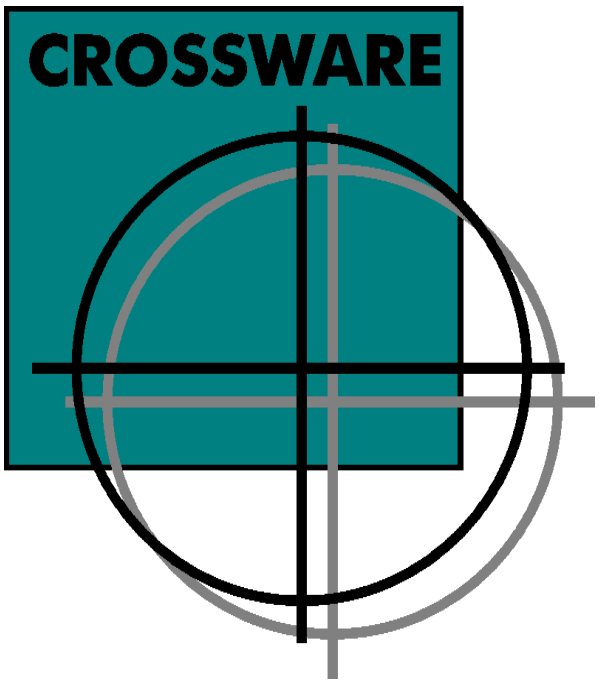
8051

C COMPILER



Crossware Products

Old Post House, Silver Street,

Litlington, Royston, Herts,

SG8 0QE, United Kingdom

Tel: + 44 (0) 1763 853500

Fax: + 44 (0) 1763 853330

HTTP://WWW.CROSSWARE.COM

CROSSWARE EMBEDDED DEVELOPMENT SUITE

8051 ANSI C COMPILER USER GUIDE

INTRODUCTION

본 매뉴얼은 Windows 95/NT 4.0(C8051N)에서 동작하는 Crossware 8051 C 패키지에 대하여 설명합니다. C8051NT 는 두 가지의 형태로 제공됩니다.

- Embedded Development Studio 가 포함된 완전한 풀 패키지
- Embedded Development Studio 가 설치 되어있을 때 추가모듈

C8051NT 는 다음과 같이 구성되어 있습니다.

- Embedded Development Studio (풀패키지로 구입한 경우)
- ANSI C 컴파일러와 실행 라이브러리
- 리로케이터블(relocatable) 크로스 어셈블러
- 링커
- 라이브러리 매니져
- 프로그램 관리 유틸리티

이 매뉴얼은 C 컴파일러와 런타임 라이브러리, 크로스 어셈블러, 링커 그리고 마이크로프로세서(8051)의 명령어에 관하여 설명합니다. 라이브러리 매니져와 프로그램 메인터넌스 유틸리티는 모든 Crossware C 컴파일러에서 공통입니다. 그리고 어셈블러는 별도의 매뉴얼에서 설명합니다. Embedded Development Studio 도 별도의 매뉴얼에서 설명합니다.

WHAT'S NEW

New with Compiler version 3.00

Tiny 메모리 모델과 Small 메모리 모델에서 /AD 옵션을 지정하면 컴파일러는 Direct 어드레싱 모드를 사용합니다.

New with Linker version 3.13

링커는 코드뱅크 스위치 프로세스의 최대 효율을 얻기 위하여 최적화된 뱅크 코드 스위칭 코드를 생성합니다. 자세한 것은 링커 최적화를 참고하시기 바랍니다.

New with Compiler version 2.77

메모리상에서 절대 어드레스에 변수를 배치하기 위한 _at 키워드를 추가 하였습니다. 자세한것은 Absolute Object Location 을 참고하기 바랍니다.

KEIL 과 **TASKING** 사의 8051 C 컴파일러의 8051 extension 을 지원하며 인식합니다. 자세한것은 **Support of Keil extension** 과 **Support of Tasking extension** 지원을 참고하기 바랍니다.

New with Compiler version 2.73

컴파일러는 개선된 인터럽트 루틴 코드를 생성합니다. Call Tree 분석기를 사용하면, 모든 프로그램에 대

하여 선택적으로, 각각의 인터럽트 함수 진입시 스택에 저장됩니다. 자세한것은 Call Tree Analysis 를 참고하기 바랍니다.
C 소스코드안에 _asm 신택스를 사용하여 어셈블리 코드를 삽입할수 있습니다. 이 신택스를 사용하면 어셈블리 코드로부터 쉽게 C 변수를 구동합니다. 자세한 것은 In Line Assembler Code 를 참조하기 바랍니다.
Code bank switching 을 C 에서 지원합니다. ROM 과 RAM 뱅크의 스위칭을 어셈블러에서 지원합니다. 자세한 것은 Code Bank Switching 과 Bankable Segments 를 참고하기 바랍니다.
컴파일러는 인터럽트 함수에 대하여 선택적으로 벡터 코드를 발생합니다. 자세한것은 Automatic Coding of Interrupt Vector 를 참고하기 바랍니다.

C COMPILER

Running the Compiler from the Command Line

C 컴파일러는 커맨드 라인에서 커맨드에 의하여 동작합니다.

CL [sfiles] [/options]

여기서 [sfiles]는 컴파일되는 소스의 리스트(스페이스에 의하여 구분)와 [/option]은 컴파일러, 크로스 어셈블러 그리고 링커의 옵션 리스트(스페이스에 의하여 구분)입니다.
커맨드 라인에서 파일 이름과 옵션은 순서에 관계없습니다. 그러나 첫번째 파일 이름은 프로그램 코드를 만들기 위한 파일 이름이어야 합니다.

C 소스파일 이름은 확장자를 가지고 있어야 합니다. 그렇지 않으면 링커에 포함된 object 파일로 간주되며 컴파일되지 않습니다. 파일의 확장자가 .ASM 이면 어셈블리 코드로 간주되며 크로스 어셈블러가 어셈블링하기 위하여 파일을 엽니다. 각각의 소스파일은 컴파일되며(어셈블리 코드는 어셈블링) 리로케이터블 오브젝트 모듈을 생성합니다. 이 오브젝트 파일은 소스파일 이름과 동일하게 만들어지며 확장자는 .OBJ 가 됩니다.

옵션 /c 를 사용하지 않으면 오브젝트 파일은 최종 파일을 만듭니다(디폴트 INTEL HEX 파일). 디폴트 라이브러리 파일은 자동적으로 연결되지 않은 참조항목을 찾습니다. 부동소숫점 연산(float, double 또는 long double)이 사용된 경우에는 부동소숫점 라이브러리가 자동으로 참조됩니다. 디폴트 라이브러리 파일은 지정한 메모리 모델과 non-reentrant 코드와 reentrant 코드 선택에 의하여 결정됩니다.

타겟 파일 이름으로 소스 파일이름과 동일한 것이 사용됩니다. 그러나 파일 출력 포맷에따라서 확장자는 다릅니다. (.HEX, .OMF, .S19, .695, .ASC 또는 .BIN). 링크 맵은 처음의 소스파일 이름이 사용되며 확장자는 .MAP 이됩니다. 컴파일러 옵션은 다음 섹션에서 설명됩니다. 크로스 에셈블러에 대한 옵션과 링커 옵션은 각각의 설명 부분을 참조하기 바랍니다. 다음은 커맨드 라인의 예를 보여 줍니다:

CL MAIN.C CALS.C IO.ASM TIMER /Fc /NORM /DDEBUG

이 커맨드 라인은 MAIN.C 와 CALS.C 를 컴파일하여 오브젝트 파일 MAIN.OBJ 과 CALS.OBJ 를 만듭니다. 파일 IO.ASM 을 어셈블링하여 IO.OBJ 를 만듭니다. 그리고 링커는 MAIN.OBJ, CALS.OBJ 그리고

TIMER.OBJ 를 링크하여 MAIN.HEX 와 링크맵 파일 MAIN.MAP 을 생성합니다. /Fc 는 컴파일러 옵션이며 MAIN.LST 와 CALS.LST 어셈블러 코드 리스팅이 출력하도록 컴파일러에게 지시합니다.(크로스 어셈블러는/Fc 옵션에 관계없이 IO.LST 를 생성합니다.)

/NOROM 은 링커 옵션이며 프로그램 코드 메모리와 외부데이터 메모리 영역이 동일하게 사용되는 코드를 출력합니다.

/DDEBUG 는 DEBUG 가 정의되는 것을 나타냅니다.

아무런 메모리 모델 지정이 없으므로 컴파일러는 Large 메모리 모델을 적용합니다. non-ertrant/reentrant 의 지정이 없었으므로 non-reentrant 모드 코드가 만들어 집니다.

Compiler Command Line Option

Compile Only (/c)

/c 옵션은 링크과정을 생략하고 컴파일만 실행합니다. 리로케이터블 오브젝트 파일을 작성하면 컴파일은 종료됩니다.

Generate Assembler Code (/Fa)

옵션 /Fa 는 컴파일러에게 리로케이터블 오브젝트 코드를 생성하지 말고 어셈블리 코드를 생성하도록 지시합니다. 어셈블러 코드의 파일이름은 소스파일 이름과 동일합니다. 그러나 확장자가 .ASM 이 됩니다. 최종 프로그램 생성시 연속적으로 어셈블리 코드를 출력하지 않도록 합니다. 이렇게 하면 링커의 크로스 모듈 통합 기능의 장점을 잃게될 것입니다.

Generate Assembler Code Listing (/Fc)

옵션 /Fc 는 컴파일러에게 리로케이터블 오브젝트 코드 파일에 주석문이 부가된 어셈블리/오브젝트 코드 리스트를 추가하여 출력할 것을 지시합니다.

Generate Debug Information (/DEBUG)

옵션 /DEBUG 는 디버그 정보를 오브젝트 파일에 첨가하여 만들도록 지시합니다. 링커는 이 정보를 모든 프로그램에서 디버그 정보를 포함하는 IEEE695 포맷(binary) 출력 파일을 만듭니다.

Define Identifier (/D)

옵션 /D[identifier] 는 정의된 오브젝트로서 identifier 를 만듭니다. 연속된 참조는 선행실행 지시자 #ifdef 또는 #if defined() 에서 반환값을 TRUE 로 합니다.

Warning as Errors (/WX)

이 옵션을 사용하면 컴파일러에게 warning 도 error 로 카운트 되게 합니다. 이 옵션은 링커나 라이브러리 매니져에게 warning 이 있으면 최종적인 파일이 만들어지는 것을 금지하려 할 때 사용합니다.

Warning Level 0 (/W0)

모든 waning 을 금지합니다.

Warning Level 1 (/W0)

현재 level 1 에대한 것은 없습니다. 이 옵션을 사용하면 /W0 와 동등하게 작용합니다.

Warning Level 2 (/W0)

이 옵션을 사용하면 참조되지 않는 심볼과 초기화되지 않은 상수변수 그리고 디폴트로 smart pointer 가 사용된 것에 대한 warning 을 금지 합니다.

Warning Level 3 (/W0)

이 옵션을 사용하면 초기화되지 않은 상수변수 그리고 디폴트로 smart pointer 가 사용된 것에 대한 warning 을 금지 합니다.

Warning Level 4 (/W0)

모든 컴파일러 warning 을 허용합니다.

Preserve Iram (/Oi)

이 옵션은 단지 Large 메모리 모델의 reentrant 함수에서만 적용됩니다. Large memory 모델의 reentrant 함수를 실행중에 함수가 호출되면 8051 내부 스택에 프레임 포인터를 확보합니다. 그리고 함수가 호출된 후에 또는 재 게산된후에 반환합니다. 디폴트로 컴파일러는 8051 내부 스택을 확보하여 빠르고 컴팩트한 코드를 만듭니다. 그러나 이것은 귀중한 내부램을 2 바이트 점유합니다. /Oi 옵션은 컴파일러에게 프레임 포인터를 다시 계산하고 내부램에 저장하지 말것을 지시합니다.

Default to Reentrant Code (/Os)

옵션 /Os 는 컴파일러에게 reentrant 코드를 디폴트로 만들것을 지시합니다. 그러므로 함수이름에 _nonreentrant 키워드가 사용되지 않는한 reentrant 함수로 컴파일합니다.

Favour Faste Code (/Of)

컴파일러는 프로그래머의 C 소스코드를 표시하기 위하여 in-line 8051 명령을 발생합니다. 이것은 최소한의 pre-written 서브루틴의 호출을 최소화하여 사용합니다. 매번 2 개의 long 정수가 더해져서 연산 동작을 위한 완전한 코드가 만들어 집니다. 코드 모듈이 완성되면 한번 이상 일치되는 코드가 있는지 탐색합니다. 반복되는 순서가 대략 6 바이트 의 코드이면 추출하여 별도의 서브루틴으로 만듭니다. 이러한 방법은 매우 코드사이즈를 줄이는 방법이 됩니다. /Of 옵션은 컴파일러에게 이러한 머지기능을 중지하도록 지시합니다. 최대의 실행소도를 를 요구하며 코드 사이즈가 중요하지 않을때 사용합니다. 또한 컴파일러에서 생성된 어셈블리 코드를 확인하려 할 때에도 이 옵션을 사용합니다. 머징 과정을 거친 코드를 논리적으로 따라가는것은 쉽지 않습니다. 최적화 pragma 는 이 옵션을 가능하게 할 때 사용가능합니다. 이것은 특별한 함수에서 머징 과정을 실행하지 않도록 합니다.

Use Register Keywords (/Or)

레지스터 키워드가 사용되었더라도 컴파일러에게 어떠한 레지스터 변수를 배치하지 말것을 지시합니다. 옵션 /Or 은 컴파일러에게 레지스터 변수를 구성하려할때 레지스터 키워드를 identifier 로 처리하도록 지시합니다. identifirer 가 컴파일러의 레지스터 배치 조건에 적합하면 identifier 는 레지스터 변수로 설정됩니다.

Global Register Allocation (/Og)

이 옵션은 컴파일러에게 전역 레지스터 배치를 허용 하도록 합니다.

Suppress Integral Promotion (/Op)

옵션 /Op 는 컴파일러에게 정수확장을 하지 않도록 지시합니다.

일반적으로 정밀도를 잃지 않는 범위 내에서 컴파일러는 모든 항목의 식에서 값을 포함하기 이전에 int 보다 작은 크기가 되도록 합니다. (이것은 ANSI 표준 요구사항입니다.) 이러한 변환이 되지 않도록 하면 컴파일러는 좀더 효율적인 코드를 만듭니다.

그러나 수식에 포함된 항목의 값에 따라 정수확장을 제한하면 정확도를 잃어버리는 경우가 있습니다. 다음의 식을 예로들면:

unsigned char a, b, c, d;

```
func()
{
    a = (b + c)/d;
}
```

만약 b 와 c 의 합이 255 보다 크면 정수확장을 실시하지 않으면 상위 바이트는 분실됩니다. 만약 d 가 1 보다 크면 결과 값 a 는 정수확장을 하지 않을 때와 다른 결과가 될 것입니다. 그러므로 위의 예에서 a 는 정수가 되어야 b , c 의 값에 관계없이 정확성을 잃지않습니다. 정수확장은 optimize pragma 에서 지역적으로 가능상태, 불능상태로 지정하는 것이 가능합니다.

Large Memory Model (/AL)

이 옵션은 컴파일러에게 Large 메모리 모델을 적용할 것을 컴파일러에게 지시합니다. 모든 object 는 다음의 경우를 제외하고 외부 메모리에 두게 됩니다.

- 상수로 지정된 개체는 코드 메모리 영역에 배치합니다.
- 스트링 문자는 코드 메모리 영역에 배치합니다.
- _bit 개체는 내부 램의 비트메모리 영역에 배치합니다.
- 메모리 영역이 지정된 개체는 지정된 메모리 영역에 배치합니다.

컴파일러는 내정 상태로 Large 메모리 모델을 사용합니다. 그러므로 Large 메모리 모델 코드를 만들때에 이 옵션을 지정할 필요는 없습니다.

Small Memory Model (/AS)

이 옵션은 컴파일러에게 Small 메모리 모델을 사용할것을 지시합니다.

만약 /AD 옵션이 사용되지 않았다면 모든 개체는 다음의 경우를 제외하고 간접 지정 데이터 메모리에 배치됩니다.

- 개체가 상수 값일경우 코드 메모리 영역에 선언됩니다.
- 스트링 문자는 코드 메모리 영역에 배치됩니다.
- _bit 개체는 내부 램의 비트메모리 영역에 배치합니다.
- 메모리 영역이 지정된 개체는 지정된 메모리 영역에 배치합니다.

만약 /AD 옵션이 사용된다면 위의 예외사항에 관계없이 모든 개체는 직접지정 어드레싱 메모리 영역에 배치됩니다.

Small Memory Model Direct Addressing (/AD)

이 옵션은 /AS 옵션이 사용되지 않으면 무시됩니다.

/AD 옵션은 컴파일러에게 모든 개체가 idata 또는 data 영역에 있도록 합니다.

이 옵션은 내부 RAM 이 7F hex 를 넘지 않는 small 메모리 모델에서 사용합니다. 이것은 @R0 또는 @R1 어드레싱 모드에 비하여 직접 개체를 지정하므로 좀더 효율적입니다. 내부 RAM 이 7F hex 를 넘을때에도 이 옵션을 사용하는것이 가능합니다. 모든 변수, 스태틱 프레임등이 7F hex 메모리 상위에 제공됩니다. 스택은 상위 방향으로 동작하므로 80hex 로 남습니다.

Use Acall/Ajmp

이 옵션은 컴파일러에게 CALL 과 JMP 명령 대신에 ACALL, AJMP 명령을 사용하도록 지시합니다. Tiny 메모리 모델에서는 디폴트 이므로 /AT 가 지정된 경우에는 이 옵션을 사용할 필요가 없습니다. Embedded Development Studio 는 프로그램 어드레스 영역이 0800hex 를 초과하지 않으면 자동적으로 이

옵션을 적용합니다.

Tiny Memory Model (/AT)

이 옵션은 컴파일러에게 Tiny 메모리 모델을 사용할 것을 지시합니다. 모든 개체는 다음의 경우를 제외하고 내부 메모리에 배치됩니다.

- 개체가 상수 값일경우 코드 메모리 영역에 선언됩니다.
- 스트링 문자는 코드 메모리 영역에 배치됩니다.
- _bit 개체는 내부 램의 비트메모리 영역에 배치합니다.
- 메모리 영역이 지정된 개체는 지정된 메모리 영역에 배치합니다.

_xdata 메모리 영역 지시자는 금지됩니다.(어떤 개체이든 외부 데이터 메모리에 배치할 수 없습니다.)그리고 모든 점프와 콜 명령은 AJMP 와 ACALL 이 사용됩니다. 단지 2 K 바이트의 코드가 지원됩니다. 프로그래머는 이 링커 옵션에 대하여 정확히 이해하고 있어야 합니다. 그렇지 않으면 부정확한 코드가 만들어 집니다.

Philips 사의 8XC750, 8XC748, 8X751, 8XC749 그리고 8XC752 는 LCALL, LJMP 또는 MOX 명령을 지원하지 않으므로 tiny 메모리 모델로 지정하여야 합니다.

Interrupt Vector Offset (/VO:)

이 옵션은 컴파일러에게 지정된 양에의한 인터럽트 벡터 코드를 사용하여 벡터 어드레스를 변경할것을 지시합니다. 예를들면:

/VO:8000

는 컴파일러에게 8000h 를 더할것을 지시합니다.

자세한것은 Automatic Coding of Interrupt Vector 를 참고하시기 바랍니다.

Embedded Development Studio 에서 작업하는 경우에는 컴파일러 세팅 탭에서 벡터옵셋을 지정할수 있습니다. (Build -> Setting -> Compiler)

Disable Keil Keywords (/Kk)

이 옵션을 사용하면 KEIL 8051 C 컴파일러와 호환성을 유지하기 위하여 추가된 키워드를 금지상태로 설정합니다. 자세한것은 Keil Extensions for details of the additional keywords 를 참고합니다.

Disable Tasking Keywords (/Kt)

이 옵션을 사용하면 Tasking 8051 C 컴파일러와 호환성을 유지하기 위하여 추가된 키워드를 금지상태로 설정합니다. 자세한것은 Tasking Extensions for details of the additional keywords 를 참고합니다.

C Language Definition

ANSI features

컴파일러는 모든 ANSI C 규정을 지원합니다. 라이브러리는 ANSI 표준 라이브러리의 형식을 갖습니다.

General Language Extensions

컴파일러가 지원하는 일반적인 언어 확장입니다.

- 변수의 길이 제한은 없으며 유효한 모든 문자를 사용 가능합니다.
- C++의 주석문 기호 "//"를 사용할 수 있습니다.
- _interrupt 키워드를 함수에 사용하면 인터럽트 루틴으로 처리됩니다.

- _persist 키워드는 변수가 파워 다운시에도 보존되도록 선언합니다.(배터리 보존 RAM) 즉 초기화 하지 않습니다.
- #asm 과 #endasm 선행처리기를 사용하여 C 소스코드에서 어셈블러 코드를 사용하도록 합니다.

8051 Specific Language Extensions

컴파일러는 마이크로 컨트로러의 내부구조를 최대한 이용하기 위하여 8051 의 여러 사양을 지원합니다.

