps** 탭을 클릭하고, **Class name**에 **CPCDoc**, **Object IDs**에 **IDM_RECORD**, **Messages**에 **COMMAND**를 선택하고, **Add Function…** 버튼을 클릭, 팝업 창에 **OK**를 클릭하고 **Edit Code** 버튼을 클릭하면 명령 핸들러인 **OnRecord( )**가 만들어진다. **OnRecord( )** 안에 다음의 소스를 코딩해 넣는다. 이 소스는 음성 메시지를 녹음 작업을 수행하는 쓰레드(**ThreadRecord**)를 가동하는 코드가 포함되는데, **ThreadRecord( )**는 차후에 작성하도록 한다.

```
void CPCDoc::OnRecord()
{
	// TODO: Add your command handler code here

	if (m_PcBusy == 0)
	{
		// 쓰레드 실행
		AfxBeginThread(ThreadRecord, this);
	}
}
```

• 동일한 방법으로 **재생** 메뉴(**IDM_PLAY**)에 대한 명령 핸들러인 **OnPlay( )**를 만들고 다음의 소스를 코딩해 넣는다. 이 소스는 녹음된 음성 메시지를 재생하는 작업을 수행하는 쓰레드(**ThreadPlay**)를 가동하는 코드가 포함되는데, **ThreadPlay( )**는 차후에 작성하도록 한다.

```
void CPCDoc::OnPlay()
{
	// TODO: Add your command handler code here

	if (m_PcBusy == 0)
	{
```

```
            // 쓰레드 실행
            AfxBeginThread(ThreadPlay, this);
        }
}
```

• 동일한 방법으로 **전송** 메뉴(**IDM_TRANSMIT**)에 대한 명령 핸들러인 **OnTransmit( )**를 만들고 다음의 소스를 코딩해 넣는다. 이 소스에는 녹음된 음성 메시지를 하위노드로 전송하는 작업을 수행하는 쓰레드(**ThreadTransmit**)를 가동하는 코드가 포함되는데, **ThreadTransmit( )**는 차후에 작성하도록 한다.

```
void CPCDoc::OnTransmit()
{
        // TODO: Add your command handler code here

        if ((m_PcBusy == 0) && (m_FnInuse == 0))
        {
            // 쓰레드 실행
            AfxBeginThread(ThreadTransmit, this);
        }
}
```

[Step.4] 통신설정 및 위치설정 명령 핸들러에 m_PcBusy 판별 코드 추가

• **PCDoc.cpp**의 **OnComSelect( )** 함수와 **OnPosSelect( )** 함수의 초반에 m_PcBusy 값을 판별하는 코드를 추가한다. 이 것은, 녹음이나 재생 혹은 전송 동작이 수행되는 도중에 통신설정 메뉴나 위치설정 메뉴를 클릭할 때, 기존 동작에 간섭을 주지 않도록 하기 위함이다.

```
void CPCDoc::OnComSelect()
{
        // TODO: Add your command handler code here

        if(m_PcBusy == 1) return;
```

```
        CComDlg dlg;

        if(dlg.DoModal() == IDOK)
        {
        ...이하 생략...


void CPCDoc::OnPosSelect()
{
        // TODO: Add your command handler code here

        if(m_PcBusy == 1) return;

        CPosDlg dlg;

        if(dlg.DoModal() == IDOK)
        {
        ...이하 생략...
```

[Step.5] OnNewDocument 함수에 변수 초기화

• **PCDoc.cpp**의 **OnNewDocument( )** 함수에 m_RT_TABLE, m_SpkPos, m_FnInuse, m_PcBusy, m_RxComplete 값을 초기화한다. 이들은 응용프로그램이 기동될 때 초기화된다.

```
BOOL CPCDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
```

```
        // 라우팅 테이블
        m_RT_TABLE[0][0] = 0x0001;      // SPK0 스피커(고정노드1)의 주소는 0x0001
        m_RT_TABLE[0][1] = 1;
        m_RT_TABLE[0][2] = 0x0001;

        m_RT_TABLE[1][0] = 0x0002;      // SPK1 스피커(고정노드2)의 주소는 0x0002
        m_RT_TABLE[1][1] = 2;
        m_RT_TABLE[1][2] = 0x0001;
        m_RT_TABLE[1][3] = 0x0002;

        m_SpkPos = 0;   // SPK0 스피커(고정노드1)를 위치설정 초기값으로 설정
        m_FnInuse = 0;
        m_PcBusy = 0;
        m_RxComplete = 0;

        return TRUE;
}
```