- 메모리 영역 지시어는 선택한 메모리 모델과 상관 없이 bit, data, idata, xdata 그리고 code 메모리 영역에 특별히 지정합니다.
- 비트 변수를 직접 구동하기 위하여 bdata 영역에 배치합니다.
- C 변수는 8 비트 sfr, 16 비트 sfr, 그리고 sfr 비트를 직접 구동합니다.
- 인터럽트 벡터를 지원하며 인터럽트 함수의 레지스터 뱅크를 선택할 수 있습니다.

crossware 8051 C 컴파일러는 다음과 같은 데이터 크기를 지원합니다. Signed 는 디폴트로 되어있지만 모든 데이터 유형에서 signed 키워드를 사용하여 정의할 것입니다.
(1 바이트는 8 비트 입니다)

Type	**Storage Size**
Char	1 byte
short int	2 bytes
long int	4 bytes
int	2 bytes
unsigned char	1 bytes
unsigned short int	2 bytes
unsigned long int	4 bytes
unsigned int	2 bytes
float	4 bytes
double	8 bytes
long double	8 bytes
enum	Up to 4 bytes
bit fields	Up to 32 bits
bit	1 bit

Integer 는 LSB 가 먼저 저장됩니다. 부동 소숫점 데이터는 IEEE754 포맷으로 저장됩니다.

Pragmas

#pragma version

#pragma version

이 프라그마는 C 컴파일러의 이름과 버젼 번호를 컴파일러 리스팅에 추가합니다.

#pragma date

이 프라그마는 컴파일러 리스팅에 현재 날짜와 시간을 주석문으로 삽입합니다.

#pragma time

이 X 프라마는 컴파일러 리스팅에 시간을 주석문으로 삽입합니다.

#pragma message

컴파일러는 PC 디스프레이(stdout)에 메시지를 즉시 출력합니다.
지시형식은 다음 예에서 설명합니다.
#pragma message("Compiling release version")

#pragma optimize

#pragma optimize("[optimization-list]",{on|off})
지정한 optimize 레벨을 함수 단위로 설정합니다. optimize 프라그마는 함수의 바깥에 기술하며 pragma 가 나타난 첫번째 함수에서 유효합니다.

g 전역 레지스터 배치 허용
r 레지스터 키워드 사용
f 빠른 코드 사용
i 내부 메모리 예약
p 정수확장 제한
s 함수의 중복 사용 가능형으로 기본설정

위의 기호는 /O 컴파일러 옵션에서 사용하는 문자와 동일합니다. 예를들면
#pragma optimize ("rg", on)

optimize pragma 에서 공백 스트링 (" ")을 사용하면 원래 설정된 디폴트 값으로 복구됩니다.
#pragma optimize ("", off)
#pragma optimize ("", on)

#pragma_bank

#pragma bank("functions",<number>,<type>,<address>,<value>,<mask>)
다음의 함수가 코드뱅크 번호<number>에 놓여지도록 지정합니다.
<number> 는 0 부터 255 입니다.
<type>는 메모리 뱅크에서 선택하는 라이트 동작의 유형을 나타내며 "port", "sfr" 또는 "xdata" 가 될것입니다.
<address> 는 메모리 뱅크를 스위치하기 위하여 쓰여지는 어드레스 입니다.
<value> 는 메모리 뱅크를 선택하기 위하여 쓰여지는 어드레스에 쓰여지는 값입니다.
<mask> 라이트 동작에서 사용되는 마스크 입니다. 만약 <type> 가 "xdata" 이면 <mask> 는 0XFF 입니다.
예를들면:
#pragma bank("functions", 1, "port", 0X90, 0X20, 0X0F)

#pragma bank("functions", -1)

앞선 뱅크 pragma 를 금지하며 뒤이은 함수는 스위칭 가능한 코드뱅크로 놓지 않습니다.
bank pragma 의 사용에 관한 자세한것은 Code Bank Switching 을 참고합니다.

8051 Specific Features

Reentrant/Non-reentrant Functions

Crossware C 컴파일러는 re-entrant 와 non-re-entrant 함수를 모두 지원합니다. Reentrant 함수란 ANSI C 함수와 완전한 호환성을 갖습니다. 그리고 함수간 재귀 호출을 허용합니다. reentrant 함수는 정상 프로그램 실행상태(인터럽트 실행중이 아님)이거나 하위레벨의 인터럽트 실행중에 인터럽트에 의한 재실행이 가능합니다. 파라메터나 지역변수 영역은 메모리 모델에 따라 내부의 간접 어드레싱 데이터 메모리나 외부 메모리로 스택을 통하여 동적으로 배열됩니다. 그러므로 이 스택 영역은 함수가 실행될 때 존재하며 reentrant 코드는 데이터 메모리를 효율적으로 관리합니다. 그러나 8051 의 명령어는 통상적인 스택동작에 특별히 적합하지는 않습니다.(8051 은 데이터 메모리 구동 시 index 어드레싱 모드를 지원하지 않습니다.) 고정적인 스택 배치 시스템은 속도와 프로그램 크기가 개선됩니다. 모든 함수는 파라메터와 지역변수를 저장하기 위하여 자기의 고정된 메모리를 갖습니다. 메모리 위치가 고정되어 있기 때문에 함수가 실행중에 다시 호출되어 재실행된다면 지역변수가 오버 라이트 될 것입니다. 이러한 함수는 재 사용 할 수 없으므로 non-reentrant 함수라고 합니다.
또 다른 non-reentrant 함수의 장점은 진정한 _bit 형식의 지역변수를 제공합니다. 만약 _bit 변수를 파라메터나 지역변수로 reentrant 함수에서 사용한다면 unsigned char 로 변환될 것입니다. 그러나 non-reentrant 함수에서는 _bit 타입은 비트 어드레스 지정 가능 영역에 저장됩니다.
컴파일러는 커맨드 라인 옵션에서 (또는 Embedded Development Studio 의 대화상자 옵션)내정상태로 reentrant 인지 non-reentrant 인지 선택합니다. 이것은 모든 함수에서 개별적으로 선택하지 않는 한 내정상태가 적용되는 것입니다. 키워드 _reentrant 또는 _nonreentrant 를 함수 이름 원쪽에 놓으면 내정 상태에 관계없이 지정됩니다. 이 키워드는 함수 프로토 타입이 다른 모듈로부터 호출될 때 정확한 것인지 확실히 하기 위하여 사용 가능합니다.
linker 는 non-reentrant 함수가 re-entrnat 로서 특별 함수로 선언되어 재 사용 하려할 때 경고를 내보냅니다. 이 경고는 함수가 recursive 로 묵여지는 것을 의미하지 않습니다.

특별히 라이브러리 함수가 re-entrant 이길 원하지만 (혼동됨이 없이 정상 코드와 인터럽트 코드로써 공동으로 사용) 디폴트가 non-reentrant 이면 re-entrant 라이브러리 함수로 하기 위하여 특별한 과정이 필요합니다. non-reentrant 프로그램이 아니면 re-entrant 버젼의 strcpy()을 사용하기 원한다고 말합니다. 예를 들어 reentrant 버젼의 strcpy()
라이브러리 헤더 파일 string.h 에서 함수형식 선언자를 수정합니다.

```
char * strcpy(char * string1, const char * string2);
```

을 아래와 같이 변경합니다.

```
char * _reentrant strcpy (char* string1, const char char * string2);
```

해당 메모리에서 reentrant 버젼의 strcpy()를 빼버려야 합니다. non-reentrant 라이브러리는 마지막 문자가 n 입니다. 그러므로 snall 메모리 모델의 non-reentrant 라이브러리는 slib51n.lib 입니다. small 메모리 모델의 reentrant 모델은 slib51.lib 입니다. slib51.lib 에서 strcpy()를 추출하는것은 커맨드 라인에서 다음과 같이 입력합니다.

lib *strcpy slib51.lib

(slib51.lib 는 현재 디렉토리 내에 있어야 하며 lib.exe 는 PATH 로 연결되어 있어야 합니다.
이것은 원래의 라이브러리를 변경하지 않기 위하여 strcpy.obj 를 만듭니다.

Memory Models Pointer Types and Smart Pointers

8051/8052 에서 메모리 영역은 code space, internal ram, external ram 그리고 특수기능 레지스터 sfr 로 분류됩니다.
다른 메모리 영역에서 가능한 어드레스는 다음 표와 같이 표시됩니다.

	Minimum address (hexadecimal)	Maximum address (hexadecimal)
code space	0	FFFF
internal ram	0	FF (8051 에서 7F)
external ram	0	FFFF
sfr space	80	FF

모든 내부 RAM 은 간접 어드레스 지정이 가능합니다. 그러나 0 부터 7F 는 직접 어드레싱도 가능합니다. 직접 어드레싱 방법으로 80 부터 FF 영역을 구동하면 sfr 영역이 됩니다. 내부 메모리의 특정 영역과 sfr 의 특정 위치들은 비트 어드레싱이 가능합니다.
Crossware 8051 ANSI C 컴파일러는 8051 의 모든 어드레스 영역을 C 에서 직접 구동하는 것이 가능합니다. C 변수의 선언 방법은 위에서 열거한 메모리 영역에 두거나 지정합니다. 8051 메모리 모델은 복잡하지만 C 프로그래머에게 small 과 large 2 개의 기본 메모리 모델을 제공합니다. tiny 모델은 특별히 8051 의 변형 버젼을 위하여 지원되는 것입니다. Tiny 모델의 프로그램 영역은 최대 2 K 바이트이며 외부 RAM 영역은 없습니다. small 메모리 모델에서 모든 C 변수는 내부 RAM 에 배치합니다. 상수로 정의된 것은 code 메모리에 배치됩니다. 함수의 파라메터와 지역변수는 내부 RAM 에 저장됩니다. Large 메모리 모델에서 모든 C 변수는 외부 메모리 영역에 저장합니다. 상수로 정의된 것은 code 영역에 저장됩니다. 함수의 파라메터와 지역변수는 외부 RAM 에 저장됩니다.

Small 메모리 모델이나 Large 메모리 모델에서 memory space qualifier 를 사용하여 선언된 것은 지정된 곳으로 배치합니다. 위의 테이블에서처럼 서로 다른 메모리 영역의 어드레스 겹쳐져 있습니다. 예를 들어 위치를 지시하는 포인터 2F(hex)가 code 영역, 내부 ram, 외부 ram 을 지시할 수 있습니다. 이 애매한 상황을 Crossware 8051 ANSI C 컴파일러는 개선된 포인터 구조를 제공합니다. 메모리 의존 포인터는 하나의 특정 메모리 영역-code, internal ram, 또는 external ram 만을 지시합니다. 내부 ram 용 포인터는 1 바이트로 구성됩니다. 코드영역과 외부 RAM 은 2 바이트로 됩니다. generic 포인터는 3 개의 메모리 영역 어느 곳이든 지시하는 것이 가능합니다. generinc 포인터는 3 개의 바이트로 구성됩니다. 2 바이트는 generic 포인터가 지시하는 실제의 어드레스 입니다. 3 번째 바이트는 지시하는 메모리 영역에 대한 정보

를 가지고 있습니다.

Generic 포인터는 strcpy() 와 같은 라이브러리 함수를 호출하는 기본적인 것입니다. 포인터 매개 변수는 코드 영역, 내부 ram, 외부 ram 일 것입니다. 그러나 genric pointer 는 상대적으로 비효율적입니다. 컴파일러는 generic 포인터의 세번 째 바이트를 해독하는 코드를 만듭니다. 그 값에 따라 적절한 메모리 영역으로 점프하는 명령어 군을 만듭니다. Crossware 8051 ANSI C 컴파일러는 generic 포인터를 피하기 위하여 큰 길이로 만들어져 있습니다. 프로그래머에게 가능한 서로 다른 메모리 모델에 따른 주의가 필요치 않도록 합니다. 프로그래머는 포인터 선언에 관한 특별한 주의 없이 ANSI C 프로그램을 작성합니다. 그리고 컴파일러는 사용되는 포인터 종류에 따라 지능적으로 판단합니다. 이러한 포인터를 smart 포인터라고 합니다. 포인터가 포인터를 그리고 정수를 지시할 때 사용된 방법에 따라 적절한 non-generic 포인터를 구성하는 것이 가능합니다.

하나의 파일에서 포인터를 정의하고 사용하며 extern 으로 하여 외부 파일에서 사용할 때 각각의 파일에서 다른 포인터 형식을 지정하면 다르게 선택됩니다.

이 문제는 극복하기 쉽습니다. Crossware Linker 는 프로그램의 모든 모듈에 대하여 편리한 광범위한 형식을 검사하며 프로그래머에게 주의하도록 합니다.

strcpy()라이브러리 함수와 같이 generic 포인터가 본질적인 것이지만 프로그래머는 여전히 ANSI 호환 코드를 작성할 수 있습니다. strcpy()함수는 strcpy(char*, const char*)로 선언됩니다. 컴파일러는 선언에서 const 를 사용하여 포인터가 코드영역 또는 ram 영역을 지시한것을 나타냅니다. 이렇게 하여 함수 인자를 generic 포인터로 변환합니다.

Smart 포인터는 프로그래머에게 주요 상황에서 다른 포인터 형식에 대하여 자세한 것은 무시합니다. spaced 포인터의 효율을 실현합니다. 프로그래머는 small 과 large 메모리 모델을 소스코드의 수정 없이 변경할 수 있습니다.

Memory Space Qualifiers

메모리 영역 지시자는 글로벌 개체의 특정 메모리 영역을 선택하거나 포인터가 특별 메모리 영역을 지시하도록 선언합니다. 아래의 표는 Crossware 8051 ANSI C 컴파일러서 지원하는 메모리 영역 지시자 입니다.

Memory space qualifier	**Associates memory space**
_code	program memory
_data	internal data memory
_idata	internal indirectly addressable data memory
_xdata	external data memory
_generic	all memory space

메모리 영역 지시자의 사용은 컴파일러에 의하여 사용되지 않으면 디폴트로 오버라이드 됩니다.

(키워드 _bit, _sbit, _sfr 그리고 _sfr16 은 메모리 영역 지시자로 사용되는 것은 아닙니다.)

메모리 영역 지시자는 항목의 왼쪽에 놓여집니다.

Qualifying a Object

개체 자신의 위치를 지정할 때 형식 지시자는 변수와 공백문자를 두고 기술합니다.

```
int _data nCount;
char * _xdata pszText;
```

struct tagCapture _idata sCapture

위의 예에서

- 2 바이트의 내부 데이터 메모리가 정수 nCount 로 예약됩니다.
- 1,2 또는 3 바이트의 외부 데이터 메모리가 pszText 포인터로 예약됩니다. (pszText 는 정해지지 않은 포인터이며 컴파일러는 포인터에 사용 방법에 따라 어떤 크기의 바이트가 필요한지 결정합니다.)
- 간접 어드레스 지정 데이터 메모리가 스트럭쳐 sCapture 로 예약됩니다. (스트럭쳐에 일치하는 바이트 수가 예약됩니다.)

단지 포인터의 내용이 generic 이 됩니다. 그리고 _generic 지시자는 개체 자신으로 사용할 수 없습니다. 지역 개체와 함수 파라메터는 언제나 형식이 지정되지 않은 디폴트 메모리 공간 위치에 배정됩니다.

Qualifying the Contents of a Pointer

포인터 내용의 형식 지시자는 * 심볼의 바로 원쪽에 놓습니다.

예를들면:

```
char _idata * pszText;

char _xdata * _idata * ppszMessage;
struct tagInfo {
      char _idata ** ppszName
      char**ppszAddress;
}
```

위의 예에서

- pszText 는 디폴트 메모리 영역에 배치하며 한 개의 바이트로 구성된 포인터가 간접 어드레싱 모드로 지정 가능한 내부 메모리 영역에 있는 한 개의 바이트 문자를 지시합니다.
- ppszMessage 는 디폴트 메모리 영역에 배치하며 한 개의 바이트 포인터(간접 어드레스 지정이 가능한 내부 ram)가 두개의 바이트로 구성된 포인터를 지시합니다. 이 포인터는 외부 메모리의 한 개의 문자를 지시합니다.
- ppszName 은 스트럭쳐 태그정보에 사용되는 개체의 메모리 영역에 있습니다. 1,2 또는 3 개의 바이트로 구성되며 한개의 바이트를 지시합니다. 이것은 간접 어드레스 지정 내부 데이터 메모리의 문자를 지시합니다.
- ppszAddress 는 ppszName 과 동일한 메모리 영역에 배치됩니다. tagInfo 구조에 사용된 개체에 따라 변경됩니다. 1 2 또는 3 개의 바이트가 한개의 바이트를 지시하는 1 , 2 또는 3 개 바이트를 지시합니다.

ppszName 과 ppszAddress 는 완전히 정의되지 않았습니다. 컴파일러는 선언의 미지적된 부분을 위하여 적절한 포인터 형식을 결정할 것입니다.

```
typedef struct {
        Char _idata** ppszName;
```

```
    Char** ppszAddress;
} INFO;
INFO Info1;
INFO info2;
```

포인터 Info1.ppszAddress 와 Info2.ppszAddress 는 다른 포인터 타입이 됩니다. 일반적으로 포인터 내용의 성격 규정은 필요치 않습니다. smart 포인터 기술은 컴파일러가 적절한 포인터 형식을 결정합니다.

Default Memory Space Usage

메모리 영역 지시자가 사용 되었더라도 Crossware 8051 ANSI C 컴파일러는 다음의 위치로 개체를 배치합니다.

- _bit 로 정의된 개체는 비트-어드레스 데이터 메모리에 배치합니다. reentrant 함수에서 자동 변수로 처리되며 unsigned char 로 변환됩니다.
- 스트링 문자열은 프로그램 메모리에 배치합니다.

기타:

- const 로 선언된 개체는 program memory 에 선언됩니다.

기타:

- small 또는 Tiny 메모리 모델이 디폴트일때 간접 어드레싱 가능한 내부 메모리에 개체가 배치됩니다.
- large 메모리 모델이 디폴트이면 개체는 외부 데이터 메모리에 저장됩니다.

Bit Objects

비트 데이터 타입은 _bit 키워드로 선언합니다 예를 들면:

```
_bit flag;
```

변수 flag 는 1 비트 크기이며 8051 의 비트 어드레스 메모리 영역에 배치됩니다. flag 는 0 또는 1 을 저장하게 됩니다.

비트 변수는 비트 메모리에 있는 비트 변수에 대하여 비트 오프셋에 의하여 선언될 수 있습니다. 자세한것은 비트 어드레스 개체 부분을 참조합니다.

비트 데이터 타입은 매우 효율 좋은 코드를 만듭니다. 예를들면:

```
if (flag)
{
}
```

단순히 jb _flag <label> 명령을 만듭니다.

비트 개체가 수식에서 사용되면 수식이 계산되기 전에 unsigned char 로 변환됩니다.

비트 데이터 변수는 언제나 전역 변수로 됩니다. 함수인자 제공하는 함수가 non-reentrant 일때 그리고 함수 포인터에 의하여 호출되지 않을 때 비트 변수는 지역변수나 함수의 인자로 사용 가능합니다. 만약 함수가 reentrant 이고 _bit 키워드가 사용되면 모든 지역 변수와 _bit 키워드에 의하여 선언된 함수 인자는 분명히 unsigned char 로 변환됩니다.

비트를 지시하는 포인터는 정의될수 없으며 비트 개체의 어드레스를 추출하기위하여 & 연산자도 사용할 수 없습니다.

Bit Addressable Objects

개체는 _bdata 키워드를 사용하여 비트 메모리 영역에 배치할 수 있습니다. 예를들면:

int _bdata nMask;

비트지정 가능 메모리내에 정수변수를 배치합니다.

비트 메모리 영역에 변수를 배치하는 것은 개체의 개별 비트를 구동하려는 유일한 목적이 있습니다. 이것은 비트 개체에 대하여 _sftbit 키워드를 사용하여 지정하며(이 경우 비트 가 sfr 영역에 있지 않더라도) '^' 연산자에 의하여 결합됩니다. 예를들면:

int _bdata nMask;

_sfrbit bit3 = nMask^3;

변수 bit3 은 변수 nMask 의 비트 3 을 구동하는 것에 사용된것입니다.

모든 전역적 개체는 비트 어드레싱 가능 메모리 영역에 배치 가능합니다. - int, float, double, structure, array 등 공간이 확보되는 한- 지역 개체와 함수 인자는 비트 어드레스 지정 가능 메모리 영역에 배치할 수 없습니다.

Generic Pointers

Generic 포인터는 어느 메모리 영역이든 지시합니다. 3 바이트 영역을 가지고 있습니다. 3 번째 바이트는 메모리 영역에 대한 정보를 가지고 있습니다. 3 번째 바이트는 1 에서 5 의 값을 가지고 있으며 다음과 같은 의미를 가지고 있습니다.

Value	Memory area
1	idata
2	xdata
3	reserved for future use
4	data (including bata)
5	code

이 값을 특별히 generic 포인터에 상수로 지정할 수 있습니다. 예를 들면 외부 데이터 메모리 영역 어드레스 8200(hex)에 선언과 지정은 다음과 같습니다.

```
char _generic *pGeneric = 0X28200;
```

generic 포인터는 여러 가지 방법으로 만듭니다.

- 위에서 설명한대로 _generic 키워드에 의하여 선언합니다.
- 컴파일러의 smart 포인터 기술에 의하여 만듭니다. 이 경우 제너릭 포인터가 형식 지정되지 않은 포인터를 지시할 때 형식 지정되지 않은 포인터는 또한 generic 포인터가 됩니다. 그러므로 다음과 같은 코드 예에서 pAnother 는 자동적으로 generic 포인터가 됩니다.

```
char _generic *pGeneric = 0x28200;
char *pAnother;
pAnother = pGeneric;
```

- 상수 포인터로써 함수의 인자를 정의하는 방법으로 만들 수 있습니다. 예를 들면 다음의 예에서 pszText 는 묵시적으로 generic 포인터가 됩니다.

```
int func(const char *pszText)
{
    int nCount = 0;
    while (*pszText != '\0')
            nCount++;
      return nCount;
}
```

포인터를 상수 값으로 하여 함수의 인자로 정의하는 방법은 generic 포인터를 만드는 가장 일반적인 방법이 되며 프로그래머가 이해하기 위하여 가장 중요한 방법이 됩니다. generic 포인터는 선택한 메모리 모델이 무엇이든 수정하지 않고 표준 C 라이브러리 함수의 기반이 됩니다. 또한 generic 포인터가 필요 없으면 비효율성을 피할 수 있습니다.

규정은 다음과 같습니다. 컴파일러가 함수인자를 보고 포인터 상수일 때 개체가 code 영역에 있을 가능성이 있는 것으로 가정합니다. 그러나 개체가 데이터 영역에 있을 가능성을 배재하지 않습니다. 따라서 generic 포인터를 만듭니다. 이것은 rom 이든 ram 이든 함수 인자를 const 가 사용된 모든 함수의 표준 C 라이브러리에서 정확하게 동작합니다.

예를들어 strcpy()는 다음과 같이 선언됩니다.

```
char strcpy(char pszDestination, const char* pszSource);
```

여기서 pszSource 는 스트링 문자열이 되며 자동적으로 코드 영역에 배치됩니다.

```
char    szBuffer[50];
strcpy(szBuffer, "Hello world");
```

또는 pszSource 는 ram 에 있게 됩니다.

```
char szBuffer[50];
char szText[ ] = "hello world";
strcpy(szBuffer, szText);
```

strcpy() 함수가 데이터를 라이트 하게 될 것이므로 pszDestination 는 데이터 영역에 있습니다. 만약 프로그래머가 자신의 인자를 갖는 함수에 라이트 하려할 때 generic 포인터가 필요합니다. _generic 키워드를 사용하여 선언하는 것이 가능하다면, 더욱 편리하며 포인터를 상수로 선언하는 것과 동일한 효과 입니다.

함수인자가 generic 포인터로부터 상수를 지시하는 함수 인자를 금지 하려면 적당한 형식 선언자를 사용할 필요가 있습니다. 예를 들어 pszMessage 가 code 영역만 지시하기 원할 때 함수를 다음과 같이 선언하여야 합니다.

```
void func(const char _code *pszMessage);
```

상수를 지시하는 포인터가 함수의 인수로될때 제너릭 포인터가 됩니다. 만약 프로그래머가 포인터를 상수로 선언한다면 그것은 smart 포인터 기술이 적용됩니다.

Absolute Object Location

C 변수는 _at 키워드에 의하여 절대위치 어드레스에 배치할 수 있습니다. 이것은 디버그 용도의 모니터링과 같은 다른 프로그램의 사용 영역을 간단히 예약하는 방법이 되며 또한 memory mapped I/O 를 구동하는 방법이 됩니다.

예를들면:

long _xdata variable _at 0X8000;
은 어드레스 0X8000 을 지시하는 것이며 4 개의 연속적인 바이트 0X8000, 0X8001, 0X8002, 0X8003 에 라이트하거나 리드합니다.

다른 예:

```
struct tag8255 {
        unsigned char PortA;
        unsigned char PortB
        unsigned char PortC;
        unsigned char I_O;
};
#define BASE8255    0xFF00
volatile struct tag8255 _xdata S8255 _at BASE8255;

func()
{
    S8255.PortA = 0X88;      //write to address 0XFF00
    S8255.PortC = 0X98;      // write to address 0XFF01
}
```

Compatibility Features

Support for Keil extensions

Crossware C 컴파일러는 다음의 Keil 키워드를 인식합니다.

Additional Keywords

키워드 bit, sbit, data, bdata, idata, xdata, sfr, sfr16, code, generic, using, reentrant, interrupt 그리고 _at_ 을 지원합니다. 이 키워드는 Crossware 키워드 _bit, _sftbit, _data, _bdata, _idata, _xdata, _sfr, _sfrword, _code, _generic, _using, _reentrant, _interrupt 그리고 _at 의 동일한 키워드로 변환됩니다.
위의서 열거한 부가적인 키워드는 디폴트 상태에서 인식됩니다. 그러나 커멘드 라인에서 /Kk 옵션을 사

용하면 금지됩니다. 또한 Embedded Development Studio 에서 Build->Setting->Compiler 메뉴에서 Allow Keil Keyword 를 선택하면 검사하지 않습니다.

Additional Formats

컴파일러는 다음의 포맷을 추가 지원합니다.

인터럽트에서 using 과 reentrant 키워드는 인터럽트 함수 이름 다음에 위치합니다. (using 키워드는 interrupt 키워드 다음에 와야 하며 예외는 없습니다.) 개체의 메모리 형식 선언자는 선언의 초기에 있어야 합니다. 예를 들면:

data char x 는 char data x 와 같습니다.

그리고

data char xdata* p 는 char xdata* data p 와 같습니다.Crossware 의 _nonreentrant 키워드를 사용할 수 있지만 디폴트가 reentrant 이므로 권장하지 않습니다. 모든 비트 파라메터는 컴파일러가 함수를 non-reentrant 로 만들기 전에 unsigned char 로 변환합니다. 개체정보가 처음에 비트로 설정된 것은 더이상 유효하지 않으며 컴파일러는 파라메터 리스트를 비트로 전환하기 위하여 파싱을 실시하지 않습니다.

Intrinsic Function

Crossware C 컴파일러는 Keil 의 고유함수를 지원합니다 (_chkfloat 제외)

이것은 intrins.h 에 마크로 또는 intrinic 함수로 정의되어 있습니다. _nop_() 과 _testbit_()은 마크로 이며 _cror_(), _iror_(), _crol_(), _irol_()과 _lrol_()은 컴파일러에서 hard 코딩된 intrinic 함수입니다. _testbit_() 을 마크로로 설정하여 비트가아닌 부울리안 플래그로 사용하는 정수로 사용할 수 있습니다. 그러므로 reentrant 함수에서 자동 bits 는 unsigned char 로 변환되어 _testbit_() 는 정확하게 동작합니다.

Support for Tasking extensions

Crossware C 컴파일러는 다음의 Tasking 확장을 지원합니다.

Additional Keywords

키워드 bit, _sfrbyte, data, _bdat, _iat, idat, _xdat, xdat, _rom 그리고 rom 을 지원합니다. 이 키워드는 Crossware 동일한 키워드 _bit, _sfr, _data, _bdata, _idata, _idata, _xdata, _xdata, _code 와 _code 로 각각 변환됩니다.

추가적으로 _atbit() 어트리뷰트를 지원합니다. 이것은 먼저 정의된 마크로 동격으로 존재합니다.

```
#define _atbit(a,b) = (a)^(b)
```

그리고 construct 는 자동적으로 Crossware 컴파일러에 의하여 변환되어 인식됩니다.

추가적인 키워드와 형식은 내정상태가 가능입니다. 그러나 명령라인에서 /Kt 옵션을 사용하거나 Embedded Development Studio 에서 Build->Setting->Compiler 탭에서 Allow Tasking keywords 옵션을 사용하여 검사하지 않도록 할 수 있습니다. 참고로 _atbit 는 #undef 지시자에 의하여 미 정의 상태로 할 수 있습니다.

Accessing the 8051 Internal Registers

SFR Bytes

_sfr 키워드는 sfr 데이터 영역의 8 비트폭의 위치를 지정하는 식별자로 선언할 때 사용합니다. 선언의 형식은 다음과 같습니다.

_sfr <identifier name> = <address>;

예를들어:

_sfr Inputport = 0XA0;

InputPort 는 sfr 영역의 어드레스 A0(hex)로 선언됩니다.

InputPort 에 값을 지정하면 어드레스 A0 에 라이트 합니다.

예를들어:

InputPort = 0X20;

은 어드레스 A0(hex) 에 20(hex)를 라이트 합니다.

InputPort 를 다른 변수에 지정하거나 또는 수식에서 사용하면 어드레스 A0 로부터 값을 읽습니다.

예를들어:

unsigned char nPortValue;

nPortValue = InputPort;

어드레스 A0 로 부터 읽어 변수 nPortValue 에 저장합니다.

변수와 함께 OR InputPort 하면 어드레스 A0 로부터 값을 읽어 변수와 Or 연산 후에 어드레스 A0 에 라이트 합니다.

예를들어:

unsugned char nPortValue = 0X32;

InputPort |= nPortValue;

어드레스 A0 로부터 값을 읽습니다. nPortValue 와 Or 연산하여 어드레스 A0 에 반환합니다.

SFR Words

_sfrword 키워드는 sfr 데이터 영역에서 16 비트 폭의 위치를 구동하는 식별자를 선언할때 사용합니다. 선언의 형식은 다음과 같습니다:

_sfrword<identifier name>=<address>;

예를들어:

_sfrword Counter = 0XCA;

는 Counter 가 sfr 어드레스 영역의 어드레스 CA 로 선언합니다.

Counter 에 값을 지정할 때 LSB 는 어드레스 CA 에 MSB 는 어드레스 CB 에 라이트 됩니다.

Counter 에 다른 값을 지정하거나 Counter 가 수식에 사용되면 LSB 는 CA 에서 MSB 는 CB 에서 읽습니다.

변수와 함께 OR Counter 를 사용하면 어드레스 CA 와 CB 로 부터 값을 읽어 변수와 OR 연산 후 CA 와 CB 로 반환합니다.

SFR Bits

_sfrbit 키워드는 1 비트 폭의 어드레스 지정 위치를 구동하기 위한 식별자를 선언하기 위하여 사용됩니다. (위치는 sfr 데이터일 필요는 없습니다 - non sfr 데이터 영역에서 어떻게 _sfrbit 를 사용하는가는 뒤에 오는 Bit Addressable Object 를 참고합니다.)

선언은 다음과 같은 형식입니다:

_sfrbit <identifier name> = < bit address >;

또는:

_sfrbit <identifier name> = <bit addressable identifier>^<bit offset>

예를들어:

_sfrbit Pin = 0X84;

는 비트 어드레스 0X84 를 Pin 으로 선언합니다.

Pin 에 데이터를 지정하면 비트 어드레스 84(hex) 에 비트값이 라이트 됩니다.

Pin 을 다른 변수로 지정하거나 수식에서 사용하면 비트 어드레스 84(hex)로 부터 값을 읽습니다.

변수 또는 상수와 Pin 을 OR 연산하면 비트 어드레스 84(hex)로부터 비트 값을 읽어 변수와 OR 연산 후 비트 어드레스 84(hex)로 반환합니다.

다른 예:

_sfr Port = 0X90;

_sfrbit Pin3 = Port^3;

Pin3 는 8 비트 폭 어드레스에 의하여 정의된 Port 의 4 번째 비트입니다. 이 경우 비트 어드레스는 93(hex) 입니다?

Predefined 8051 Internal Registers Variables

8051 의 모든 특수기능 레지스터(special Function Register)는 C 프로그램에서 _sfr, _sfrword 그리고 _sfrbit 변수형으로 구동됩니다.

SFR 정의는 2 개의 형식으로 됩니다.

- 언더스코어("_")가 앞에온 소문자
- 언더스코어("_") 가 없는 대문자

첫번째 형식은 ANSI 표준안(제조사에서 정의한 변수이름에는 언더 스코어가 필요)과 호환성을 유지하는 것입니다. 소문자를 사용하는 것은 대문자 기호는 전통적으로 C 마크로로 사용됩니다.

두번째 형식은 다른 C 컴파일러와의 호환성을 유지하기 위한 것입니다.

헤더 파일 sfr.h 은 첫번째 형식을 사용하며 reg.h 는 두번째 형식을 사용합니다.

sfr.h 의 내용은 아래와 같습니다. 다른 진보된 8051 의 추가적인 레지스터의 정의는 이 파일에 추가합니다.

```
/* SFR definitions are provided in two forms:
 *
 * - lower case with a leading underscore
 * - upper case without a leading underscore
 *
 * The first form uses a leading underscore to be compatible
 * with that ANSI standard (which requires a leading underscore
 * for vendor defined variables) and uses lower case because upper
 * case symbols are traditionally associated with C macros.
 *
 * The second form is provide compatible with other 8051 C compilers.
 *
 * Include sfr.h to use the first form.
 * Include reg.h to use the second form.
 */

#ifndef _SFR_H
#define _SFR_H

/**** 8051 8 bit SFR's *****/

_sfr _acc = 0XE0;/* accumulator */
_sfr _b = 0XF0;          /* accumulator B */
_sfr _dph = 0X83;/* data pointer high byte */
_sfr _dpl = 0X82;/* data pointer low byte */
_sfr _ie = 0XA8;         /* interrupt enable */
_sfr _ip = 0XB8;         /* interrupt priority */
_sfr _p0 = 0X80;         /* port 0 */
_sfr _p1 = 0X90;         /* port 1 */
_sfr _p2 = 0XA0;         /* port 2 */
_sfr _p3 = 0XB0;         /* port 3 */
_sfr _psw = 0XD0;/* program status word */
_sfr _sbuf = 0X99;       /* serial port buffer */
_sfr _scon = 0X98;       /* serial port controller */
_sfr _sp = 0X81;         /* stack pointer */
_sfr _tcon = 0X88;       /* timer control */
_sfr _th0 = 0X8C;/* timer 0 high byte */
_sfr _th1 = 0X8D;/* timer 1 high byte */
_sfr _tl0 = 0X8A;/* timer 0 low byte */
_sfr _tl1 = 0X8B;/* timer 1 low byte */
_sfr _tmod = 0X89;       /* timer mode */
_sfr _pcon = 0X87;       /* power control register */

/**** 8051 16 bit SFR's *****/

_sfrword _dptr = 0X82;   /* data pointer low byte */

/**** 8051 SFR bits *****/

_sfrbit _cy = _psw^7;    /* carry flag */
_sfrbit _ac = _psw^6;    /* auxiliary carry flag */
_sfrbit _f0 = _psw^5;    /* flag 0 */
_sfrbit _rs1 = _psw^4;   /* register bank select 1 */
```

```
_sfrbit _rs0 = _psw^3;    /* register bank select 0 */
_sfrbit _ov = _psw^2;     /* overflow flag */
_sfrbit _p = _psw^0;      /* parity flag */

_sfrbit _tf1 = _tcon^7;   /* timer 1 overflow flag */
_sfrbit _tr1 = _tcon^6;   /* timer 1 run control bit */
_sfrbit _tf0 = _tcon^5;   /* timer 0 overflow flag */
_sfrbit _tr0 = _tcon^4;   /* timer 0 run control bit */
_sfrbit _ie1 = _tcon^3;   /* interrupt 1 edge flag */
_sfrbit _it1 = _tcon^2;   /* interrupt 1 type control bit */
_sfrbit _ie0 = _tcon^1;   /* interrupt 0 edge flag */
_sfrbit _it0 = _tcon^0;   /* interrupt 0 type control bit */

_sfrbit _sm0 = _scon^7;   /* serial mode control bit 0 */
_sfrbit _sm1 = _scon^6;   /* serial mode control bit 1 */
_sfrbit _sm2 = _scon^5;   /* serial mode control bit 2 */
_sfrbit _ren = _scon^4;   /* receive enable */
_sfrbit _tb8 = _scon^3;   /* transmit bit 8 */
_sfrbit _rb8 = _scon^2;   /* receive bit 8 */
_sfrbit _ti = _scon^1;    /* transmit interrupt flag */
_sfrbit _ri = _scon^0;    /* receive interrupt flag */

_sfrbit _ea = _ie^7;              /* enable all interrupts */
_sfrbit _es = _ie^4;              /* enable serial port interrupt */
_sfrbit _et1 = _ie^3;     /* enable timer 1 interrupt */
_sfrbit _ex1 = _ie^2;     /* enable external interrupt 1 */
_sfrbit _et0 = _ie^1;     /* enable timer 0 interrupt */
_sfrbit _ex0 = _ie^0;     /* enable external interrupt 0 */

_sfrbit _rd = _p3^7;      /* read data for external memory */
_sfrbit _wr = _p3^6;      /* write data for external memory */
_sfrbit _t1 = _p3^5;      /* timer/counter 1 external flag */
_sfrbit _t0 = _p3^4;      /* timer/counter 0 external flag */
_sfrbit _int1 = _p3^3;    /* interrupt 1 input pin */
_sfrbit _int0 = _p3^2;    /* interrupt 0 input pin */
_sfrbit _txd = _p3^1;     /* serial port transmit pin */
_sfrbit _rxd = _p3^0;     /* serial port receive pin */

_sfrbit _ps = _ip^4;      /* priority of serial port interrupt */
_sfrbit _pt1 = _ip^3;     /* priority of timer 1 interrupt */
_sfrbit _px1 = _ip^2;     /* priority of external interrupt 1 */
_sfrbit _pt0 = _ip^1;     /* priority of timer 0 interrupt */
_sfrbit _px0 = _ip^0;     /* priority of external interrupt 0 */

/***** 8052 8 bit SFR's *****/

_sfr _t2con = 0XC8;       /* timer/counter 2 control register */
_sfr _th2 = 0XCD;/* timer 2 high byte */
_sfr _tl2 = 0XCC;/* timer 2 low byte */
_sfr _rldh = 0XCB;        /* timer 2 auto reload high byte */
_sfr _rldl = 0XCA;        /* timer 2 auto reload low byte */

/**** 8052 SFR bits *****/
```

```
_sfrbit _tf2 = _t2con^7;  /* timer 2 overflow flag */
_sfrbit _exf2 = _t2con^6; /* timer 2 external flag */
_sfrbit _rclk = _t2con^5; /* receive clock flag */
_sfrbit _tclk = _t2con^4; /* transmit clock flag */
_sfrbit _exen2 = _t2con^3;        /* timer 2 external enable flag */
_sfrbit _tr2 = _t2con^2;  /* timer 2 run control bit */
_sfrbit _c_t2 = _t2con^1; /* timer/counter select bit */
_sfrbit _cp_rl2 = _t2con^0;       /* capture/reload flag */

_sfrbit _et2 = _ie^5;             /* enable timer 2 interrupt */

#endif   /* _SFR_H */
```

Compiler Data Formats

Compiler Data Formats

Crossware 8051 C Compiler 는 개체를 8051 메모리에 저장할 때 LSB 를 하위 어드레스에 배치합니다. 다음은 각 데이터 형식의 포맷을 설명한 것입니다.

_bit/_sfrbit

1 비트 폭입니다.

전역변수 flag 가 _bit 로 선언될 때 비트 어드레스 영역에 _flag 가 저장됩니다.

char/unsigned char/_sfr

8 비트 폭입니다.

전역 변수 iVar 가 char 로 선언될때 값은 어드레스 _iVar 에 저장됩니다.

int/short/unsigned int/unsigned short/_sfrword

16 비트 폭입니다.

전역변수 iVar 가 정수로 선언될때 LSB 가 _iVar 에 배치되며 MSB 는 _iVar+1 에 저장됩니다.

long/unsigned long

32 비트 크기 입니다.

iVar 이 정수로 선언될때 LSB 는 어드레스 _iVar 에 저장되며 MSB 는 _iVar+3 에 저장됩니다.

float

32 비트 크기 입니다.

전역변수 fVar 가 float 로 선언될때 LSB 는 어드레스 _fVar 에 저장되며 MSB 는 _fVar+3 에 저장됩니다. 부동 소숫점 데이터는 IEEE754 single 데이터 포맷에 따릅니다.

Sign:	bit position 31(1 bit)
Exponent:	bit positions 30 to 23 (8 bits)
Mantissa	bit positions 22 to 0 (24 bits - 최상위 비트는 숨겨져 있습니다.)

double

64 비트 크기 입니다.

전역변수 fVar 가 double 로 선언될 때 LSB 는 어드레스 _fVar 에 저장되며 MSB 는 _fVar+7 에 저장됩니다.

부동 소숫점 데이터는 IEEE754 double 데이터 포맷에 따릅니다.

Sign:	bit position 63(1 bit)
Exponent:	bit positions 62 to 52 (11 bits)
Mantissa	bit positions 51 to 0 (53 bits - 최상위 비트는 숨겨져 있습니다.)

long double

이 버젼의 Crossware 8051 ANSI C 컴파일러는 64 비트 크기 입니다. 형식은 double 형식과 동일합니다. (Enhanced 버젼의 컴파일러는 80 비트 long double 형식을 지원합니다.)

전역변수 fVar 가 double 로 선언될 때 LSB 는 어드레스 _fVar 에 저장되며 MSB 는 _fVar+7 에 저장됩니다.

부동 소숫점 데이터는 IEEE754 double 데이터 포맷에 따릅니다.

Sign:	bit position 63(1 bit)
Exponent:	bit positions 62 to 52 (11 bits)
Mantissa	bit positions 51 to 0 (53 bits - 최상위 비트는 숨겨져 있습니다.)

enum

컴파일러는 enum 변수의 크기를 최소화하고 enumeration 리스트의 값 영역에서 unsigned 와 일치하도록 합니다. 이것은 효율적인 코드를 만듭니다.

enum 변수는 1,2 또는 4 바이트가 됩니다. 예를 들어:

```
enum evals { ea, eb, ec, ed, ef }
enum eValue;

enum ebigvals { eba = -123456, ebb = 0, ebc = 76 };
enum eBigVal;
```

변수 eValue 는 0 부터 4 의 값을 표시하는 것이 요구됩니다. 그러므로 수식 연산에서 unsigned 로 표현합니다. eBigVal 은 -123456 부터 +76 까지의 값을 표시하여야 하므로 수식 연산에서 signed long 으로 표시됩니다.

bit fields

컴파일러는 unsigned char, char, unsigned int, int, unsigned long 그리고 long 형식의 비트 필드를 지원합니다.

비트 필드는 전통적으로 비트 위치에 의하여 묶여집니다. 예를들면:

```
struct tag {
```

```
  unsigned char b1:1
  unsigned char b2:3
  unsigned char b3:4
} S;
```

S 는 8 비트의 개체가 될 것입니다.

- 한개의 비트필드 S.b1 은 S 의 비트 0 에 배치됩니다.
- 세개의 비트필드 S.b2 는 S 의 비트 1 - 3 에 배치됩니다.
- 네개의 비트필드 S.b3 은 S 의 비트 4 - 7 에 배치됩니다.

pointer to data/idata/bdata

8 비트 크기입니다.