[Step.6] 음성 녹음 쓰레드 제어함수 제작

• **PCDoc.cpp**의 **OnRecord( )** 함수 바로 앞에 다음의 쓰레드 제어함수 소스를 코딩해 넣는다.

```
// 음성 녹음 쓰레드 제어함수
UINT ThreadRecord(LPVOID pParam)
{
        MCI_WAVE_SET_PARMS mciSetparms;
        MCI_STATUS_PARMS mciStatus;
        MCI_GENERIC_PARMS mciGeneric;
        MCI_OPEN_PARMS mciOpen;
        MCI_RECORD_PARMS mciRecord;
        MCI_SAVE_PARMS mciSave;
        MCIDEVICEID dwDeviceID;
```

```
DWORD TotalLen, CurrentPos=0, dwSec, result=0;

CPCDoc *pDoc = (CPCDoc *)pParam;
CPCView* pView = (CPCView*)((CMainFrame*)(AfxGetApp()->m_pMainWnd))->GetActiveView();

pDoc->m_PcBusy = 1; // 음성 녹음 작업을 수행함!

pView->GetDlgItem(IDC_EDIT1)->SetWindowText((CString)"녹음중");

// Open MCI device
mciOpen.lpstrDeviceType = "waveaudio";
mciOpen.lpstrElementName = "";

result = mciSendCommand(0,MCI_OPEN,MCI_OPEN_ELEMENT|MCI_OPEN_TYPE,(DWORD)(LPVOID)&mciOpen);

dwDeviceID = mciOpen.wDeviceID;

// Wave parameter setting
mciSetparms.wFormatTag = WAVE_FORMAT_PCM;
mciSetparms.wBitsPerSample = 8;
mciSetparms.nChannels = 1;
mciSetparms.nSamplesPerSec = 8000;
mciSetparms.nAvgBytesPerSec = ((mciSetparms.wBitsPerSample)/8) *
                                                    mciSetparms.nChannels *
                                                    mciSetparms.nSamplesPerSec;
mciSetparms.nBlockAlign = ((mciSetparms.wBitsPerSample)/8) *
                                                    mciSetparms.nChannels;

result = mciSendCommand(dwDeviceID,MCI_SET,MCI_WAIT |
                                                MCI_WAVE_SET_FORMATTAG |
                                                MCI_WAVE_SET_BITSPERSAMPLE |
                                                MCI_WAVE_SET_CHANNELS |
                                                MCI_WAVE_SET_SAMPLESPERSEC |
                                                MCI_WAVE_SET_AVGBYTESPERSEC |
```

```
                                        MCI_WAVE_SET_BLOCKALIGN,
                                        (DWORD)(LPVOID)&mciSetparms);

// 녹음 시간 설정
dwSec = 10;
mciRecord.dwTo = dwSec * 1000; // 10초 동안 녹음!
TotalLen = mciRecord.dwTo; // 총 녹음 길이 : 10초

// 녹음 Progress 컨트롤 초기화
pView->m_record.SetRange(0,(short)TotalLen);
pView->m_record.SetPos(0);

mciStatus.dwItem = MCI_STATUS_POSITION;

// 녹음
result = mciSendCommand(dwDeviceID,MCI_RECORD,MCI_TO|MCI_NOTIFY,(DWORD)(LPVOID)&mciRecord);

// 녹음 시간 동안 Progress 컨트롤 작동
while(CurrentPos < TotalLen)
{
        mciSendCommand(dwDeviceID,MCI_STATUS,MCI_STATUS_ITEM,(DWORD)(LPVOID)&mciStatus);
        CurrentPos = mciStatus.dwReturn;
        pView->m_record.SetPos(CurrentPos);
}

// 녹음된 내용을 파일로 저장
mciSave.lpfilename = "tmpwav.wav";
result = mciSendCommand(dwDeviceID,MCI_SAVE,MCI_SAVE_FILE|MCI_WAIT,(DWORD)(LPVOID)&mciSave);

// Close MCI device
result = mciSendCommand(dwDeviceID,MCI_CLOSE,MCI_WAIT,(DWORD)(LPVOID)&mciGeneric);

// 녹음 Progress 컨트롤 초기화
pView->m_record.SetPos(0);
```

```
        pView->GetDlgItem(IDC_EDIT1)->SetWindowText((CString)"대기");

        pDoc->m_PcBusy = 0; // 음성 녹음 작업을 종료함!

        return 0;
}
```