전역변수 pVar 이 char _data *으로 선언되면 그 값은 _pVar 의 어드레스가 됩니다.

pointer to xdata/code

16 비트 폭입니다.

전역변수 pVar 이 char _xdata * 또는 char _code*이면 LSB 는 어드레스 _iVar 에 배치되며 MSB 는 _iVar+1 에 배치됩니다.

generic pointer

24 비트 폭입니다.

전역변수 pVar 이 char _generic * 으로 선언되면 LSB 는 어드레스 _iVar 에 MSB 는 _iVar+1 에 그리고 세그멘트 식별 바이트는 _iVar+2 에 배치됩니다.

smart pointer

smart 포인터는 8, 16, 또는 24 비트가 되며 포인터를 사용하는 방법에 따라 결정됩니다.

전역변수 pVar 이 char * 로 ~~~~~~~~ 선언되면 컴파일러는 pVar 의 어떤 형식이 될 것인지를 결정합니다. 형식이 결정되었을 때 포인터의 형식은 1 개의 바이트, 2 개의 바이트, 3 개의 바이트로 고정됩니다. 컴파일러가 pVar 포인터의 형식을 결정하기 전까지는 pVar 는 무제한 포인터입니다. 무제한 포인터는 형식을 갖지않습니다.

function pointer

함수 포인터는 reentrant 함수에서 16 비트 폭이며 small memory 모델의 non-reentrant 함수에서 24 비트 폭 그리고 large memory 모델 non-reentrant 함수에서 32 비트 폭입니다.

전역변수 pFunc 가 reentrant 함수의 포인터로 선언되면 함수의 LSB 는 어드레스 _iVar 에 배치됩니다. MSB 는 _iVar+1 에 배치됩니다.

전역변수 pFunc 는 non-reentrant 함수의 small memory 모델의 포인터로 선언될 때 함수 어드레스의 LSB

는 _iVar 에 배치되며 함수어드레스의 MSB 는 _iVar+1 에 배치되며 함수의 지역데이터는 어드레스 _iVar+2 에 배치됩니다.

전역 변수 pFunc 가 non-reentrant 함수가 large memory 모델의 포인터로 선언되면 함수 어드레스의 LSB 는 _iVar 에 배치되며 , MSB 는 _iVar+1 에 배치되고 함수 지역데이터의 LSB 는 _iVar+1 에 함수의 지역데이터 MSB 는 _iVar 에 배치됩니다.

Interrupt Functions

Manual Coding of Interrupt Vector

지시어 _interrupt 는 함수가 인터럽트인 것을 나타냅니다. 다음의 예처럼 함수 이름 앞에 표시합니다.

```
void _interrupt func()
{
  /* Your interrupt code goes here */
}
```

루틴은 내부 변수를 보호하기 위한 추가적인 코드를 생성합니다. RET 대신에 RETI 가 사용됩니다. 인터럽트에 반응하기 위한 특별한 코드가 생성되지 않습니다. 그러므로 수작업에 의하여 startup 코드를 수정하여야 합니다. 인터럽트로 사용한 함수에 언더스코어("_"를 추가하여 인터럽트 벡터가 지시하도록 합니다. 위의 예에서 Timer 1 오버 플로우가 될 때 호출되게 하려면 startup 코드에 다음과 같은 어셈블리 라인을 추가합니다.

```
cseg   at 001BH
jmp    _func
```

C 함수가 실행되기 전에 긴급히 인터럽트 반응이 필요하다면 jump 명령을 사용하기 전에 추가적인 어셈블리 명령을 추가합니다.

```
cseg   at 001BH
mov    P1, #03H
jmp    _func
```

Automatic Coding of Interrupt Vector

인터럽트 지시자 _interrupt n 을 사용하는 방법이 있습니다. 컴파일러는 _interrupt 지시자를 사용했을 때와 같은 동일한 코드를 생성하며 추가적으로 인터럽트 벡터도 만듭니다. 그러므로 startup 코드를 수정할 필요가 없습니다.

n 에 해당되는 인터럽트 벡터는 다음의 식으로 만들어 집니다.

vector address = n*8 + 3

파일 os.h 에는 표준 8051 과 8052 의 벡터가 정의되어있다.

```
#include <os.h>
void interrupt IVN_TIMER1 func()
```

```
{
    /* Your interrupt code goes here */
}
```

(INV_TIMER1 is defined as 3 in os.h)

벡터테이블의 어드레스가 0 이 아닐 경우 컴파일러 옵션에서 /VO 옵션을 사용합니다. 이것은 컴파일러에게 벡터어드레스 계산 시 옵셋 값을 더합니다. 예를 들어 벡터테이블이 8000h 일 때 /VO:8000 컴파일러 옵션을 사용합니다. 벡터 어드레스는 다음의 계산결과를 사용합니다.

Vector address = n*8 + 3 + $8000

Embedded Development Studio 를 사용한다면 컴파일러 설정 항목에서 벡터 오프셋을 지정할 수 있습니다. (Build->Setting->Compiler)

Register Bank Switching

인터럽트 함수가 빠르게 반응하기 원하면 _using n 지시어를 사용합니다. (_interrupt 지시어 또는 _interrupt n 지시어에 추가하여) 이것은 두 가지로 작용합니다.

- 내부 변수를 저장하고 빠른 인라인 코드를 생성하도록 합니다.
- 레지스터 뱅크를 스위칭하여 레지스터를 대피하지 않아도 되도록 합니다.

이 방법은 부동 소수점 연산 영역을 보관하지는 않습니다. 그러므로 인터럽트 처리기 내에서 부동 소수점을 사용하면 않됩니다.

컴파일러는 n 값에 지정된 레지스터 뱅크에 의하여 스위칭하는 코드를 발생합니다. 컴파일러는 단지 하나의 레지스터 뱅크만을 사용합니다.(register bank 0) , register bank 1, 2 ,3 은 인터럽트 루틴으로 사용 가능합니다.

위의 예처럼 하기 위하여 인터럽트 함수를 다음과 같이 선언하여야 합니다.

```
void _interrupt _using 2 func()
{
  /* Your interrupt code goes here and must not include anything that
  uses floating point */
}
```

수작업에 의하여 startup 파일에서 _func 어드레스로 점프하도록 하여야 합니다. 또는

```
void _interrupt IVN.TIMER1 _using 2 func()
{
  /* Your interrupt code goes here and must not include anythig that
  uses floating point */
}
```

이렇게 하면 Timer 1 overflow 인터럽트가 발생할 때 인터럽트가 처리 되도록 합니다.

In Line Assembler Code

In Line Assembler Code

어셈블러 코드는 두 가지 방법으로 C 소스 코드에 포함될 수 있습니다.

- #asm 과 #endasm 선행 지시자를 사용
- _asm 키워드를 사용

선행 지시자 #asm 과 #endasm 은 C 소스 코드의 어느 곳이든지 어셈블리 코드가 포함되는 것을 가능하게 합니다. #asm 과 #endasm 사이의 모든 라인은 그대로 어셈블러에게 전달됩니다. 그러므로 모든 규정은 어셈블러 문법에 따라야 합니다.

선행 지시자 #if, #ifdef, #ifndef, #else, #elif 그리고 #endif 는 #asm 과 #endasm 사이에서 사용될 수 있습니다.

_asm 키워드는 단지 함수로 사용합니다. 다음과 같은 문법을 사용합니다.

_asm(<string constant>);

string constant 는 수정되지않은 상태로 한 개의 라인으로 어셈블러에 전달됩니다. 각각의 <string constant> 는 어셈블리 코드의 유효한 한 개의 라인입니다.

_asm 신택스의 장점은 C 선행 실행기에서 토큰 교환이 됩니다. 그러므로 문장은 마크로의 연속으로 사용될 수 있습니다. 변수 이름이 cury brace 내에서 사용되면 컴파일러는 변수위치에 적절한 서브 스트링으로 대치합니다.

또한 _asm 신택스는 컴파일러가 쉽게 C 변수를 쉽게 구동할 수 있는 특별한 구성을 지원합니다. 변수 이름을 브레이스에 스트링 상수형태로 기술하면 변수위치에 해당되는 적절한 서브 스트링으로 대치합니다.

컴파일러는 대문자 기호(Uppercase Nemonics)를 만들며 소문자는 in-line assembly 코드를 사용하여 리스트 파일에서 분명하게 구별됩니다.

Accessing Global Variable

전역변수(global variable)는 어셈브러 소스 코드에서 이름에 의하여 직접 구동 됩니다. 컴파일러는 모든 변수에 언더 스코어("_") 문자를 추가합니다. 그러므로 프로그램 작성자는 어셈블러의 구동을 가능하게 하기 위하여 언더 스코어를 추가 합니다. xdata 나 code 영역의 데이터를 구동하려면 데이터 포인터 레지스터 dptr 에 어드레스를 로드 하여야 합니니다. 예를들면 large memory model 에서 nGlobal 은 xdata 영역에서 컴파일 됩니다.

```
char nGlobal;

int get_byte()
{
#asm
        mov     dptr, #_nGlobal
        mov     a, @dptr
#endasm
```

```
}
```
_asm 신택스를 사용하면 컴파일러는 언더 스코어를 추가합니다.

```
char nGlobal;

int get_byte()
{
   _asm(" mov dptr,#{nGlobal}");
   _asm(" movx a,@dptr");
}
```
다음과 같이 컴파일됩니다.
```
get_byte
        mov dptr,#_nGlobal
        movx a,@dptr
        RET
```

위에서 보는 것처럼 {nGlobal}은 _nGlobal 로 대치됩니다.
idata 영역에 있는 변수 데이터를 구동하려면 R0, R1 에 어드레스를 로드 하여야 합니다. 예를 들면 idata 영역에서 small memory model 에서 nGlobal 은 아래와 같습니다.
```
char nGlobal;
int get_byte()
{
        _asm(" mov R0,#nGlobal)");
        _asm(" mov a,@R0");
}
```

데이터 영역은 직접 구동됩니다. 다음의 예는 tiny 또는 mini 모델에서 nGlobal 은 데이터 영역에 배치된 것입니다.

```
char nGlobal

int get_byte()
{
      _asm(" mov a, {nGlobal}");
}
```

Accessing Static Variable

정적 변수는 C 컴파일러에 의하여 고유한 이름으로 할당되며 #asm / #endasm 지시자에 의하여 구동됩니다. 이 이름은 코드의 나머지 부분에 따르기 때문에 _asm 을 사용할 것을 권합니다. 컴파일러는 적당한

이름을 삽입할 것입니다.
xdata 나 code 영역의 변수를 구동할 때 이 어드레스를 dptr 에 로드할 필요가 있습니다. 예를들어 다음을 컴파일하면 large 메모리 모델에서 nStatic 은 xdata 영역에 배치됩니다.

```
static char nStatic;

int get_byte()
{   _asm("    mov    dptr,#{ nStatic }");
    _asm("    movx    a,@dptr");
}
```

다음과 같이 컴파일됩니다.

```
_get_byte
        mov     dptr,#.@2
        movx    a,@dptr
        RET
```

위의 예처럼 {nStatic} 은 .@2 로 변환되며 이 라벨은 특별한 정적 변수를 구동하는 용도로 사용됩니다.
idata 영역에서 데이터를 구동할때 R0 나 R1 에 어드레스를 로드할 필요가 있습니다. 예를 들면 다음은 small memory 모델에서 nStatic 은 idata 영역에 배치된 것을 보여줍니다.

```
static char nStatic;

int get_byte()
{
     _asm("    mov R0,#{nStatic}");
     _asm("    mov a,@R0");
}
```

data 영역의 변수는 직접 구동 가능 합니다. 예를 들면 tiny 또는 mini 모델에서 컴파일 하면 nStatic 은 데이터 영역에 배치됩니다.

```
static char nStatic;
int get_byte()
{
      _asm("    mov    a,{ nStatic }");
}
```

Accessing Local Variables and Function Parameters

OverView

지역 변수와 함수 파라메터의 구동 방법과 위치는 함수가 re-entrant , non-reentrant 인가에 따라 그리고 메모리 모델에 따라 결정됩니다. _asm 신택스에서 브레이스 ("{ }")를 사용하여 변수를 표시하는 것이 좋습니다. 변수가 지역변수인지 함수 파라메터인지 차이는 없으며 동일한 방법으로 구동됩니다. 다음 설명에서 지역변수 항목은 함수 파라메터를 포함합니다. 지역 변수는 스택 프레임을 기준으로 하여 메모리에 배치됩니다. 만약 함수가 reentrant 라면 스택 프레임은 동적이므로 함수가 시작될 때 변수가 만들어지며 함수가 실행 종료되면 없어집니다. 함수가 non-reentrant 라면 스택 프레임은 정적입니다. 링커에 의하여 결정되며 실행 시 고정됩니다.

8051 에서 정적 변수 프레임은 스택 프레임으로 부터 직접 참조하므로 좀더 효율적인 코드가 출력됩니다. 동적인 스택 프레임은 스택 포인터를 로드하여 오프셋을 더하거나 빼야합니다. 정확하게 동일한 제한이 embedded 어셈블러 코드에 적용되며 지역변수를 구동합니다. - 컴파일러는 정적인 스택 프레임에서 정확한 어드레스를 제공합니다. 반면에 동적인 스택 프레임은 오프셋을 제공합니다.

Function Return Value

함수에서 리턴되는 정수 값은 어드레스 __CW51D001 에 있습니다. 이것은 4 개의 바이트 로 데이터 영역에 예약됩니다.

부동 소숫점 함수에서 리턴 값은 어드레스 __CW51D003 에 있습니다. 이것은 8 개의 바이트로 data 영역에 예약됩니다.

Predefined Macros

ANSI Predefined Macros

ANSI 에서 정의된 마크로를 지원합니다.

__LINE__	현재 소스라인의 라인 번호이며 십진수로 표시되는 상수
__FILE__	소스 파일의 이름이며 문자 스트링
__DATE__	호스트 머신의 현재 날자이며 mm dd yyyy 의 문자 스트링
__TIME__	호스트 머신의 현재 시간이며 hh:mm:ss 의 문자 스트링
__STDC__	1 은 ANSI 표준과 정확히 일치하는 것을 나타내며 0 은 그렇지 않음을 나타냅니다. (이것은 /Zz 옵션을 사용하지 않는 한 항상 0 입니다.)

Compiler Specific Predefined Macros

다음의 컴파일러 사양에 따른 마크로를 지원합니다.

_XC51_VER	8051PSDS-A 의 버젼 2.XX 에서 200 으로 정의 되며 C8051NT 컴파일러의 버젼 3.XX 에서 300 으로 정의됩니다.
_X_I51TM	메모리 모델은 Tiny
_X_I51SM	메모리 모델은 Small (Direct and Indirect)
_X_I51LM	메모리 모델은 Large

_X_I51DM	메모리 모델은 Small Direct
_X_I51NR	함수는 non-reentrant 형식 입니다. 함수의 바깥에서 정의되었으며 컴파일러는 non-reentrant 함수로 디폴트로 됩니다.
_X_LDBLE_BITS	표준 판에서 64 비트 부동 소수점이며 확장 판에서 80 비트 부동 소수점으로 정의됩니다.
_atbit	Tasking 키워드를 지원하지 않으면 사용금지 됩니다. _atbit 는 (a)^(b)로 정의됩니다.

Compiler Support Files

Headers and Library Files

많은 종류의 파일이 C 컴파일러와 함께 지원됩니다. 헤더 정보나 라이브러리 코드 입니다. 다음의 파일이 제공됩니다.

Header files

stdio.h stdlib.h string.h malloc.h ctype.h stdarg.h stddef.h limits.h conio.h math.h sfr.h os.h errno.h

Library files

xlib51.lib	standard library for large memory model reentrant code
xlib51n.lib	standard library for large memory model non-reentrant code
xflib51.lib	floating point library for large memory model reentrant code
xflib51n.lib	floating point library for large memory model non-reentrant code
slib51.lib	standard library for small memory model reentrant code
slib51n.lib	standard library for small memory model non-reentrant code
sflib51.lib	floating point library for small memory model reentrant code
sflib51n.lib	floating point library for small memory model non-reentrant code
tlib51.lib	standard library for tiny memory model reentrant code
tlib51n.lib	standard library for tiny memory model non-reentrant code
tflib51.lib	floating point library for tiny memory model reentrant code
tflib51n.lib	floating point library for tiny memory model non-reentrant code

헤더 파일은 다음과 같은 형식으로 내포 합니다.

#include "stdio.h"

헤더 파일은 현재 디렉토리나 INCLUDE 환경 설정에서 지정된 디렉토리에 있어야 합니다.

#include <stdio.h>
헤더 파일은 INCLUDE 환경 설정에서 지정한 디렉토리에 있어야 합니다.

Embedded Development Studio 에서 컴파일러가 동작한다면 INCLUDE 환경설정은 자동적으로 표준 include 디렉토리를 지시합니다.
라이브러리 루틴의 자세한 설명은 C 컴파일러 라이브러리 함수 설명을 참고합니다.

표준 라이브러리는 타겟의 사양에 의존하는 getch()와 putch()를 포함하고 있습니다. 그리고 startup 코드는 타겟 회로에 맞게 수정할 필요가 있을 것입니다.

Startup Module

표준 라이브러리에 포함된 디폴트 startup 모듈은 대부분의 경우 충분할 것입니다. 그러나 원할 경우 자신의 startup 모듈을 만들 수 있습니다.
Embedded Development Studio 사용자는 통합된 startup 파일의 장점을 이용할 수 있습니다. DOS 사용자는 STARTUP 디렉토리에서 STARTUP.ASM 파일을 이용합니다. 이것은 라이브러리를 구성하는데 필요한 것입니다.

Consol I/O

getch() 와 putch()는 consil i/o 에 필요한 타켓에 의존하는 특수한 함수입니다. printf(), puts(), fgetc() 등의 함수는 getch() 와 putch()를 이용하기 때문에 타겟 시스템에서 정확히 동작하여야 합니다. 다른 i/o 함수는 stdin, stdout 그리고 stderr 의 i/o 스트림에서 정상 동작합니다. 라이브러리에 포함된 putch() 와 getch() 는 8051 의 시리얼 포트를 이용합니다. 이 함수를 이용하려면 라이브러리 함수 _InitSerialPort()를 실행하여야 합니다. getch(), putch() 그리고 InitSerialPort() 의 소스 코드는 STARTUP 디렉토리에 SERIO.C 에 포함되어 있습니다. 또한 이 디렉토리에는 작은 프로그램 MAIN.C 에서 이 함수의 사용법을 설명합니다. 라이브러리 버젼은 인터럽트를 사용하므로 _InitSerialPort() 가 실행되면 인터럽트가 사용가능 상태로 되며 i/o 는 인터럽트 방식으로 동작합니다. putch() 는 문자를 출력 버퍼에 쓰고 즉시 리턴 합니다. getch() 는 마지막 수신된 문자를 리턴 하거나 다음번 바이트를 수신하기 위하여 대기할 것입니다. 입력버퍼에 문자가 있는지 알아보려면 kbhit() 을 사용하면 됩니다. 문자가 있는 경우 non-zero 값을 반환합니다.

레지스터 뱅크 2 가 시리얼 포트 모니터 용으로 인터럽트 루틴에 의하여 사용됩니다.

라이브러리에서 getch(), putch() 를 제거하고 사용자가 작성한 함수로 대치하려면 라이브러리 매니져를 이용하여 커맨드 라인에서 다음과 같이 합니다.

Lib -serio <filename>;

C COMPILER LIBRARY FUNCTION REFERENCE

Introduction to Compiler Library Functions

C 컴파일러에 포함되어있는 라이브러리 루틴에 관하여 상세히 설명합니다. 모든 설명은 머리글 요약, 설명 그리고 반환 값 순으로 설명합니다. 요약정보(Summary Information)는 함수 파라메터를 정의하며 반환 값의 형식과 함수가 선언되어있는 내포파일을 표시합니다. 설명부분은 함수의 주요사양에 대하여 설명하며 반환 값 부분은 함수가 리턴 할 때 값의 형태에 대하여 설명합니다.

라이브러리 루틴의 자세한 설명은 알파벳 순서로 나열되었습니다.

abs( ) absl( )

#include <stdlib.h>
int abs(int n); 정수변수의 절대값 계산
long absl(long n) 롱형 변수의 절대값 계산

Parameter

n 정수 값이 변환됩니다.

Description

파라메터 n 의 절대 값을 반환합니다. 즉 n 이 음수이면 양수 값을 반환합니다.

Return

주어진 값을 양의 수로 반환합니다.

Example

```
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        int x;
        int y;

        _InitSerialPort();

        x = -76;
        y = abs(x);

        printf("ABS(%d) = %d\n", x, y);

        while (1);

}
```

acos( ) acosl( )

#include <math.h>
double acos(double x)　　배정도 arc cosine 연산
long double acosl(long double x)　　확장 배정도 arc cosine 연산

Parameter

x　　arc cosine 으로 변환될 -1 부터 +1 범위의 값

Description

이 함수는 x 의 arc cosine 값을 반환합니다. x 가 [-1...+1] 범위에 있지 않으면 0.0 이 반환되며 EDOM 에 errno 가 세트됩니다.

Return

0 부터 pi 범위의 arc cosine 값을 반환합니다.

Example

```
#include <stdio.h>
#include <math.h>

void _InitSerialPort(void);

main()
{
        float x;
        float y;

        _InitSerialPort();

        for (x = -1.0; x <= 1.0; x += 0.1) {
           y = acos (x);
           printf ("ACOS(%f) = %f\n", x, y);
        }

        while(1);
}
```

asin( ) asinl( )

#include <math.h>
double asin(double x)　　배정도 arc sine 연산
long double asinl(long double x)　　확장 배정도 arc sine 연산

Parameter

x　　arc sine 으로 계산될 -1 부터 1 범위의 값

Description

이 함수는 x 의 arc sine 값을 반환합니다. x 가 [-1...+1] 범위에 있지 않으면 0.0 이 반환되며 EDOM 에 errno 가 세트됩니다.