[Step.7] 음성 재생 쓰레드 제어함수 제작

• **PCDoc.cpp**의 **OnPlay( )** 함수 바로 앞에 다음의 쓰레드 제어함수 소스를 코딩해 넣는다.

```
// 음성 재생 쓰레드 제어함수
UINT ThreadPlay(LPVOID pParam)
{
        MCI_STATUS_PARMS mciStatus;
        MCI_GENERIC_PARMS mciGeneric;
        MCI_OPEN_PARMS mciOpen;
        MCI_PLAY_PARMS mciPlay;
        MCIDEVICEID dwDeviceID;

        DWORD TotalLen, CurrentPos=0, result=0;

        CPCDoc *pDoc = (CPCDoc *)pParam;
        CPCView* pView = (CPCView*)((CMainFrame*)(AfxGetApp()->m_pMainWnd))->GetActiveView();

        pDoc->m_PcBusy = 1; // 음성 재생 작업을 수행함!

        pView->GetDlgItem(IDC_EDIT2)->SetWindowText((CString)"재생중");

        // Open MCI device
        mciOpen.lpstrDeviceType = "waveaudio";
        mciOpen.lpstrElementName = "tmpwav.wav";
```

```
result = mciSendCommand(0,MCI_OPEN,MCI_WAIT|MCI_OPEN_ELEMENT,(DWORD)(LPVOID)&mciOpen);

dwDeviceID = mciOpen.wDeviceID;

// 총 녹음 길이 파악 : 10초
mciStatus.dwItem = MCI_STATUS_LENGTH;
mciSendCommand(dwDeviceID,MCI_STATUS,MCI_STATUS_ITEM,(DWORD)(LPVOID)&mciStatus);
TotalLen = mciStatus.dwReturn;

// 재생 Progress 컨트롤 초기화
pView->m_play.SetRange(0,(short)TotalLen);
pView->m_play.SetPos(0);

mciStatus.dwItem = MCI_STATUS_POSITION;

// 재생
result = mciSendCommand(dwDeviceID,MCI_PLAY,MCI_NOTIFY,(DWORD)(LPVOID)&mciPlay);

// 재생 시간 동안 Progress 컨트롤 작동
while(CurrentPos < TotalLen)
{
        mciSendCommand(dwDeviceID,MCI_STATUS,MCI_STATUS_ITEM,(DWORD)(LPVOID)&mciStatus);
        CurrentPos = mciStatus.dwReturn;
        pView->m_play.SetPos(CurrentPos);
}

// Close MCI device
result = mciSendCommand(dwDeviceID,MCI_CLOSE,MCI_WAIT,(DWORD)(LPVOID)&mciGeneric);

// 재생 Progress 컨트롤 초기화
pView->m_play.SetPos(0);

pView->GetDlgItem(IDC_EDIT2)->SetWindowText((CString)"대기");

pDoc->m_PcBusy = 0; // 음성 재생 작업을 종료함!
```

```
	return 0;
}
```