Return

주어진 값의 -pi/2 와 pi/2 범위의 arc sine 값을 반환합니다.

Example

```
#include <stdio.h>
#include <math.h>

void _InitSerialPort(void);

main()
{
    float x;
    float y;

   _InitSerialPort();

    for (x = -1.0; x <=1.0; x += 0.1) {
       y = asin (x);
       printf("ASIN(%f) = %f\n", x, y);
    }

    while(1);

}
```

atan( ) atanl( )

#include <math.h>

double atan(double x)　　배정도 arc tangent 값 계산

long double atanl(long double x)　　확장 배정도 arc tangent 값 계산

Parameter

x　　arc tangent 로 변환될 입력값

Description

이 함수는 x 의 arc tangent 값을 계산합니다.

Return

이 함수는 -pi/2 와 pi/2 범위의 arc tangent 값을 반환합니다.

Example

```
#include <stdio.h>
#include <math.h>

void _InitSerialPort(void);

main()
{
        float x,y;

        _InitSerialPort();

        for ( x = -10.0; x <= 10.0; x += 0.5) {
           y = atan (x);
          printf("ATAN(%f) = %f\n", x, y);
        }

}
```

atan2( ) atan2l( )

#include <math.h>

double atan(double x)　　배정도 arc tangent 값 계산

long double atanl(long double x)　　확장 배정도 arc tangent 값 계산

Parameter

x　　arc tangent 로 변환될 입력 값

Description

이 함수는 x 의 arc tangent 값을 계산합니다.

Return

이 함수는 -pi/2 와 pi/2 범위의 arc tangent 값을 반환합니다.

Example

```
#include <stdio.h>
#include <math.h>

void _InitSerialPort(void);

main()
{
        float x, y, z;

        _InitSerialPort();

        x = -1.0;

        for ( y = -10.0; y <= 10.0; y += 0.5) {
           z = atan2(y, x);
          printf("ATAN2(%f/%f) = %f\n", y, x, z);
        }

}
```

atoi( ) atol( ) atof( ) atolf( )

#include <stdlib.h>

int atoi(const char* string);	스트링을 int 로 변환합니다.
long atol(const char* string);	스트링을 long 으로 변환합니다.
double atof(const char* string);	스트링을 double 로 변환합니다.
long double atolf(const char* string);	스트링을 long double 로 변환합니다.

Parameter

string 변환될 스트링의 포인터

Description

이 함수는 문자 스트링을 정수값(atoi()), 롱정수(atol()), 배정도 부동소수점 변수(atof()) 또는 확장 배정도 변수로 변환합니다.

atoi() 와 atol():

스트링의 시작위치의 공백문자는 무시됩니다. 첫번째 문자는 + 또는 십진숫자가 될 것입니다. 십진 숫자가 아닌 문자를 만날 때까지 스트링을 읽습니다. 대부분의 경우에 Null 문자가 될 것입니다.

atof() 와 atolf():

스트링의 시작위치의 공백문자는 무시됩니다. 다음과 같은 포맷이 권장됩니다.

[sign][digit][.digit][E 또는 e][sign][digits]

변환은 위와 같은 포맷이 아닐 때 종료됩니다. 대부분 Null 코드일 것입니다.

Return

이 함수는 십진수로 표현된 스트링 입력을 숫자로 변환합니다. 오버 플로우가 발생하더라도 오버 플로우 검사는 실시되지 않으며 리턴 값은 정의되지 않습니다. 변환에 실패한 경우 0 이 반환됩니다.

Example atoi()

```
#include <stdio.h>
#include <stdlib.h>

void _InitSerialPort(void);

main()
{
        int x;
        char s[] = "12345";

        _InitSerialPort();

        x = atoi(s);

        printf("ATOI(%s) = %d\n", s, x);

}
```

Example atof()

```
#include <stdio.h>
#include <stdlib.h>

void _InitSerialPort(void);

main()
{
        float x;
        char s[] = "-1.124E2";

        _InitSerialPort();

        x = atof(s);

        printf("ATOF(%s) = %f\n", s, x);

}
```

Example atol()

```
#include <stdio.h>
#include <stdlib.h>

void _InitSerialPort(void);

main()
{
        long x;
        char s[] = "1234567890";

        _InitSerialPort();

        x = atol(s);

        printf("ATOL(%s) = %ld\n", s, x);

}
```

ceil( ) ceill( )

#include <math.h>

double ceil(double x); 근접한 배정도 소수점으로 끝자리 교정합니다.

long double ceill(long double x) 근접한 확장 배정도 소수점으로 끝자리 교정합니다.

Parameter

x 부동 소수점 값을 교정합니다.

Description

이 함수는 x 보다 작지 않은 가장 작은 정수를 계산합니다.

Return

이 함수는 x 보다 작지 않은 가장 작은 정수를 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        float x, y;
        int n;


        _InitSerialPort();

        y = ceil(1.05);
        printf("The ceil (1.05) is %f\n", y);    /* y is equal to 2.0 */

        y = ceil(-1.05);
        printf("The ceil (-1.05) is %f\n", y);    /* y is equal to -1.0 */

        while(1);
}
```

clearerr( )

#include <stdio.h>
void clearerr(FILE *stream); 스트림의 에러 클리어를 지정합니다.

Parameter
stream FILE 스트럭쳐의 포인터

Description
이 함수는 지정한 스트림에 대하여 error 표시를 0 으로 리셋합니다. 에러 표시는 자동적으로 리셋 되지 않습니다. 에러 표시는 자동적으로 클리어 되지 않습니다. 에러 표시가 한번 세트 되면 clearerr() 가 실행되기 전까지는 스트림 조작시 계속해서 에러가 반환됩니다.

Return
반환되는 값은 없습니다.

cos( ) cosl( )

#include <math.h>

double cos(double x)　　배정도로 cosine 계산합니다.

long double cosl(long double x)　　확장 배정도로 cosine 계산합니다.

Parameter

x　　라디안 값으로 나타낸 부동 소수점 값 값 각도

Description

x 의 cosine 값을 계산 합니다.

Return

이 함수는 파라메터 전달 값의 cosine 값을 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

#define pi 3.141592

void _InitSerialPort(void);

main()
{
        int i;
         float x, y;

         _InitSerialPort();

         for ( i = 0 ; i <= 90; i += 5) {
              x = pi * i / 180;                                    // Degree to Radian
              y = cos( x );
              printf("Rad = %f COS(Deg %d) = %f\n", x, i, y);
         }
         while(1);
}
```

cosh( ) coshl( )

#include <math.h>

double cosh(double x)　　배정도로 하이퍼블릭 cosine 을 계산합니다.

double coshl(long double x)　　확장 배정도로 하이퍼블릭 cosine 을 계산합니다.

Parameter

x　　라디안으로 표시된 부동 소수점 값 각도

Description

이 함수는 x 의 하이퍼블릭 cosine 값을 계산합니다.

Return

하이퍼블릭 cosine 값을 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

#define pi 3.141592

void _InitSerialPort(void);

main()
{
        float x, y;

        _InitSerialPort();

        for ( x = 0 ; x < (2 * pi); x += 0.1) {
            y = cosh (x);
            printf("COSH(%f) = %f\n", x, y);
        }
        while(1);
}
```

_ecvt( ) _ecvtl( )

#include <stdlib.h>

char* **_ecvt**(double value, int count, int* dec, int* sign);	부동 소수점 값을 스트링으로 변환합니다.
char* **_evctl**(long double value, int count, int* dec, int* sign);	부동 소수점 값을 스트링으로 변환합니다.

Parameter

value	변환될 값
count	소수점 이후의 디지트 개수
dec	십진 포인터 위치의 저장 포인터 정수
sign	부호 표시용 정수저장

Description

_ecvt() 와 _ecvtl() 함수는 부동 소수점 숫자를 Null 문자 스트링으로 변환합니다. 변환된 값의 디지트는 스트링에 정적으로 배치되며 Null 문자가 추가 됩니다. 카운트 인자는 저장될 디지트의 숫자를 지정합니다. 카운트 디지트 보다 작을 때 0 이 추가 됩니다. 단지 디지트만 스트링에 저장됩니다. 십진 소수점의 위치와 값의 부호는 호출 시 제공되는 정수에 의하여 분리되어 저장됩니다. 부호인자는 정수를 지시하며 이것은 값의 부호의 위치를 나타냅니다. 이것이 0 이면 값은 양수이며 0 이 아니면 값은 음수 입니다.

Return

이 함수는 디지트 스트링의 포인터를 반환합니다.

Example

```
#include <stdlib.h>
#include <stdio.h>

char *buffer;

void _InitSerialPort(void);

// buffer will contain “3141592654”

main(){
        int decimal, sign;
        int precision = 10;

      _InitSerialPort();

      buffer = _ecvt(3.1415926535, precision, &decimal, &sign);
      printf("buffer = \"%s\", decimal = %d, sign = %d\n", buffer, decimal, sign);

      while(1);
}
```

exit( )

#include <stdlib.h>
void exit(int status); 프로그램을 종료합니다.

Parameter
status exit 상태를 나타내는 정수

Description
exit() 을 호출하면 프로그램이 종료됩니다. 라이브러리에서 이 함수는 단순히 프로그램 어드레스 0 으로 점프합니다. 그러므로 exit() 를 호출하면 프로그램이 재실행됩니다.

Return
exit() 은 반환되지 않습니다.

Example

```
#include <stdlib.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        _InitSerialPort();

        printf("Terminate\n");

        exit();

        while(1);
}
```

exp( ) expl( )

#include <math.h>

double exp(double x); 배정도 exponential 을 계산합니다.

long double expl(long double x) 확장 배정도 exponential 을 계산합니다.

Parameter

x 부동 소수점 값

Description

x 의 e 승 값을 계산합니다.

Return

이 함수는 x 의 e 승 값을 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        float x, y;

        _InitSerialPort();

        x = 2.302585093;
        y = exp(x);                    // y = 10

        printf("The EXP(%f) = %f\n", x, y);

        while(1);
}
```

fabs( ) fabsl( )

#include <math.h>

double fabs(double x);　　배정도 부동 소수점 숫자의 절대값을 구합니다.

long double fabsl(long double x);　　확장 배정도 부동 소수점 숫자의 절대값을 구합니다.

Parameter

x　　부동 소수점 값

Description

x 의 절대값을 계산합니다.

Return

전달된 값의 절대값을 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        float x, y;

        _InitSerialPort();

        x = -12.345;
        y = fabs(x);
        printf("FABS(%f) = %f\n", x, y);

        x = 98.7654;
        y = fabs(x);
        printf("FABS(%f) = %f\n", x, y);

        while(1);
}
```

_fcvt( ) _fcvtl( )

#include <stdlib.h>

char* _fcvt(double value, int count, int* dec, int* sign); 부동 소수점 값을 스트링을 변환합니다.

char* _fcvtl(long double value, int count, int* dec, int* sign); 부동 소수점 값을 스트링을 변환합니다.

Parameter

value	변환할 수
count	소수점 이후의 자리수
dec	십진 소수점 위치를 표시하는 정수의 포인터
sign	부호를 표시하는 정수의 포인터

Description

_fcvt() 와 _fcvtl() 함수는 부동 소수점 숫자를 Null 문자 스트링으로 변환합니다. 변환된 값의 디지트는 스트링에 정적으로 배치되며 Null 문자가 추가 됩니다. 카운트 인자는 저장될 디지트의 숫자를 지정합니다. 카운트 디지트보다 적을 때에 0 이 추가 됩니다. 단지 디지트만 스트링에 저장됩니다. 십진 소수점의 위치와 값의 부호는 호출 시 제공되는 정수에 의하여 분리되어 저장됩니다. 부호인자는 정수를 지시하며 이것은 값의 부호의 위치를 나타냅니다. 이것이 0 이면 값은 양수이며 0 이 아니면 값은 음수 입니다.

Return

이함수는 디지트 스트링의 포인터를 반환합니다.

Example

```
#include <stdlib.h>
#include <stdio.h>

char *buffer;

void _InitSerialPort(void);
// buffer will contain "3141592654"

main(){
        int decimal, sign;
        int precision = 10;

        _InitSerialPort();
        buffer = _fcvt(3.1415926535, precision, &decimal, &sign);
        printf("buffer = \"%s\", decimal = %d, sign = %d\n", buffer, decimal, sign);

        while(1);

}
```

ferror( )

#include <stdio.h>
int ferror(FILE* stream); 주어진 스트림에 대하여 에러를 검사합니다.

Parameter
stream FILE 스트럭쳐의 포이터

Description
이 함수는 스트림의 에러 값을 반환합니다. 리딩 또는 라이팅 에러가 발생하면 clearerr() 가 호출 되기 전 까지는 에러 표시가 세트 됩니다.

Return Value
에러가 발생하면 값이 반환됩니다. 에러가 없으면 0 이 반환됩니다.

fgetc( )

#include <stdio.h>
int fgetc(FILE* stream); 스트림으로 부터 문자를 읽습니다.

Parameter
stream FILE 스트럭쳐의 포인터

Description
입력 스트림으로 부터 한개의 문자를 읽습니다. 입력 스트림은 stdin 을 지원합니다.

Return Value
리턴 값은 읽은 문자이거나 에러가 발생한 경우 EOF 입니다.

fgets( )

#include <stdio.h>
int fgets(char* string, int n, FILE* stream); 스트림으로 부터 문자를 읽습니다.

Parameter

string	수신된 문자의 문자 어레이 포인터
n	읽은 문자 수
stream	파일 스트럭쳐의 포인터

Description

이 함수는 스트림으로 부터 스트링으로 지정된 배열에서 n-1 개의 문자를 읽습니다. 입력은 개행 문자를 읽으면 종료합니다. 개행 문자에는 '\0"이 포함됩니다. stdin 은 단지 입력 스트림을 지원합니다.

Return Value

스트링이나 에러일경우 NULL 을 반환합니다.

fileno( )

#include <stdio.h>
int fileno(FILE* stream); 주어진 스트림에서 파일 핸들을 얻습니다.

Parameter
stream FILE 스트럭쳐의 포인터

Description
이 함수는 주어진 스트림에서 파일 핸들을 반환합니다.

Return Value
반환 되는 것은 파일 핸들입니다. 스트림이 유효한 것인가는 확인하지 않습니다. 유효하지 않은 스트림을 전달하면 반환결과는 예측할 수 없습니다.

floor( ) floorl( )

#include <math.h>
double floor(double x); 배정도 정수로 자리내림 합니다.
long double floorl(double x); 확장 배정도 정수로 자리내림 합니다.

Parameter

x 부동 소수점 값

Description

이 함수는 x 보다 크지 않은 최대 정수를 계산합니다.

Return

이 함수는 x 보다 크지 않은 최대 정수를 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        float y;

        _InitSerialPort();

        y = floor(2.8);                         // y is 2.0
        printf("The floor of 2.8 is %f\n", y);

        y = floor(-2.8);                        // y is -3.0
        printf("The floor of -2.8 is %f\n", y);

        while(1);

}
```

fprintf( )

#include <stdio.h>
int fprintf(FILE* stream, const char* format, ...);

Parameter

stream	FILE 스트럭처의 포인터
format	포맷 컨트롤 스트링
...	변환및 라이트될 선택적인 인수

Description

이 함수는 출력 스트림에 주어진 인수를 포맷 처리하여 라이트 합니다.

Return Value

스트림에 라이트된 문자의 개수를 반환합니다.

fputs( )

#include <stdio.h>
int fputs(const char* string, 스트림에 스트링을 라이트 합니다.
FILE* stream);

Parameter

string	라이트 되는 스트링의 포인터
stream	FILE 스트럭쳐의 포인터

Description

이 함수는 stream 에 스트링을 출력하며 Null 문자('\0') 대신에 개행 문자('\n')를 첨부합니다.

Return Value

반환되는 값은 언제나 마지막 문자이며 개행 문자가 됩니다. EOF(-1) 이 리턴된 경우는 에러를 나타냅니다.

free( )

#include <malloc.h>
void free(void* ptr); 지정된 메모리의 프리 블록

Parameter
ptr 지정된 메모리 블록의 포인터

Description
이 함수는 malloc() 함수에 의하여 배치되었던 메모리 블록을 해체합니다. 파리메터 ptr 은 메모리 블록의 포인터이며 포인터 값은 malloc() 에 의한 원래의 값이 반환됩니다. 해체되는 바이트 수는 malloc() 설정시의 바이트 수와 동일 합니다.

Return
반환되는 값은 없습니다.
malloc() 에 의하여 설정된 메모리 블록을 ptr 값이 올바르게 지시하는지는 검사하지 않습니다. 그리고 유효하지 않은 포인터가 사용되었을 때 메모리 포인터는 여유 메모리 풀에 추가 됩니다. 이것은 연속적으로 malloc() 을 호출할 때 메모리 충돌이 발생할 것입니다.

frexp( ) frexpl( )

#include <math.h>

double frexp(double x, int *exp)	가수부와 지수부를 분리합니다.
long double frexpl(long double x int *exp)	가수부와 지수부를 분리합니다.

Parameter

x	부동 소수점 값
	지수부를 저장할 정수 포인터

Description

이 함수는 부동소숫점 숫자의 가수부와 지수부를 분리합니다. 가수부는 0.5 부터 1.0 범위의 값입니다.

Return

이 함수는 가수부를 반환합니다. x 가 0 일 때 가수부와 지수부는 0 이 됩니다. 에러가 리턴 되지는 않습니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        float x, y;
        int n;


        _InitSerialPort();

        x = 16.4;

        y = frexp(x, &n);

       /* Y = 0.512500 and n =5 */
        printf("Y = %f and n =%d\n", y, n);

        while(1);
}
```

fscanf( )

#include <stdio.h>

int fscanf(FILE* stream, const char* format, ...);

Parameter

stream	FILE 스트럭쳐의 포인터
format	포맷컨트롤 스트링
...	스캔 데이터 인수

Description

이 함수는 스트림의 현재 위치로부터 포맷된 데이터를 읽습니다. 그리고 변환된 값을 배열된 포인터형 인수에 지정합니다. 포맷컨트롤 스트링은 scanf() 함수의 포맷 인수와 같습니다.

Return Value

함수는 필드수를 성공적으로 변환하여 지정합니다.

getc( )

#include <stdio.h>
int getc(FILE* stream);　　스트림으로 부터 문자를 읽습니다.

Parameter
stream　　FILE 스트럭쳐 포인터

Description

입력 스트림으로 부터 한 개의 문자를 읽습니다.

Return Value

읽은 문자를 반환하거나 에러가 발생한 경우 EOF 를 반환합니다.

getch( )

#include <conio.h>
int getch(void);　　콘솔로부터 한 개의 문자를 읽습니다.

Description

이 함수는 콘솔로부터 문자를 읽습니다. 루틴은 타겟 시스템에 적합하도록 구성 가능합니다. 라이브러리에는 8051 시리얼 포트로부터 문자를 읽는 버전이 포함되어 있습니다. 다른 방법으로 구성하는 경우는 Consol I/O 부분을 참고하시기 바랍니다.

Return Value

읽은 문자를 반환합니다. 에러인 경우 EOF 를 반환합니다.

Example

```
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        char c;

        _InitSerialPort();

        while ((c = getch ()) != 0x1B) {
                printf (" Character = %c %d(Dec) %X(Hex) %o(Oct)\n", c,c,c,c);

        }

        while(1);
}
```

getchar( )

#include <stdio.h>
int getchar(void); stdin 으로부터 문자를 읽습니다.

Description

표준입력 -stdin 으로부터 한개의 문자를 읽습니다.

getchar() 는 getc(stdin)으로 마크로 정의되어 있습니다.

Return Value

읽은 문자가 반환됩니다. 에러가 발생한 경우 EOF 가 반환됩니다.

Example

```
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        char c;

        _InitSerialPort();

        while ((c = getchar ()) != 0x1B) {
                printf (" Character = %c %d(Dec) %X(Hex) %o(Oct)\n", c,c,c,c);

        }

        while(1);
}
```

getche( )

#include <conio.h>
int getche(void);　　콘솔로부터 문자를 읽고 에코합니다.

Description

이 함수는 getch()를 이용하여 문자를 콘솔로부터 읽습니다. 읽은 문자는 스크린으로 에코합니다.

Return Value

읽은 문자를 반환합니다. 에러를 리턴 하지 않습니다.

Example

```
#include <conio.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        char c;

        _InitSerialPort();

        while ((c = getche ()) != 0x1B) {
                printf (" Character = %c %d(Dec) %X(Hex) %o(Oct)\n", c,c,c,c);

        }

        while(1);
}
```

gets( )

#include "stdio.h"
int gets(char* string); stdin 으로부터 한개의 라인을 읽습니다.

Parameter
string 입력된 문자열의 포인터

Description
이 함수는 스트링에 의하여 포인트 표준 입력 stdin 의 다음 라인을 읽습니다. 라인의 끝의 개행 문자는 '\0' 으로 대치됩니다.

Return Value
이 함수는 스트링을 반환합니다. 에러인 경우 NULL 을 반환합니다.

Example

```
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        char buffer[100];

        _InitSerialPort();

        do {
                gets(buffer);
                printf("Input string \"%s\"", buffer);

        } while (buffer[0] != '\0');

        while(1);
}
```

isalnum( )

#include <ctype.h>
int isalnum(int c);　　알파벳 문자인지 검사합니다.('A'-'Z', 'a'-'z' 또는 '0'-'9')

Parameter
C　　테스트할 정수

Description
이 함수는 전달된 정수가 영문자/숫자인지를 검사합니다.
문자는 ASCII 인 것으로 가정합니다.

Return Value
문자가 영문자/숫자인 경우 TRUE(1) 이 반환되며 그렇지 않은 경우 FALSE(0)이 반환됩니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (isalnum (c) ? yes : no);
           printf("%d 0x%X isalnum (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

isalpha( )

#include <ctype.h>
int isalpha(int c);　　문자인지 검사합니다 ('A'-'Z' 또는 'a'-'z')

Parameter
C　　검사할 문자(정수)

Description
이 함수는 전달된 정수가 영문자인지를 검사합니다.
문자는 ASCII 인 것으로 가정합니다.

Return
문자가 영문자/숫자인 경우 TRUE(1) 이 반환되며 그렇지 않은 경우 FALSE(0)이 반환됩니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (isalpha (c) ? yes : no);
           printf("%d 0x%X isalpha (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

isascii( )

#include <ctype.h>
int isascii(int c);　　ASCII 문자(0X00 - 0X7F)인지 검사합니다.

Parameter
c　　검사할 문자(정수)

Description
이 함수는 전달됨 문자(정수)가 ASCII 문자영역(0X00 - 0X7F)에 있는지 검사합니다.

Return Value
알파 뉴메릭일 때 TRUE(1) 이 반환됩니다. 다른 경우는 FALSE(0) 이 반환됩니다.

iscntrl( )

#include <ctype.h>
int iscntrl(int c); 제어문자(0X00 - 0X1F 또는 0X7F) 인지 검사합니다.

Parameter

c 검사할 문자(정수)

Description

이 함수는 문자(정수)가 제어문자(0X00 - 0X1F 또는 0X7F)인지 검사합니다.

Return Value

제어 문자일 때 TRUE(1) 을 반환합니다. 그렇지 않은 경우 FALSE(0)이 반환됩니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (iscntrl (c) ? yes : no);
           printf("%d 0x%X iscntrl (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

isdigit( )

#include <ctype.h>
int isdigit(int c);　　숫자인지 검사 ('0' - '9')

Parameter

c　　검사할 문자(정수)

Description

이 함수는 문자가 숫자('0' - '9')인지 검사합니다.

Return Value

문자가 숫자일 경우 TRUE(1)을 반환합니다. 그렇지 않을 경우 FALSE(0)을 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (isdigit (c) ? yes : no);
           printf("%d 0x%X isdigit (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

isgraph( )

#include <ctype.h>
int isgraph(int c);　　프린트 가능한 문자(0X21 - 0X7E)인지 검사합니다.

Parameter
c　　검사할 문자(정수)

Description
이 함수는 문자(정수)가 프린트 가능한 문자인지 검사합니다.

Return Value
프린트 가능한 문자이면 TRUE(1)을 반환합니다. 그렇지 않으면 FALSE(0)을 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (isgraph (c) ? yes : no);
           printf("%d 0x%X isgraph (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

islower( )

#include <ctype.h>
int islower(int c);　　소문자('a' - 'z') 인지 검사합니다.

Parameter
c　　검사할 문자(정수)

Description
이 함수는 문자가 소문자인지 검사합니다.

Return Value
문자가 소문자이면 TRUE(1) 을 반환합니다. 그렇지 않으면 FALSE(0)을 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (islower (c) ? yes : no);
           printf("%d 0x%X islower (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

isprint( )

#include <ctype.h>
int isprint(int c);　　프린트 가능한 문자인지 검사합니다.

Parameter

c　　검사할 문자(정수)

Description

이 함수는 프린트가 가능한 문자(0X20 - 0X7E)인지 검사합니다.

Return Value

프린트 가능한 문자이면 TRUE(1)을 반환합니다. 그렇지 않으면 FALSE(0) 을 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (isprint (c) ? yes : no);
           printf("%d 0x%X isprint (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

ispunct( )

#include <ctype.h>
int ispunct(int c);　　기호 문자(punctuation)인지 검사합니다.

Parameter
c　　검사할 문자

Description
이 함수는 기호 문자(puctuation)인지 검사합니다.
기호문자

!	**“**	**#**	**$**	**%**	**&**	**‘**	**(**
)	*****	**+**	**,**	**-**	**.**	/	**:**
;	**<**	**=**	**>**	**?**	**@**	**[**	\
]	^	**_**	`	**{**	**|**	**}**	~

Return Value
기호 문자 (punctuation)일 때 TRUE(1)을 반환합니다. 그렇지 않으면 FALSE(0)을 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (ispunct (c) ? yes : no);
           printf("%d 0x%X ispunct (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

isspace( )

#include <ctype.h>
int isspace(int c);　　공백문자(0X09 - 0X0D 또는 0X20)인지 검사합니다.

Parameter

c　　검사할 문자(정수)

Description

이 함수는 문자가 공백문자 (0X09 - 0X0D 또는 0X20)인지 검사합니다.

Return Value

공백 문자일 때 TRUE(1)을 반환합니다. 그렇지 않으면 FALSE(0)을 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (isspace (c) ? yes : no);
           printf("%d 0x%X isspace (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

isupper( )

#include <ctype.h>
int isupper(int c);　　대문자('A' - 'Z')검사 합니다.

Parameter
c　　검사할 문자(정수)

Description
이 함수는 문자가 대문자('A' - 'Z')인지 검사 합니다.

Return Value
문자가 대문자('A' - 'Z') 이면 TRUE(1) 을 반환합니다. 그렇지 않으면 FALSE(0)을 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (isupper (c) ? yes : no);
           printf("%d 0x%X isupper (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

isxdigit( )

#include <ctype.h>
int isxdigit(int c);　　헥사 데시멀('A'-'F', 'a'-'f' 또는 '0'-'9')인지 검사합니다.

Parameter

c　　검사할 문자(정수)

Description

이 함수는 문자가 헥사 데시멀('A'-'F', 'a'-'f' 또는 '0'-'9')인지 검사합니다.

Return Value

문자가 헥사 데시멀이면 TRUE(1) 을 반환합니다. 그렇지 않으면 FALSE(0)을 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;
        char *p;
        char yes[] = "YES";
        char no[] = "NO";

        _InitSerialPort();

        for ( c = 0; c <= 127; c++) {
          p = (isxdigit (c) ? yes : no);
           printf("%d 0x%X isxdigit (%c) %s\n", c, c, c, p);
        }
        while(1);
}
```

itoa( )

#include <stdlib.h>
char *itoa(int number, char* string, int radix); 스트링으로 변환합니다.

Parameter

Number	변환될 숫자(정수)
String	문자열을 저장할 스트링의 포인터
Radix	변환될 문자 개수

Description

이 함수는 정수를 ASCIIZ 스트링으로 변환하여 저장합니다. 스트링의 길이는 radix 로 지정합니다.

Return Value

스트링의 포인터를 반환합니다.

Example

```
#include <stdlib.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        char str[20] ="";
        int i;
        char *s;

        _InitSerialPort();

        i = 1234;

        s = itoa(i, str, 10);

        printf("%s\n", s);

        while(1);
}
```

kbhit( )

#include <conio.h>
char* _kbhit(void); 읽지 않은 문자가 있는지 검사합니다.

Description

이 함수는 콘솔에서 읽지않은 문자가 있는지 검사합니다.

Return Value

문자가 있으면 non-zero 값이 반환되며 그렇지 않으면 zero 가 반환됩니다.

ldexp( ) ldexpl( )

#include <math.h>

double ldexp(double x, int exp)	가수부와 지수부로부터 계산합니다.
long double ldexpl(long double x, int exp)	확장 배정도 부동소수점의 지수부와 가수부로부터 계산합니다.

Parameter

x	부동소수점 값
exp	정수 지수부

Description

이 함수는 x 의 부동소수점 값에 2 승 누계 값을 곱합니다.

Return

이 함수는 x 에 2 의 exp 승 값을 곱하여 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        float x, y;
        int p;

        _InitSerialPort();

        x = 1.5;
        p = 5;
        y = ldexp(x, p);                    /* y = 48.0 */

         printf("The ldexp(%f, %d) = %f\n", x, p, y);

        while(1);

}
```

log( ) logl( )

#include <.math.h >

double log(double x)	배정도 자연 로그를 계산합니다.
long double log(long double x)	확장 배정도 자연로그를 계산합니다.

Parameter

x	부동 소수점값

Description

이 함수는 x 의 자연 로그 값을 계산합니다. x 가 0 보다 작을 때 결과 값은 예측할 수 없습니다.

Return

전달된 값의 자연 log 값을 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        float x, y;

        _InitSerialPort();

        for (x = 0; x <=100; x += 10) {
                y = log(x);
                printf("LOG(%f) = %f\n", x, y);
        }
        while(1);
}
```

log10( ) log10l( )

#include <math.h>

double log10(double x)	배정도로 밑 수가 10 인 log 값을 계산합니다.
long doube log10l(long double x)	확장 배정도로 밑 수가 10 인 log 값을 계산합니다.

Parameter

x	부동소수점 값

Description

이 함수는 밑 수가 10 인 x 의 log 값을 구합니다. x 가 0 보다 작으면 결과값은 예측할 수 없습니다.

Return Value

밑 수가 10 인 x 의 log 값을 계산하여 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        float x, y;

        _InitSerialPort();

        for (x = 0; x <=100; x += 10) {
                y = log10(x);
                printf("LOG10(%f) = %f\n", x, y);
        }
        while(1);
}
```

ltoa( )

#include <stdlib.h>
char *ltoa(long number, char* string, int radix); 스트링으로 변환합니다.

Parameter

number	변환될 정수
string	문자열이 저장될 포인터
radix	밑수

Description

이 함수는 배정도 정수를 ASCIIZ 스트링으로 변환하여 스트링에 저장합니다.

Return Value

스트링의 포인터를 반환합니다.

malloc( )

#include <malloc.h>
void *malloc(size_t size);　　메모리 블록을 배정합니다.

Parameter
size　　메모리 블록 설정에 필요한 바이트 수

Description
이 함수는 지정한 크기의 바이트수로 메모리 블록을 설정합니다. 블록의 첫번째 바이트는 메모리 형식의 저장에 맞도록 적절히 확보됩니다.

Return Value
블록의 첫번째 바이트를 지시하는 포인터를 반환합니다.
메모리가 충분치 않으면 NULL 이 반환됩니다.

Example

```
#include <malloc.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{

        char *p;

        _InitSerialPort();

        p = malloc(200);

        if(p == NULL)
           printf("Not enough memory space\n");
           else
              printf("Memory allocated\n");

        while(1);
}
```

memchr( )

#include <string.h>
char *memchr(void *buf, int c, size_t count); 버퍼에서 문자를 찾습니다.

Parameter

buf	버퍼 포인터
c	찾을 문자
count	찾을 문자의 개수

Description

이 함수는 buf 에서 지정한 숫자 범위 내에서 문자 c 를 찾습니다.

Return Value

buf 에서 문자를 찾았으면 포인터를 반환합니다. 그렇지 않으면 NULL 이 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        char buff[100] = "Crossware 8051 C compiler";
        char *c;
        char t = 'm';

       _InitSerialPort();

        c = memchr (buff, t, sizeof (buff));

        if (c == NULL)
           printf("'%c' was not found in the buffer\n", t);
           else
              printf ("Found '%c' in the buffer\n", t);

        while(1);
}
```

memcmp( )

#include <string.h>

char *memcmp(void **buf1, void** buf2, 두개의 버퍼에 있는 문자를 비교합니다.
size_t count);

Parameter

buf1	첫번째 버퍼의 포인터
buf2	두 번째 버퍼의 포인터
count	비교할 문자의 개수

Description

이 함수는 buf1 과 buf2 의 지정된 문자 수를 비교합니다.

Return Value

이 함수는 buf1 과 buf2 의 다음과 같은 관계에 따라 숫자 값을 반환합니다.

Value	Meaning
<0	buf1 이 buf2 보다 작다.
=0	buf1 과 buf2 가 같다.
>0	buf1 이 buf2 보다 크다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
            char string1[] = "012345";
            char string2[] = "234567";

           char c;

          _InitSerialPort();

            c = memcmp(string1, string2, 7);

            printf("Return Value = %d\n", c);

            while(1);
}
```

memcpy( )

#include <string.h>
char *memcpy(void *dest, void* src, size_t count); 버퍼의 내용을 다른곳으로 복사합니다.

Parameter

dest	저장될 버퍼 포인터
src	복사할 버퍼 포인터
count	복사 바이트 수

Description

이 함수는 src 로 부터 dest 로 복사합니다. source 와 destination 버퍼가 겹쳐지게 되면 동작이 정확하지 않게 됩니다. 버퍼가 겹쳐진 경우 memmove()를 사용합니다.

Return Value

destination 버퍼의 start 포인터를 반환합니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
            char string1[] = "012345678";
            char string2[] = "ABCDEFGHI";

           char *p;

          _InitSerialPort();

           printf("Before: String 1 = '%s' String 2 = '%s'\n", string1, string2);

            p = memcpy(string1, string2, 4);

           printf("After:    String 1 = '%s' String 2 = '%s'\n", string1, string2);

            while(1);
}
```

memmove( )

#include <string.h>

char *memmove(void *dest, void* src, size_t count); 버퍼로부터 문자를 다른 곳으로 복사합니다.

Parameter

dest	dest 포인터
src	source 포인터
count	복사할 문자 개수

Description

이 함수는 src 로 부터 dest 로 지정된 수만큼 복사합니다. src 와 dest 가 겹칠 경우 이 함수는 src 를 먼저 복사하여 두므로 원래의 문자가 정확히 복사됩니다.

Return Value

destination 버퍼의 시작 포인터가 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
            char string1[] = "0123456789";
            char string2[] = "ABCDEF";

           char *p;

          _InitSerialPort();

           printf("Before: String 1 = '%s' String 2 = '%s'\n", string1, string2);

            p = memmove(string1, string2, sizeof(string2));

           printf("After:    String 1 = '%s' String 2 = '%s'\n", string1, string2);

            while(1);
}
```

memset( )

#include <string.h>
char *memset(void *dest, int c, size_t count); 지정한 문자로 버퍼를 채웁니다.

Parameter

dest	dest 버퍼 포인터
c	버퍼를 채울 문자
count	문자 개수

Description

이 함수는 dest 버퍼에 지정한 숫자의 문자를 채웁니다.

Return Value

dest 버퍼의 시작 포인터를 반환합니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
             char buf[] = "0123456789";

            char *p;
             char t = 'A';

            _InitSerialPort();

             printf("Before: Buffer = '%s'\n", buf);

              p = memset(buf, t, 5);

             printf("After:    Buffer = '%s'\n", buf);

              while(1);
}
```

pow( ) powl( )

#include <math.h>

double pow(double x, double y)	배정도로 x 의 y 승을 계산합니다.
long double powl(long double x, long double y)	확장 배정도로 x 의 y 승을 계산합니다.

Parameter

x	부동 소수점 값
y	부동 소수점 값

Description

이 함수는 x 의 y 승 값을 계산합니다.

Return Value

이 함수는 x 의 y 승 값을 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{           float base;
             float power;
             float y;

           _InitSerialPort();

             base = 3.14;
              power = 2.14;

              y = pow (base, power);

             printf("Base = %f, Power = %f, Result = %f\n", base, power, y);

              while(1);
}
```

printf( )

#include <stdio.h>
int printf(const char *format, ...);

Parameter

format	포맷 제어 스트링
...	옵션 인수

Description

이 함수는 지정한 방법으로 데이터를 포맷하여 표준 출력장치로 출력합니다.

포맷 제어 스트링은 두개의 정보를 포함하고 있습니다.

- 정규문자
- 변환방법

정규문자는 단순히 표준 출력장치로 복사됩니다. 변환방법지정은 다음 인자에 의하여 변환되어 프린트 됩니다.

변환 형식은 '%' 로 시작하여 아래에 나열한 변환문자로 종료합니다. 제어 자수는 '%' 와 아래 나열된 변환문자 사이에 있습니다.

Control Characters:

마이너스 부호는 인수가 해당 영역에서 왼쪽으로 정열 되는 것을 의미합니다. .

숫자 스트링은 최소 필드 폭을 지정합니다. 변환되는 인수는 이 영역에서 최소 폭으로 프린트 됩니다. 변환 인수의 디지트 수가 지정한 영역보다 작을 때 오른쪽 또는 왼쪽으로 정열되어 프린트 됩니다. 첨가되는 문자는 공백 문자이거나 지정된 필드 폭 숫자의 처음이 0 이면 0 이 첨부됩니다.

마침표(소수점)은 다음 디지트 스트링으로 부터 필드 폭을 분리합니다.

정밀도를 나타내는 디지트 스트링은 스트링이 프린트 되는 최대 문자 수를 나타냅니다.

길이 수정자 l (문자 l)은 정수보다도 해당 아이템이 long 일 때 해당 데이터 항목을 의미합니다. 길이 수정자 L 은 해당 항목이 64 비트 배정도 부동소수점을 의미합니다.

printf 의 변환 문자는 다음과 같습니다.

d, i	int	부호부 십진수 표시
o	unsigned int	8 진법 표시 (선두의 0 은 생략됨)
x	unsigned int	헥사 데시멀 표시법 (선두의 0X 는 생략)
u	unsigned int	무 부호 십진수 표시
c	int	단일문자이며 unsigned char 로 변환
s	char*	스트링에서 Null 코드를 만나기 전까지 프린트 되는 문자 또는 정도 지정된 문자 수까지 프린트 되는 문자.
e,E	double	십진표시 [-]m.dddddE-xx(E 는 소문자 가능) d 는 정도표시. 디폴트 정도는 6 이며
f	double	[-]mmm.ddd 형식의 십진 표시 d 는 정밀도를 표시
g, G	double	%e 또는 %E 가 사용될 때 지수부가 -4 보다 작거나 정도와 같을 때 다른 경우는 %f 가 사용됩니다.

만약 '%'문자이후가 변환 문자가 아닐 때 이 문자는 표준 출력장치로 보내집니다. 이런 방법으로 '%%'로 표시하면 '%'가 표시됩니다.

Return Value

프린트 문자의 개수를 반환합니다.

Example

```
#include <stdio.h>

void _InitSerialPort(void);

main(){
        int m;
        long int n;
        float y;
        char t = 'A';
        char s[] = "Believe it or not.";

        _InitSerialPort();

         m = -1;
         /* -1, -1, 65535, 177777, ffff, FFFF */
         printf("%d, %i, %u, %o, %x, %X\n", m, m, m, m, m, m);

         y = -0.01234567;
         /* -0.012345 -1.234567-e2 -1.234567-E2 -1.234567-e2 -1.234567-E2 */
         printf("%f %e %E %g %G\n", y, y, y, y, y);

         /* A */
         printf("%c\n", t);

         /* Believe it or not. */
         printf("%s\n", s);

         /* [            Believe it or not.] [         Believe] */
         printf("[%25s] [%12.7s]\n", s, s);

         n = 12345678;
         /* [12345678] [    12345678] */
         printf("[%ld] [%10ld]\n", n,n);

         y = 123.4567;
         /* [       123.46] */
         printf("[%10.5f]\n", y);

        while(1);
}
```

putc( )

#include <stdio.h>
int putc(int c, FILE* stream);　　지정한 스트림으로 문자를 출력합니다.

Parameter

c	라이트될 문자
stream	FILE 스트럭쳐 포인터

Description

이 함수는 출력 스트림으로 한 개의 문자를 출력합니다. stdout 은 단지 출력 스트림을 지원합니다.

Return Value

출력된 문자를 반환합니다. EOF(-1)는 에러를 나타냅니다. EOF 는 정규 정수 값이므로 ferror()를 사용하여 error 가 발생한 것인지 확인하여야 합니다.

putch( )

#include <conio.h>
int putch(int ch);　　콘솔로 문자를 출력합니다.

Description

이 함수는 콘솔로 문자를 출력합니다. 이 루틴은 사용자의 하드웨어에 맞게 수정할 수 있습니다. 8051 시리얼 포트로 출력하는 라이브러리가 내장되어 있습니다. 특정용도로 사용하기 위하여 Consol I/O 를 참고하기 바랍니다.

Return Value

이 함수는 ch 를 반환합니다.

Example

```
#include <conio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;

      _InitSerialPort();

        for (c = 0x20; c < 0x7F; c++) {
         putch(c);
      }
         while(1);
}
```

putchar( )

#include <stdio.h>
int putchar(int c); stdout 으로 문자 출력

Description

이 함수는 출력 스트림 stdout 으로 한 개의 문자를 출력합니다.

putchar() 는 putc(c, stdout)으로 마크로 선언되어 있습니다.

Return Value

출력한 문자가 반환됩니다. EOF(-1)은 에러를 나타냅니다. EOF 는 정규 정수 값이므로 ferror()를 사용하여 error 가 발생한 것인지 확인합니다.

Example

```
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        unsigned char c;

      _InitSerialPort();

        for (c = 0x20; c < 0x7F; c++) {
        putchar(c);
      }
        while(1);
}
```

puts( )

#include <stdio.h>
int puts(const char* string); stdout 으로 스트링을 출력합니다.

Parameter
string 출력되는 스트링

Description
이 함수는 표준 출력장치 stdout 으로 스트링을 출력합니다. Null ('\0') 대신 개행 문자 ('\n')를 출력합니다.

Return Value
반환 값은 마지막인 개 행 문자가 됩니다. EOF(-1)은 에어를 의미합니다.

Example

```
#include <stdio.h>

void _InitSerialPort(void);

main()
{
            char str1[] = "1234567890";
            char str2[] = "abcdefghijklmnopqrstuvwxyz";

          _InitSerialPort();

            puts(str1);
            puts(str2);

             while(1);
}
```

rand( )

#include <stdlib.h>
int rand(void)

Description

이 함수는 0 - 32767 사이의 의사(pseudo) 랜덤 값을 반환합니다.

Return Value

0 - 32767 사이의 의사(pseudo) 랜덤 값을 반환합니다.

Example

```
#include <stdlib.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        int i, y;