[Step.8] 음성 전송 쓰레드 제어함수 제작

• **PCDoc.cpp**의 **OnTransmit( )** 함수 바로 앞에 다음의 쓰레드 제어함수 소스를 코딩해 넣는다.

```
// 음성 전송 쓰레드 제어함수
UINT ThreadTransmit(LPVOID pParam)
{
	CPCDoc *pDoc = (CPCDoc *)pParam;
	CPCView* pView = (CPCView*)((CMainFrame*)(AfxGetApp()->m_pMainWnd))->GetActiveView();

	pDoc->m_PcBusy = 1; // 음성 전송 작업을 수행함!
	pDoc->m_FnInuse = 1; // 고정노드 사용중

	pView->GetDlgItem(IDC_EDIT3)->SetWindowText((CString)"전송중");

	// wav 파일 읽기
	CFile cfile;
	if( !cfile.Open( "tmpwav.wav", CFile::modeRead ) )
	{
		AfxMessageBox("전송할 음성 데이터가 없습니다!");

		pView->GetDlgItem(IDC_EDIT3)->SetWindowText((CString)"대기");

		pDoc->m_PcBusy = 0;
		pDoc->m_FnInuse = 0;

		return -1;
	}
```

```
DWORD dwSize = cfile.GetLength();
unsigned char *pWav;
pWav = new unsigned char [dwSize];

cfile.Read( pWav, dwSize );
cfile.Close();

// PCM 데이터 파싱
unsigned char *pPcm;
pPcm = pWav + 44;

// 전송 Progress 컨트롤 초기화
pView->m_transmit.SetRange(0,800);
pView->m_transmit.SetPos(0);

/*****************************************************************
 pPcm부터 100 바이트 식 총 dwSize-44(= 80000) 바이트의 PCM 데이터 전송
*****************************************************************/

unsigned char TxPacket[109]; // 하향패킷 최대사이즈 = 109바이트 = 오버헤드 9바이트 + PCM 데이터 100바이트
DWORD NOP, NOB;
for(NOP=0; NOP<800; NOP++)
{
        // 하향 패킷 구성 : Length, C, NOS, FN#_L, FN#_H, NOP_L, NOP_H, PCM DATA
        TxPacket[0] = pDoc->m_RT_TABLE[pDoc->m_SpkPos][1] * 2 + 104; // Length = 2*NOS + 104
        TxPacket[1] = 1; // Command
        TxPacket[2] = pDoc->m_RT_TABLE[pDoc->m_SpkPos][1]; // NOS

        for (unsigned int i=0; i<pDoc->m_RT_TABLE[pDoc->m_SpkPos][1]; i++)
        {
                TxPacket[i*2+3] = pDoc->m_RT_TABLE[pDoc->m_SpkPos][i+2] & 0xFF; // FN#_L
                TxPacket[i*2+4] = ((pDoc->m_RT_TABLE[pDoc->m_SpkPos][i+2] & 0xFFFF) >> 8) & 0xFF; // FN#_H
        }

        TxPacket[pDoc->m_RT_TABLE[pDoc->m_SpkPos][1]*2+3] = (unsigned char)(NOP & 0xFF); // NOP_L
```

```
TxPacket[pDoc->m_RT_TABLE[pDoc->m_SpkPos][1]*2+4] = (unsigned char)(((NOP & 0xFFFF) >> 8) & 0xFF); // NOP_H

for(NOB=0; NOB<100; NOB++) // PCM DATA 100 바이트
{
	TxPacket[pDoc->m_RT_TABLE[pDoc->m_SpkPos][1]*2+5 + NOB] = *pPcm;
	pPcm++;
}

// 타임아웃 설정
pDoc->m_Timeout = FALSE;
SetTimer(*pView, 1, 10000, NULL); // 무응답 타임아웃 설정 : 10초

// 시리얼포트 송신
pDoc->m_pSerial.Output(&TxPacket[0], TxPacket[0] + 1);

/*****************************************************************
 시리얼포트를 통해 서버노드로부터 응답 패킷 수신
*****************************************************************/

while (!pDoc->m_RxComplete) // 서버노드로부터 응답 대기
{
	if(pDoc->m_Timeout == TRUE)
	{
		pDoc->m_RX_DATA[0] = 5; // 음성 데이터 갱신 결과를 보고
		pDoc->m_RX_DATA[1] = 1; // 통신 오류(무응답)
		break;
	}
}

// 서버노드로부터 응답 패킷 수신함! 타이머 해제
KillTimer(*pView, 1);

pDoc->m_RxComplete = 0; // 초기화

/*****************************************************************
```

```
		 응답 패킷 분석 및 처리
		************************************************************/

		if(pDoc->m_RX_DATA[0] == 5) // 음성 데이터 갱신 결과를 보고
		{
			switch(pDoc->m_RX_DATA[1]) // 오류 코드
			{
				case 0: // 정상! for문 계속 수행..
				{
					break;
				}
				case 1: // 통신 오류(무응답)!
				{
					CString str1;
					str1.Format("case 1:통신 오류(무응답)!");
					AfxMessageBox(str1);
					NOP = 9999; // for문 강제 탈출 유도
					break;
				}
				case 2: // 통신 오류(패킷 오류)!
				{
					CString str1;
					str1.Format("%x case 2:통신 오류(패킷 오류)!",pDoc->m_RX_DATA[2] | (pDoc->m_RX_DATA[3]
<< 8) );
					AfxMessageBox(str1);
					NOP = 9999; // for문 강제 탈출 유도
					break;
				}
				default:
				{
					CString str1;
					str1.Format("%x default:통신 오류(패킷 오류)!",pDoc->m_RX_DATA[2] | (pDoc->m_RX_DATA[3]
<< 8) );
					AfxMessageBox(str1);
					NOP = 9999; // for문 강제 탈출 유도
```

```
                                        break;
                                    }
                                }
                            }
                            else // 통신 오류(패킷 오류)!
                            {
                                AfxMessageBox("통신 오류(서버노드 패킷 오류)!");
                                NOP = 9999; // for문 강제 탈출 유도
                            }

                            // 전송 시간 동안 Progress 컨트롤 작동
                            pView->m_transmit.SetPos(NOP);

        } // for(NOP=0; NOP<800; NOP++)

        if (NOP == 800) AfxMessageBox("음성 전송 완료!");

        /*****************************************************************
         쓰레드 종료
        *****************************************************************/

        // 전송 Progress 컨트롤 초기화
        pView->m_transmit.SetPos(0);
        pView->GetDlgItem(IDC_EDIT3)->SetWindowText((CString)"대기");

        delete pWav; // PCM 데이터 버퍼 해제
        pDoc->m_pSerial.bufread.Rstemp(); // 시리얼포트 수신 버퍼 클리어
        pDoc->m_RxComplete = 0;
        pDoc->m_FnInuse = 0;
        pDoc->m_PcBusy = 0;

        return 0;
}
```