      _InitSerialPort();

      for(i = 1; i<=10; i++){
          y = rand();
          printf("I = %d, RAND = %d\n", i, y)
      }

        while(1);
}
```

realloc( )

#include <malloc.h>
void realloc(void memblock, size_t size);　　메모리 블록을 재배치 합니다.

Parameter

memblock	선지정된 블록의 포인터
size	새로 지정되는 블록에 필요한 바이트 수

Description

이 함수는 먼저 정해진 메모리 블록을 크기를 변화하여 재설정합니다. 포인터 memblock 이 0 일 때 realloc()은 malloc()과 같이 동작하며 새로운 블록을 만듭니다. 다른 경우는 memblock 은 앞서 호출한 malloc()이나 realloc()의 포인터가 반환됩니다. 크기 인수는 블록의 새로 설정될 크기입니다. 블록의 내용은 변화하지 않습니다. 새 블록은 처음 블록과는 다른 위치이며 realloc()에 의하여 포인터가 반환되며 memblock 을 통하여 전달된 인수(포인터)와 다를 것입니다. 크기가 0 이고 memblock 이 NULL 이 아닌 경우 원래의 메모리 블록은 자유 영역이 됩니다.

Return Value

이 함수는 설정된 공간을 지시하는 void 포인터가 반환됩니다.
크기가 0 이고 메모리 블록이 NULL 이 아닐 때 또는 요청한 크기로 확장하는 것이 가능하지 않을 때 NULL 이 반환됩니다.

sbrk( )

#include <malloc.h>
char* sbrk(int n);　　메모리 브레이크 값

Description

이 함수는 nbytes 바이트 단위로 memory break 포인터를 증가 시킵니다. n 이 홀수이면 메모리 포인터가 모든 개체에 대하여 정확히 배열하기 위하여 짝수로 자리 올림 됩니다. sbrk()는 malloc()에 의하여 사용되며 사용자는 sbrk() 보다 malloc()을 사용하는 것이 바람직합니다.

Return Value

이 루틴은 옛날 브레이크 값을 반환합니다. 메모리가 충분치 않으면 -1 이 반환됩니다.

scanf( )

#include <stdio.h>
int scanf(const char* format, ...);

Parameter

format	포맷 제어 스트링
...	스캔 데이터의 인수

Description

이 함수는 표준입력 스트림 stdin 으로부터 포맷된 데이터를 읽어 숫자 값으로 변환합니다.

포맷 제어 스트링은 다음 형식의 개체를 포함합니다:

- 공백문자(blanks)나 탭(tabs)은 무시됩니다.
- 정규문자(% 제외)는 입력 스트림에서 다음 공백문자와 만날 때까지 입니다.
- 변환규정은 %를 포함하며 서프레션(suppression)문자 *는 옵션, 최대 필드 폭을 지정하는 숫자는 옵션, 타겟의 크기를 나타내는 h, l 또는 L 과 변환문자는 옵션 입니다.

변환규정은 다음 입력 필드에 변환방법을 지정합니다. 일반적으로 결과는 다음 인수에 의하여 지정된 변수에 지정됩니다.

정수 변환 문자 d, i, o, u 와 x 는 인수가 short integer 를 지시하는 포인터일 때 h 앞에 놓여 져야 하며 long integer 를 지시하는 포인터 l 앞에 놓여져야 합니다.

부동소수점 변환문자 e, f 와 g 는 인수가 double 을 지시하는 포인터일 때 l 앞에 놓여 져야 하며 인수가 long double 을 지시하는 포인터일 때 L 앞에 놓여져야 합니다.

변환문자는 다음과 같습니다:

문자	인수 형식	입력 데이터
d	int *	십진수
I	int *	정수, 숫자가 0 으로 시작되면 8 진수(octal) 0x 또는 0X 로 시작되면 16 진수입니다.
o	int *	8 진수(Octal)이며 0 으로 시작하든 아니든 관계없습니다.
x	int *	16 진수(Hexadecimal) 이며 0x 또는 0X 로 시작하든 아니든 관계없습니다.
u	unsigned int *	무부호 십진 정수
c	char *	문자. 다음의 x 입력 문자는 인수 배열에 있어야 합니다. x 는 필드 폭을 나타내는 숫자이거나 필드 폭을 지정하지 않을 때 1 입니다. 일잔 스킵 공백문자는 제한됩니다.
s	char *	스트링. 인수는 문자가 저장될 어레이를 지시하는 포인터 입니다.
e, f, g	float *	부호와 숫자로 구성된 가수부 그리고 부호와 정수로 구성된 지수부 형식의 부동 소수점

Return Value

이 함수는 성공적으로 변환하여 값을 저장하였을 때 필드 수를 반환합니다.

sin( ) sinl( )

#include <math.h>
double sin(double x)　　배정도 정수로 sine 값을 계산합니다.

Parameter
x　　라디안 각도 값이며 부동소수점 숫자형식

Description
이 함수는 x 의 sine 값을 계산합니다.

Return Value
이 함수는 전달된 파라메터 값의 sine 값을 계산하여 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

#define pi 3.141592

void _InitSerialPort(void);

main()
{
        int i;
        float x, y;

        _InitSerialPort();

        for ( i = 0 ; i <= 90; i += 5) {
            x = pi * i / 180;                          // Degree to Radian
            y = sin( x );
            printf("Rad = %f SIN(Deg %d) = %f\n", x, i, y);
        }
        while(1);
}
```

sinh( ) sinhl( )

#include <math.h>

double sinh(double x)	배정도로 hyperbolic sine 값을 계산합니다.
long double sinhl(long double x)	확장 배정도 형식으로 hyperbolic sine 값을 계산합니다.

Parameter

x	라디안으로 표시된 부동소수점 값

Description

이 함수는 x 의 hyperbolic sine 값을 계산합니다.

Return Value

이 함수는 전달된 파라메터 값의 hyperbolic sine 값을 계산하여 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>
#define pi 3.141592

void _InitSerialPort(void);

main()
{
        float x, y;

        _InitSerialPort();

        for ( x = 0 ; x < (2 * pi); x += 0.1) {
            y = sinh (x);
            printf("SINH(%f) = %f\n", x, y);
        }
        while(1);
}
```

sprintf( )

#include <stdio.h>
int sprintf(char* buffer, const char* format, ...

Parameter

buffer	출력을 위한 저장 위치
format	포맷제어 스트링
...	변환되어 라이트 되는 선택적인 인수

Description

이 함수는 인수를 변환하여 형식에 맞추어 버퍼에 씁니다.

포맷제어 스트링은 printf() 함수와 동일합니다.

Return Value

버퍼에 쓰여진 문자 수를 반환합니다.

Example

```
#include <stdio.h>

char    buff[200];

void _InitSerialPort(void);

main(){
            int m;
            long int n;
            float y;
            char t = 'A';
            char s[] = "Believe it or not.";

            _InitSerialPort();

              m = -1;
              /* -1, -1, 65535, 177777, ffff, FFFF */
              sprintf(buff, "%d, %i, %u, %o, %x, %X\n", m, m, m, m, m, m);
              printf("%s", buff);

              y = -0.01234567;
              /* -0.012345 -1.234567-e2 -1.234567-E2 -1.234567-e2 -1.234567-E2 */
              sprintf(buff, "%f %e %E %g %G\n", y, y, y, y, y);
              printf("%s", buff);

              /* A */
              sprintf(buff, "%c\n", t);
              printf("%s", buff);

              /* Believe it or not. */
              sprintf(buff, "%s\n", s);
              printf("%s", buff);

              /* [            Believe it or not.] [         Believe] */
              sprintf(buff, "[%25s] [%12.7s]\n", s, s);
              printf("%s", buff);

              n = 12345678;
              /* [12345678] [    12345678] */
              sprintf(buff, "[%ld] [%10ld]\n", n,n);
              printf("%s", buff);

              y = 123.4567;
              /* [       123.46] */
              sprintf(buff, "[%10.5f]\n", y);
              printf("%s", buff);

            while(1);
}
```

sqrt( ) sqrtl( )

#include <math.h>

double sqrt(double x)	배정도로 평방근(square root)을 계산합니다.
long double sqrt(long double x)	확장 배정도롤 평방근(square root)을 계산합니다.

Parameter

x	부동 소수점 값

Description

이 함수는 x 의 평방근(square root)을 계산합니다. x 가 0 보다 작으면 결과 값은 예측할 수 없습니다.

Return Value

x 의 평방근(square root) 값을 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        float x, y;

       _InitSerialPort();

        x = 1000;
        y = sqrt(x);

        printf("SQRT(%f) = %f\n", x, y);

        while(1);
}
```

srand( )

#include <stdlib.h>
void srand(unsigned int seed)　　시작 값에 의한 난수를 발생합니다.

Parameter
seed　　rand() 에 필요한 시작 값

Description
이 함수는 의사(pseudo) 난수 발생함수의 시작 값을 설정합니다. 시작 값은 연속되는 난수발생시 시작 값으로 사용됩니다. 초기 시작 값은 1 입니다.

Return Value
반환되는 것은 없습니다.

Example

```
#include <stdlib.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        int i, y;

      _InitSerialPort();

      for(i = 1; i<=10; i++){
          srand(i);
          y = rand();
          printf("Seed = %d, RAND = %d\n", i, y);
      }

        while(1);
}
```

sscanf( )

#include <stdio.h>
int sscanf(char* buffer, const char* format, ...);

Parameters

buffer	입력데이터를 지시하는 포인터
format	포맷제어 스트링
...	수신할 데이터의 인수

Description

이 함수는 스트링 버퍼로부터 포맷된 데이터를 읽습니다. 그리고 변환된 값을 포인터로 표시된 인수에 저장합니다. 포맷이 더 이상 없을 때 함수는 리턴됩니다.
포맷 제어 스트링은 scanf() 함수의 포맷인자와 같은 형식입니다.

Return Value

성공적으로 변환하여 저장을 마친 경우 필드 수를 반환합니다.

strcspn( )

#include <string.h>
size_t strcspn(const char* string1, const char* string2);

Parameter

string1	스캔 대상 스트링
string2	문자 셋 스트링

Description

이 함수는 스트링 1 에서 스트링 2 의 첫번째 보다 앞에 있는 문자 수를 측정합니다. 종료문자 null 은 포함하지 않습니다.

두개의 string1 과 string2 는 반드시 null 로 종료되어 있어야 합니다.

Return Value

스트링 2 의 모든 문자에 대하여 스트링 1 에서 일치하는 문자까지의 길이(스티링 1 의 처음부터 일치하는 문자까지)를 반환합니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[] = "abcdefghijklmnopqrstuvwxyz";
        char str2[] = "pl";
        int i;

        _InitSerialPort();

        i = strcspn(str1, str2);
        /* First character location = 11 */
        printf ("First character location = %d\n", i);

        while(1);
}
```

strcat( )

#include <string.h>
char *strcat(char* string1, const char* string2); 스트링을 다른 스트링과 결합합니다.

Parameter

string1	확장할 스트링
string2	붙일 스트링

Description

이 함수는 string1 뒤에 string2 를 결합하여 string1 으로 확장합니다.

string1 과 string2 는 null 로 종료되어 있어야 합니다.

Return Value

string1 의 포인터가 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        char str1[30] = "My name is ";
        char str2[] = "Stanford";

      _InitSerialPort();


        strcat(str1, str2);
        printf("New string = %s\n", str1);

        while(1);
}
```

strchr( )

#include <string.h>
char *strchr(const char* string, char c); 스트링에서 c 와 처음 일치하는 곳을 찾습니다.

Parameter

string	탐색 스트링
c	탐색할 문자

Description

이 함수는 string 에서 첫번째 일치하는 문자 c 의 위치를 찾습니다. 탐색 스트링은 null 문자('\0')로 종료된 것입니다.

Return Value

스트링에서 문자 c 와 첫번째 일치하는 포인터 또는 발견하지 못했을 때 NULL 이 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main()
{
        static char str1[] = "abcdefghijklmnopqrstuvwxyz";

        char c = 'm';
        char *s;

        _InitSerialPort();

        s = strchr(str1, c);
        if (s != NULL)
        printf ("Found '%c' at %s\n", c,s);

        while(1);
}
```

strcmp( )

#include <string.h>

string1	첫번째 스트링의 포인터
string2	두번째 스트링의 포인터

Description

이 함수는 사전 편찬학적으로 string1 과 string2 를 비교합니다.

string1 과 string2 는 null 로 종료되어 있어야 합니다.

Return Value

Value	Meanig
Less than 0	string1 < string2
0	string1 = string2
Greater than 0	string1 > string2

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[] = "abcdefgh";
        char str2[] = "ABCDabcd";
        char str3[] = "abcd1234";
        char a, b, c;

        _InitSerialPort();

        a = strcmp(str1, str2);
        b = strcmp(str2, str3);
        c = strcmp(str3, str1);

        printf ("String1-String2 = %d\n", a);
        printf ("String2-String3 = %d\n", b);
        printf ("String3-String1 = %d\n", c);

        while(1);
}
```

strcpy( )

#include <string.h>
char *strcpy(char* string1, const char* string2); 스트링을 다른 스트링에 복사합니다.

Parameter

string1	목표 스트링
string2	소스 스트링

Description

이 함수는 string2 를 sting1 에 복사합니다.

string1 과 string2 는 null 로 종료되어 있어야 합니다.

Return Value

string1 의 포인터가 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[30] = "My name is ";
        char str2[] = "Stanford";

      _InitSerialPort();

        strcpy(str1, str2);
        printf("New string = %s\n", str1);

        while(1);
}
```

strlen( )

#include <string.h>
size_t strlen(const char* string)　　스트링의 길이를 측정합니다.

Parameter
string　　측정할 스트링

Description
이 함수는 스트링의 문자 개수를 측정합니다. 그러나 Null 문자는 포함되지 않습니다.

Return Value
스트링의 계수 된 문자 수를 반환합니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[] = "0123456789";
        char str2[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        char str3[] = "a";

        int a, b, c;

       _InitSerialPort();

        a = strlen(str1);
        b = strlen(str2);
        c = strlen(str3);

        printf("String 1 Length = %d\n", a);
        printf("String 2 Length = %d\n", b);
        printf("String 3 Length = %d\n", c);

        while(1);
}
```

strncat( )

#include <string.h>

char *strncat(char* string1, const char* string2, size_t n); 하나의 스트링을 다른 스트링에 이어 붙입니다.

Parameter

string1	확장될 스트링
string2	이어질 스트링
n	이어지는 문자 개수

Description

이 함수는 string2 의 n 개의 문자를 string1 의 끝에 이어 붙입니다. 만약 n 이 string2 보다 크면 string2 의 문자길이가 n 대신 사용됩니다.

string1 과 string2 는 null 로 종료되어 있어야 합니다. string1 은 Null 종료됩니다.

Return Value

string1 의 포인터가 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[30] = "My name is ";
        char str2[] = "OliverStanford";

        _InitSerialPort();

        strncat(str1, str2, 6);
        printf("New string = %s\n", str1);

        while(1);
}
```

strncmp( )

#include <string.h>

char *strncmp(const char* string1, const char* string2, size_t n);

스트링을 다른 스트링과 비교한다.

Parameter

string1	첫번째 스트링의 포인터
string2	두번째 스트링의 포인터
n	비교할 문자수

Description

이 함수는 string2 의 n 개 부분과 string1 을 사전 나열식 방법으로 비교합니다.

Return Value

반환값	의미
Less than 0	서브 string1 < 서브 string2
0	서브 string1 = 서브 string2
Greater than 0	서브 string1 > 서브 string2

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[] = "abcdefgh";
        char str2[] = "ABCDabcd";
        char str3[] = "abcd1234";
        char a, b, c;

        _InitSerialPort();

        a = strncmp(str1, str2, 4);
        b = strncmp(str2, str3, 4);
        c = strncmp(str3, str1, 4);

        printf ("String1-String2 = %d\n", a);
        printf ("String2-String3 = %d\n", b);
        printf ("String3-String1 = %d\n", c);

        while(1);
}
```

strncpy( )

#include <string.h>

char *strncpy(char* string1, const char* string2, size_r n);

스트링을 다른 스트링으로 복사합니다.

Parameter

string1	목표 스트링
string2	소스 스트링
n	복사할 문자 개수

Description

이 함수는 string2 의 n 개의 문자를 string1 으로 복사합니다. 만약 n 이 string2 의 길이보다 큰 경우에 string2 이 string 1 에 복사된 후 null 문자('\0')를 뒤이어 채워 개수 n 을 맞춥니다.

Return Value

string1 의 포인터가 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[30] = "My name is ";
        char str2[] = "OliverStanford";
        char *s;

        _InitSerialPort();

        s = strncpy(str1, str2, 6);
        printf("New string = %s\n", s);

        while(1);
}
```

strpbrk( )

#include <string.h>
char *strpbrk(const char* string1, const char* string2);

Parameter

string1	소스 스트링
string2	문자 셋

Description

이 함수는 string1 에서 string2 의 어떤 문자든 처음으로 일치하는 것을 찾습니다. 탐색 문자에 Null 문자는 포함되지 않습니다.

Return Value

이 함수는 string1 에서 첫번째 일치한 포인터를 반환합니다. Null 문자가 반환된 것은 2 개의 스트링이 일치하는 것이 없는 것을 의미합니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        char str1[] = "Oliver Stanford";
        char str2[] = "My name is ";
        char *s;

        _InitSerialPort();

        s = strpbrk(str1, str2);

        printf("%s\n", s);

        while(1);
}
```

strrchr( )

#include <string.h>

char *strrchr(const char* string, char c);	스트링에서 문자가 마지막으로 일치한 위치를 찾습니다.

Parameter

string	스트링
c	탐색 문자

Description

이 함수는 스트링에서 마지막으로 일치한 문자 c 의 위치를 찾습니다. 탐색할 스트링은 null 문자('\0')로 종료되어 있어야 합니다.

Return Value

스트링에서 문자 c 와 일치한 포인터 또는 찾지 못했을 때 Null 이 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        char str[] = "My name is Oliver Stanford";
        char c = 'n';
        char *s;
        int i;

        _InitSerialPort();

        s = strrchr(str, c);

        printf("%s\n", s);

        while(1);
}
```

strspn( )

#include <string.h>
size_t strspn(const char* string1, const char* string2);

Parameter

string1	스캔 스트링
string2	문자셋 스트링

Description

이 함수는 스트링 1 에 대하여 스트링 2 의 어떤 문자이든지 탐색하여 일치하는 문자까지의 개수를 계산합니다. Null 종료 문자는 탐색에 포함되지 않습니다. 스트링 1 과 스트링 2 는 Null 문자로 종료되어 있어야 합니다.

Return Value

이 함수는 스트링 1 에 대하여 스트링 2 의 어떤 문자이든지 일치하는 문자까지의 개수(스트링 1 의 처음부터 일치하는 문자까지의 갯수)를 반환합니다. 일치하는 문자가 없으면 0 이 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[] = "abcdefghijklmnopqrstuvwxyz";
        char str2[] = "yes";

        int i;

        _InitSerialPort();

        i = strspn(str1, str2);

       /* First character location = 4 */
        printf ("First character location = %d\n", i);

        while(1);
}
```

strstr( )

#include <string.h>
char *strstr(const char *string1, const char* string2); string1 에서 string2 를 찾습니다.

Parameter

string1	스캔할 스트링
string2	찾을 스트링

Description

이 함수는 string1 에서 string2 가 첫번째로 일치하는 위치를 찾습니다.

Return Value

이 함수는 string1 에서 string2 를 찾아 포인터를 반환합니다. 그렇지 않으면 Null 이 반환됩니다.

Example

```
#include <string.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){
        char str1[] = "Believe it or not.";
        char str2[] = "or";

        char *s;

        _InitSerialPort();

        s = strstr(str1, str2);
        printf ("%s\n", s);

        while(1);
}
```

tan( ) tanl( )

#include <math.h>

double tan(double x)	배 정도 부동소수점으로 tangent 값을 계산합니다.
long double tan(long double x)	확장 배정도 부동소수점으로 tangent 값을 계산합니다.

Parameter

x	라디안으로 표시되는 부동소수점 각도

Description

이 함수는 전달된 파라메터의 tangent 값을 계산합니다.

Return Value

전달된 파라메터의 tangent 값을 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

#define pi 3.141592

void _InitSerialPort(void);

main(){
        int i;
         float x, y;

         _InitSerialPort();

         for ( i = 0 ; i <= 80; i += 10) {
              x = pi * i / 180;                                    // Degree to Radian
              y = tan( x );
              printf("Rad = %f TAN(Deg %d) = %f\n", x, i, y);
         }
         while(1);
}
```

tanh( ) tanhl( )

#include <math.h>

double tanh(double x)	배정도 부동소수점으로 hyperbolic tangent 를 계산합니다.
long double tanh(double x)	확장 배정도 부동소수점으로 hyperbolic tangent 를 계산합니다.

Parameter

x	라디안으로 표시되는 부동소수점 각도

Description

이 함수는 hyperbolic tangent 값을 계산합니다.

Return Value

이 함수는 전달된 파라메터를 hyperbolic tangent 값을 계산하여 반환합니다.

Example

```
#include <math.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        float x, y, pi;

        _InitSerialPort();

        pi = 3.141592;

        for (x = -(pi/4); x < (pi/4); x += 0.1) {
            y = tanh (x);
            printf("TANH(%f) = %f\n", x, y);
        }

        while(1);
}
```

time( )

#include <time.h>
time_t time(time_t *timeptr);

Parameter

timeptr	time 을 저장할 포인터

Description

이 함수는 현재의 칼렌더(달력)를 반환합니다. 유효하지 않으면 -1 을 반환합니다.

Return Value

이 함수는 시계가 유효하지 않을 때 -1 을 반환합니다.

toascii( )

#include <ctype.h>
int toascii(int c);　　문자를 Ascii 로 변환합니다.

Parameter
c　　변환될 문자

Description
이 함수는 c 의 하위 7 비트를 0 으로 만들고 나머지는 모두 세트합니다. 변환된 문자는 ASCII 문자 셋이 됩니다.

Return Value
변환된 문자를 반환합니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        char c = 82;
        char k;

        _InitSerialPort();

        k = toascii(c);

        printf("%c is an ASCII Character\n", k);

        while(1);
}
```

tolower( )

#include <ctype.h>
int tolower(int c);　　문자를 소문자로 변환합니다.