[Step.9] 타임아웃 타이머 메시지 핸들러 제작

• **ClassWizard**를 열고, **Class name**에 **CPCView**, **Object IDs**에 **CPCView**, **Messages**에 **WM_TIMER**를 선택하고 **Add Function...** 버튼을 클릭, 팝업 창에 **OK**를 클릭하고 **Edit Code** 버튼을 클릭하면 명령 핸들러인 **OnTimer( )**가 만들어진다. **OnTimer( )** 안에 다음의 소스를 코딩해 넣는다.

```
void CPCView::OnTimer(UINT nIDEvent)
{
        // TODO: Add your message handler code here and/or call default

        CPCDoc *pDoc = GetDocument();

        pDoc->m_Timeout = TRUE;

        CFormView::OnTimer(nIDEvent);
}
```

[Step.10] 음성 녹음 기능 확인

지금까지의 작업을 저장하고 **Rebuild All**을 수행하여 **Error**와 **Warning**이 없는 것을 확인후 프로그램을 실행해본다.
메인컴퓨터에 마이크를 장착하고 **음성전송/녹음** 메뉴를 클릭하여 음성 녹음 동작을 확인한다.

[Step.11] 음성 재생 기능 확인

메인컴퓨터에 스피커를 장착하고 **음성전송/재생** 메뉴를 클릭하여 음성 재생 동작을 확인한다.

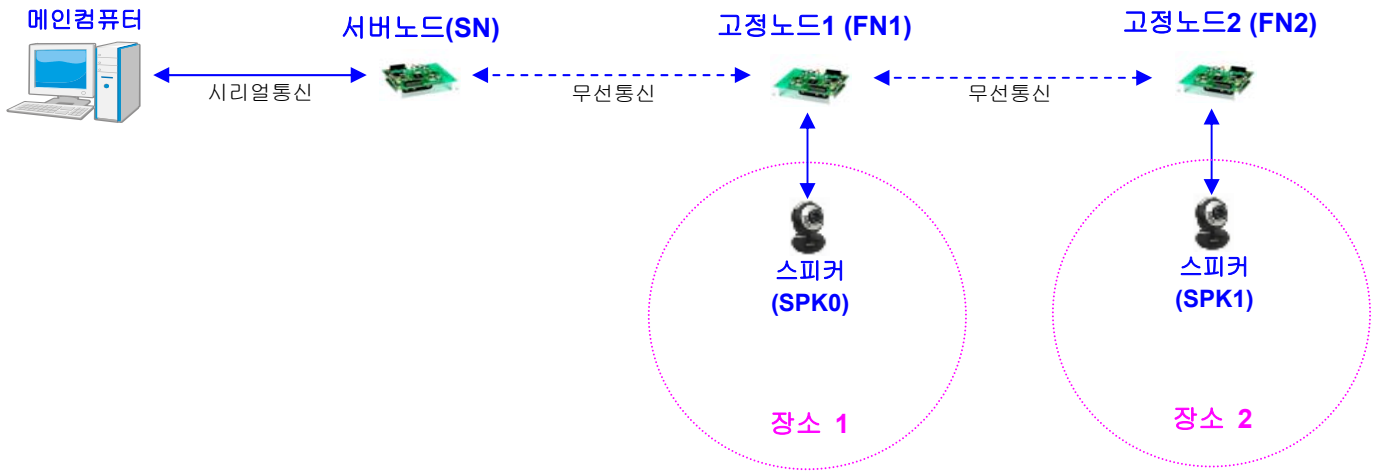
[Step.12] 음성 전송 기능 확인

서버노드 및 스피커 도터보드를 장착한 고정노드를 가동하면서 **음성전송/전송** 메뉴를 클릭하여 음성 전송 동작을 확인하고, 전송된 음성이 스피커에서 정상적으로 출력되는지 확인한다.

8. 무선 음성전송 예제 시스템 종합 실습

7절에서 제작한 고정노드의 펌웨어 프로그램과 메인컴퓨터 응용프로그램을 사용하여 무선 음성 전송 예제 시스템을 구축하고, 메인컴퓨터에서 전송한 음성 데이터가 고정노드의 스피커에 출력되는 과정을 실습해보도록 한다.

8-1. 시스템 구성



[예제 프로젝트의 시스템구성도]

8-2. 실습 절차

[Step.1] 메인컴퓨터 응용프로그램 실행

① 적당한 장소에 메인컴퓨터를 위치시키고 전원을 켠다.
② 메인컴퓨터 응용프로그램을 실행한다.

[Step.2] 서버노드 설치

① 서버노드에 해당 펌웨어를 다운로드한다.
② 적당한 장소에 설치한 후, 메인컴퓨터와 통신케이블을 접속하고 전원을 켠다.
③ 메인컴퓨터의 장치관리자에서 통신케이블이 접속된 COM포트 번호를 확인한다.

[Step.3] 고정노드1 설치

① 고정노드1에 해당 펌웨어를 다운로드하고, 스피커 모듈을 장착한다. (SPK0)
② 서버노드와 무선통신이 단절되지 않는 적당한 장소에 설치한 후 전원을 켠다.

[Step.4] 고정노드2 설치

① 고정노드2에 해당 펌웨어를 다운로드하고, 스피커 모듈을 장착한다. (SPK 1)
② 고정노드1과 무선통신이 단절되지 않는 적당한 장소에 설치한 후 전원을 켠다.

[Step.5] 음성 녹음 및 재생 실행

① 메인컴퓨터에 마이크와 스피커를 장착한다.
② **음성전송** 메뉴에서 **녹음** 부메뉴를 클릭하고 음성을 녹음한다.
③ **음성전송** 메뉴에서 **재생** 부메뉴를 클릭하고 녹음된 음성이 정상적으로 출력되는지 확인하다.

[Step.6] 음성 전송 실행

① 메인컴퓨터 응용프로그램의 **통신설정** 메뉴에서 시리얼포트를 설정한다.
② **위치설정** 메뉴에서 음성 전송을 원하는 위치의 스피커를 설정한다.
③ **음성전송** 메뉴의 **전송** 부메뉴를 클릭하여 해당 고정노드에 음성 데이터의 전송이 완료되고 스피커에 음성이 정상적으로 출력되는지 확인하다.

감사합니다

(제품문의, 기술지원)
OL마이크로웨이브 Co.
●홈페이지 http://www.olmicrowaves.com
●E-mail webmaster@olmicrowaves.com , imaman@hitel.net