Parameter
c　　변환될 문자 c

Description
이 함수는 문자가 대문자일 때 소문자로 변환합니다. 그렇지 않으면 변화되지 않습니다.

Return Value
새로운 값 c 를 반환합니다. 대문자일 때 변환되며 그렇지 않으면 변환되지 않습니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        unsigned char i, j;

        _InitSerialPort();

        for(i = 0x20; i < 0x7F; i++) {
            j = tolower(i);
            printf("tolower(%c) = %c\n", i, j);
        }
        while(1);
}
```

toupper( )

#include <ctype.h>
int toupper(int c);　　문자를 대문자로 변환합니다.

Parameter
c　　변환될 문자

Description
이 함수는 문자 c 가 소문자일 때 대문자로 변환합니다. 그렇지 않으면 c 가 변화되지 않습니다.

Return Value
문자가 소문자일 때 대문자로 변환되어 반환됩니다. 그렇지 않으면 변화되지 않습니다.

Example

```
#include <ctype.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        unsigned char i, j;

        _InitSerialPort();

        for(i = 0x20; i < 0x7F; i++) {
            j = toupper(i);
            printf("tolower(%c) = %c\n", i, j);
        }
        while(1);
}
```

ultoa( )

#include <stdlib.h>

char *ultoa(unsigned long number, char* string, int radix); 무 부호 롱 정수를 스트링으로 변환합니다.

Parameter

number	변환될 정수
string	저장될 스트링 포인터
radix	밑 수

Description

이 함수는 무 부호 롱 정수를 Null 문자가 포함된 문자 스트링으로 변환합니다. 결과 스트링의 길이는 숫자 값과 밑 수에 따라 다릅니다.

Return Value

이 함수는 스트링의 포인터를 반환합니다.

Example

```
#include <stdlib.h>
#include <stdio.h>

void _InitSerialPort(void);

main(){

        char str[20] ="";
        unsigned long int i;
        char *s;

        _InitSerialPort();

        i = 1234567890;

        s = ultoa(i, str, 10);

        printf("%s\n", s);

        while(1);
}
```

ungetc( )

#include <stdio.h>
int ungetc(int c, FILE* stream);　　문자를 스트림으로 되돌립니다.

Parameter

c	되돌릴 문자
stream	FILE 스트럭쳐의 포인터

Description

이 함수는 문자 c 를 무부호 문자로 변환하여 입력 스트림으로 반환합니다. 이 문자를 다음입력에서 읽습니다. 단지 한 개의 문자가 되돌려지는 것을 허용합니다. EOF 는 반환되지 않습니다.

Return Value

이 함수는 되돌려진 문자나 에러가 발생한 경우 EOF 가 반환됩니다.

ungetch( )

#include <stdio.h>

int ungetch(int c); stdin 으로 문자를 반환합니다.

Parameter

c 반환될 문자

Description

이 함수는 문자 c 를 unsigned 문자로 변환하여 stdin 으로 되돌립니다. stdin 으로 부터 다음번 입력시 이 문자를 읽습니다. 단지 1 개의 문자가 반환되는 것을 허용합니다. EOF 는 반환할 수 없습니다. 이 함수는 반환된 문자나 에러가 생길 때 EOF 가 반환됩니다.

Return Value

이 함수는 되돌린 문자 또는 에러발생시 EOF 를 반환합니다.

CROSSWARE 8051 ANSI - C 컴파일러
QUICK START

Crossware 8051 Development Suit 에 포함된 간단한 예제를 실행합니다. Debug 관련 아이콘과 시뮬레이터 사용법에 관하여 설명합니다.

예제 1: 새 프로젝트의 구성법
새 프로젝트를 구성하는 방법에 관하여 설명되어 있습니다. 예제 프로그램은 CYGNAL C8051F020 프로세서의 64 개의 입출력 포트에 연결된 LED 를 순차적으로 ON/OFF 합니다.

예제 2: 부동소숫점, 스트링, Structure, Union 변수, 지역, 전역 변수의 디버깅
예제 프로그램은 Crossware ANSI C 컴파일러에서 지원하는 변수 형식과 연산자의 사용법을 보여 줍니다. 비트변수, 정수변수, 부동소숫점, 스트링 그리고 Structure 변수 디버깅과 Union 변수를 이용하여 16 비트 정수를 상위바이트 하위바이트로 나누어 P1, P2 에 출력하는 방법에 대하여 설명합니다.

예제 3: 타이머 인터럽트의 사용
정해진 시간 주기에 프로그램을 시행할 때 타이머 인터럽트를 사용합니다. 리니어 로터리 엔코더에서 출력되는 A, B 상 신호를 소프트웨어적인 방법으로 분해하여 8 단 7 세그멘트 FND 에 다이나믹 방식으로 디스프레이하는 엔코더 카운터를 제작합니다.

예제 4: 시리얼 포트를 이용한 데이터 입출력 실험
인터페이스 방식으로 가장 많이 사용되는 RS-232 시리얼 인터페이스를 이용하는 방법에 관하여 설명합니다. PC 의 시리얼 포트(Com1 포트) 와 연결하여 데이터를 주고 받는 실험입니다. Crossware 의 시뮬레이터를 이용하여도 데이터 입출력 실험이 가능합니다.

예제 5: 인터럽트 INT0 와 INT1 의 사용법
인터럽트의 개념에 대하여 설명합니다. 인터럽트는 LEVEL 인터럽트 방시과 EDGE 인터럽트 방식이 있으며 이 두가지 방식에 대한 분명한 차이점과 8051 마이크로 프로세서에서 외부 Port 핀 입력을 이용한 인터럽트 사용법이 설명되어 있습니다.

Introduction

Installation / 소프트웨어 설치

CROSSWARE 8051 Develeopment Suit 의 설치는 CD 의 setup.exe 를 실행하면 됩니다. 설치가 완료되면 Windows 바탕화면에 아이콘이 만들어집니다. 프로그램 설치 후 최초로 실행 시 그림 1 과 같이 됩니다.

Development Studio 에 포함된 예제 프로그램 실행

프로그램 설치 후에 그림 1 과 같은 예제 프로젝트 윈도우가 나타납니다. 메뉴에서 C8051F000 Example: Crossbar 를 선택합니다. 이 프로젝트는 CYGNAL 사에서 제작한 C8051F000 Development Kit 에서 실행하기 위한 프로그램이며 C8051F000 의 P1.6 에 부착된 LED 를 점멸하는 예제 입니다.

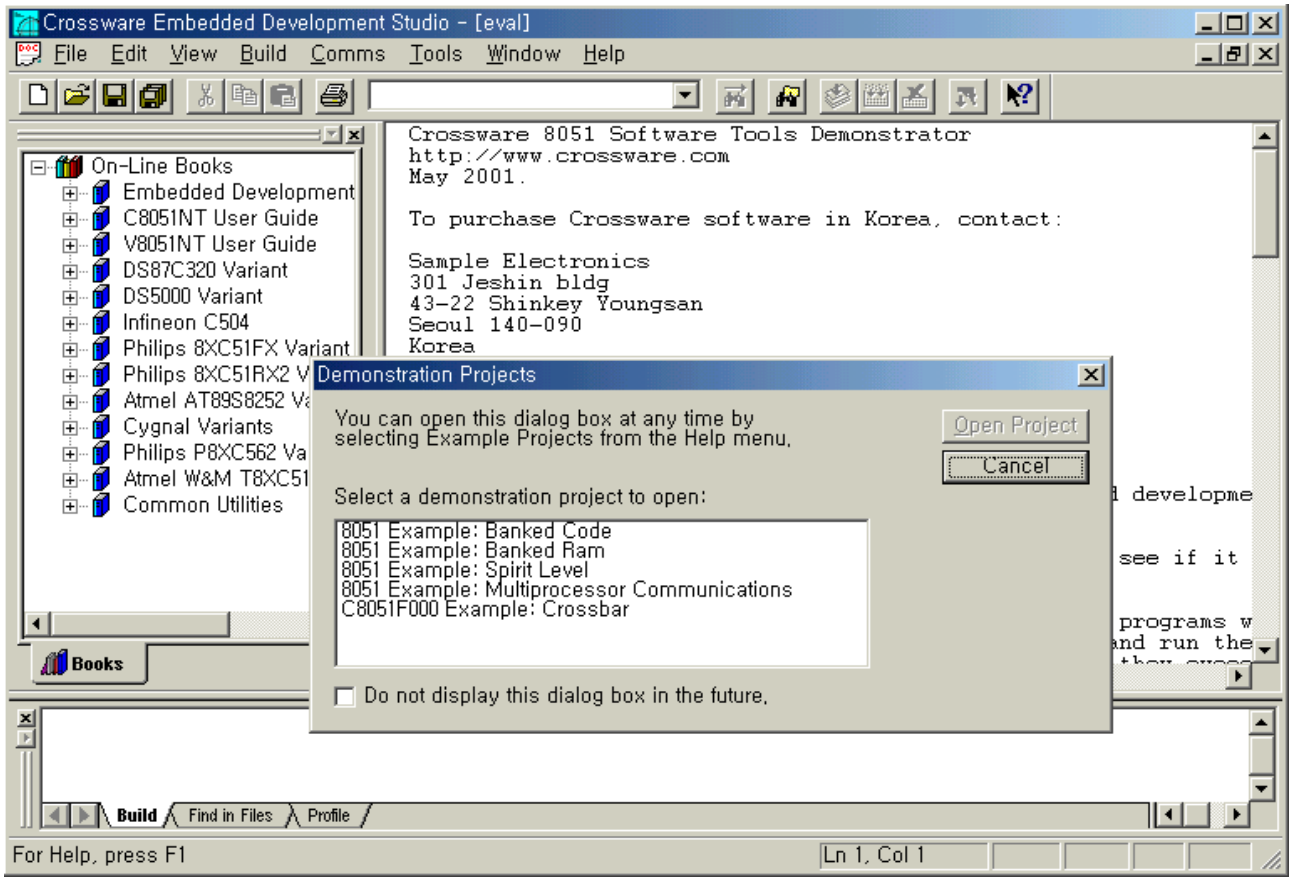


그림 1: 프로그램 설치 후 최초 화면

예제 프로젝트 crossbar 를 선택한 후 그림 3 과 같이 Workspace 윈도우와 Output 윈도우를 오픈합니다. Workspace 는 Studio 의 왼쪽에 위치한 윈도우입니다. 프로젝트에 포함된 파일을 트리 구조로 보여주며 파일을 마우스로 클릭하면 편집 윈도우에 파일이 오픈되어 편집가능 상태로 됩니다. Output 윈도우는 Studio 의 아래쪽에 나타나며 컴파일 결과를 출력하여 보여주는 윈도우 입니다. 그림 4 은 Workspace 에서 main.c 를 클릭하여 나타난 main.c 프로그램 본문입니다. 이 프로그램은 하드웨어에서 실행할 때 LED 의 반짝임 상태가 적당하도록 되어 있습니다. 시뮬레이션 모드에서 P1.6 상태를 확인하기 위하여 윈도우 편집화면에서 **DELA_CONST 10000** 을 **10** 으로 수정합니다. 그리고 그림 5 와 같이 윈도우 메뉴바의 Build-> Rebuild All 을 클릭하면 컴파일이 시작됩니다. Output 윈도우에 “Build Operation Complited”가 나오면 정상적으로 컴파일 되었음을 나타낸 것 입니다. 컴파일 완료한 예제프로그램을 시뮬레이터에 의하여 확인하려면 윈도우 메뉴바의 시뮬레이터/디버거 메뉴를 사용합니다. Crossware 의 8051 Development Studio 는 프

로그램이 “Go”  , “Trace”  , “Step to Cursor”  , “Step”  , Step Over  아이콘을 클릭하면 시뮬레이터/디버그 모드로 되며 8051 의 포트, 레지스터, 메모리, 변수 값을 확인할 수 있는 기능이 지원됩니다. 이 프로그램은 C8051F000 프로세서로 설정되어 있습니다. 그러므로 하드웨어 디버깅 모드를 사용할 때 다른 프로세서가 연결되어 있다면 경고 메시지 윈도우가 나타납니다. 이때에는 그림 34, 그림 35 와 같이 프로세서의 종류를 다시 선택하여 컴파일한 후 다운로드 하여야 합니다. **Development Studio** 의 시뮬레이션/디버깅 관련 기능은 그림 2 와 같습니다.

icon		설명
	**Restart**	프그그램의 실행 위치를 처음부터 시작합니다.
	**Undo Command**	프로그램의 실행이 정지된 상태에서 명령의 실행을 취소합니다.
	**Undo Instruction**	명령의 실행을 취소합니다.
	**Go**	현재의 프로그램을 실행합니다.
	**Trace**	현재의 프로그램을 1 라인씩 실행하며 실행 후 모든 결과 값을 표시합니다.
	**Step to Cursor**	현재 커서위치까지 프로그램을 실행 후 정지합니다.
	**Step**	1 개의 라인을 실행합니다.
	**Step Over**	실행 문이 서브루틴을 호출할 경우 (call) 서브루틴을 실행합니다.
	**Step Out**	서브루틴 실행 중 정지하여있을 때 서브루틴 끝까지 실행 호출한 곳으로 돌아가서 정지합니다.
	**Halt**	프로그램의 실행을 정지합니다.
	**Close**	프로그램의 시뮬레이션 / 디버깅을 종료합니다.
	**Capture Current State**	실행중인 현재상태의 PC (Program Counter)와 사이클(1 개의 어셈블리 명령 실행) 수를 표시합니다.
	**Set Break Point at Cursor**	현재 커서 위치에 브레이크 포인터를 설정합니다.
	**Clear All Break Point**	모든 브레이크 포인터를 해지 합니다.
	**Trigger External Interrupt 0**	소프트웨어 시뮬레이션 실행 시 INT0 에 해당하는 신호를 발생하여 인터럽트 서브루틴을 실행합니다.
	**Trigger External Interrupt 1**	소프트웨어 시뮬레이션 실행 시 INT1 에 해당하는 신호를 발생하여 인터럽트 서브루틴을 실행합니다.
	**Compile Assemble**	소스 프로그램을 컴파일/어셈블링 합니다.
	**Build**	소스 프로그램을 컴파일/어셈블링한후 링크하여 최종 출력파일(Hex, IEEE695)을 만듭니다.
	**Stop Build**	컴파일을 중단합니다.

그림 2: Development Studio 의 시뮬레이터/디버그 관련 아이콘

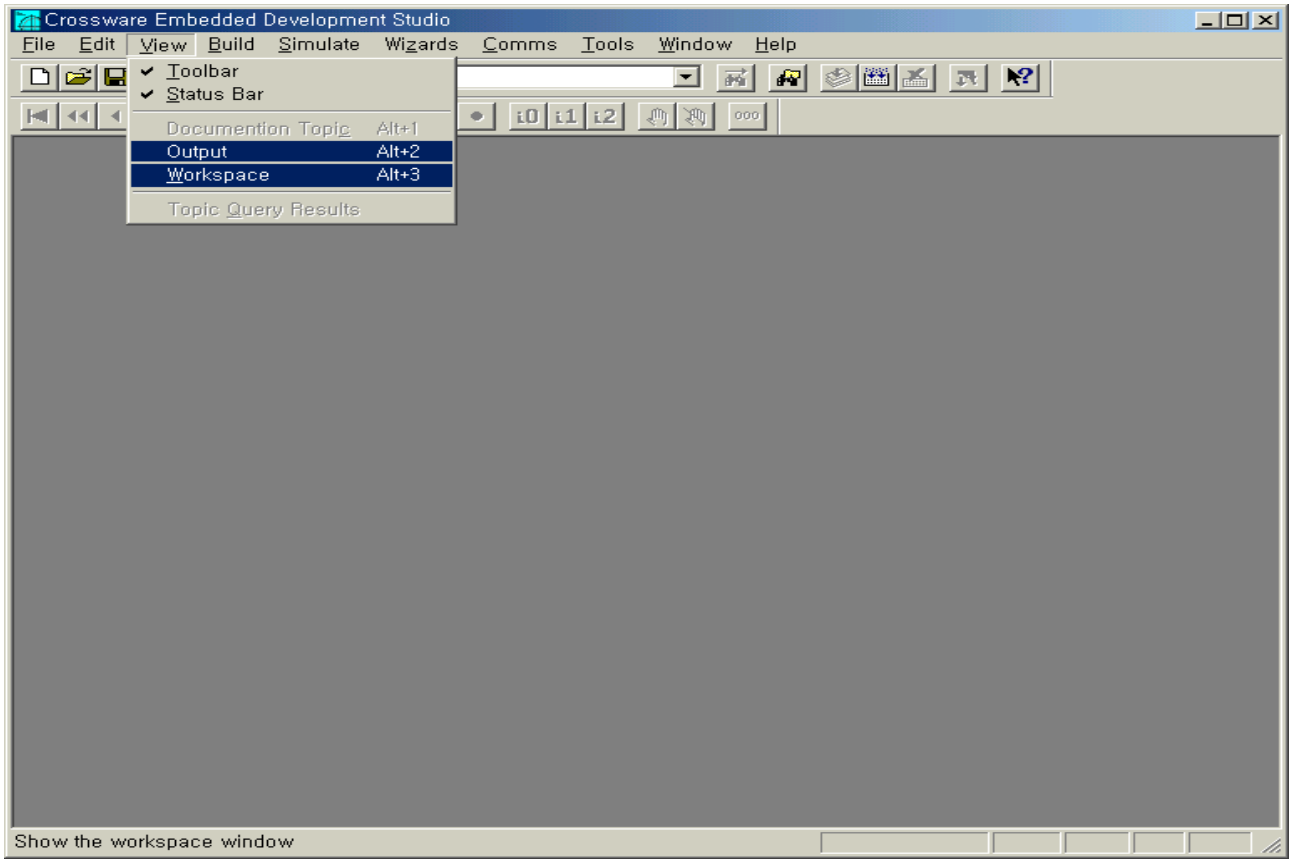


그림 3: Output 윈도우와 Workspace 윈도우 오픈

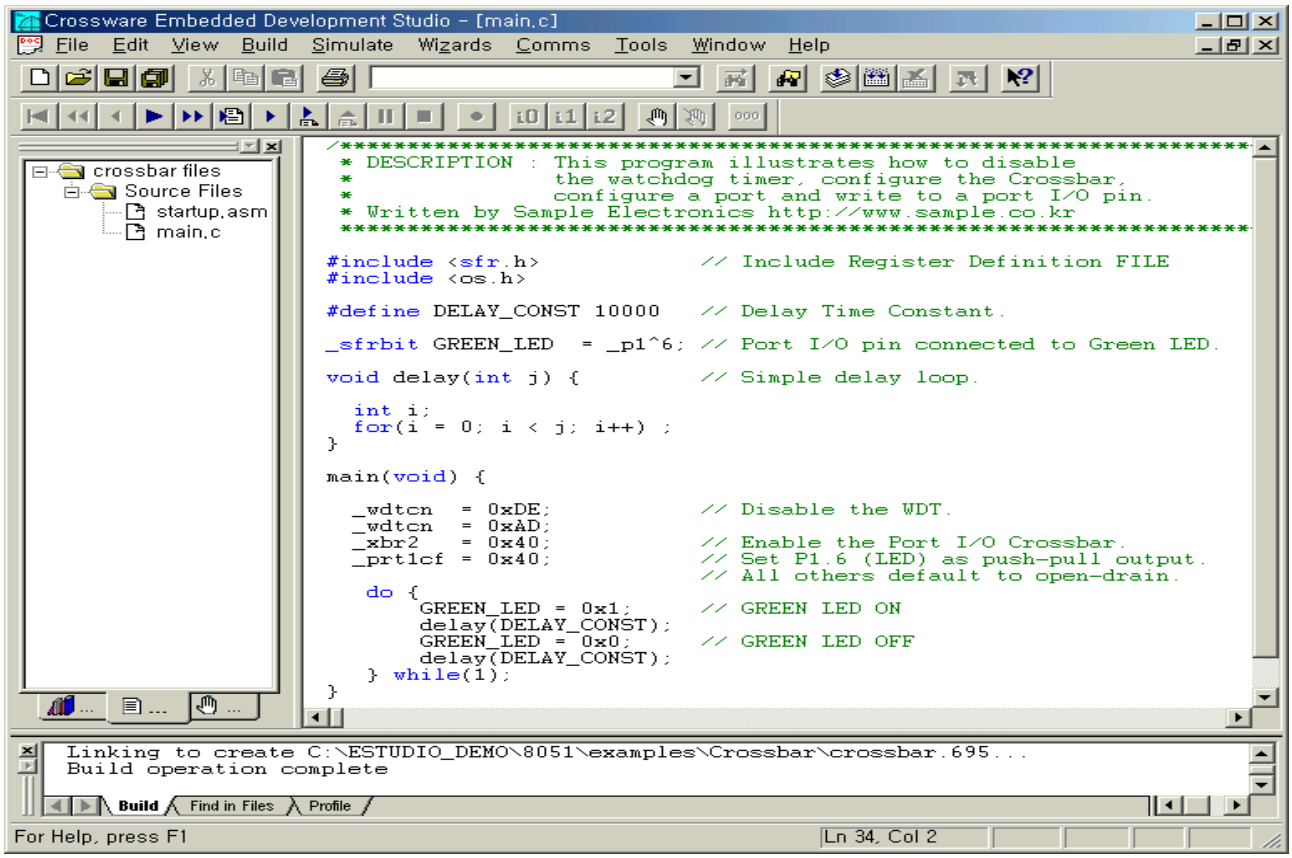


그림 4: 예제 프로그램 crossbar 의 main 함수

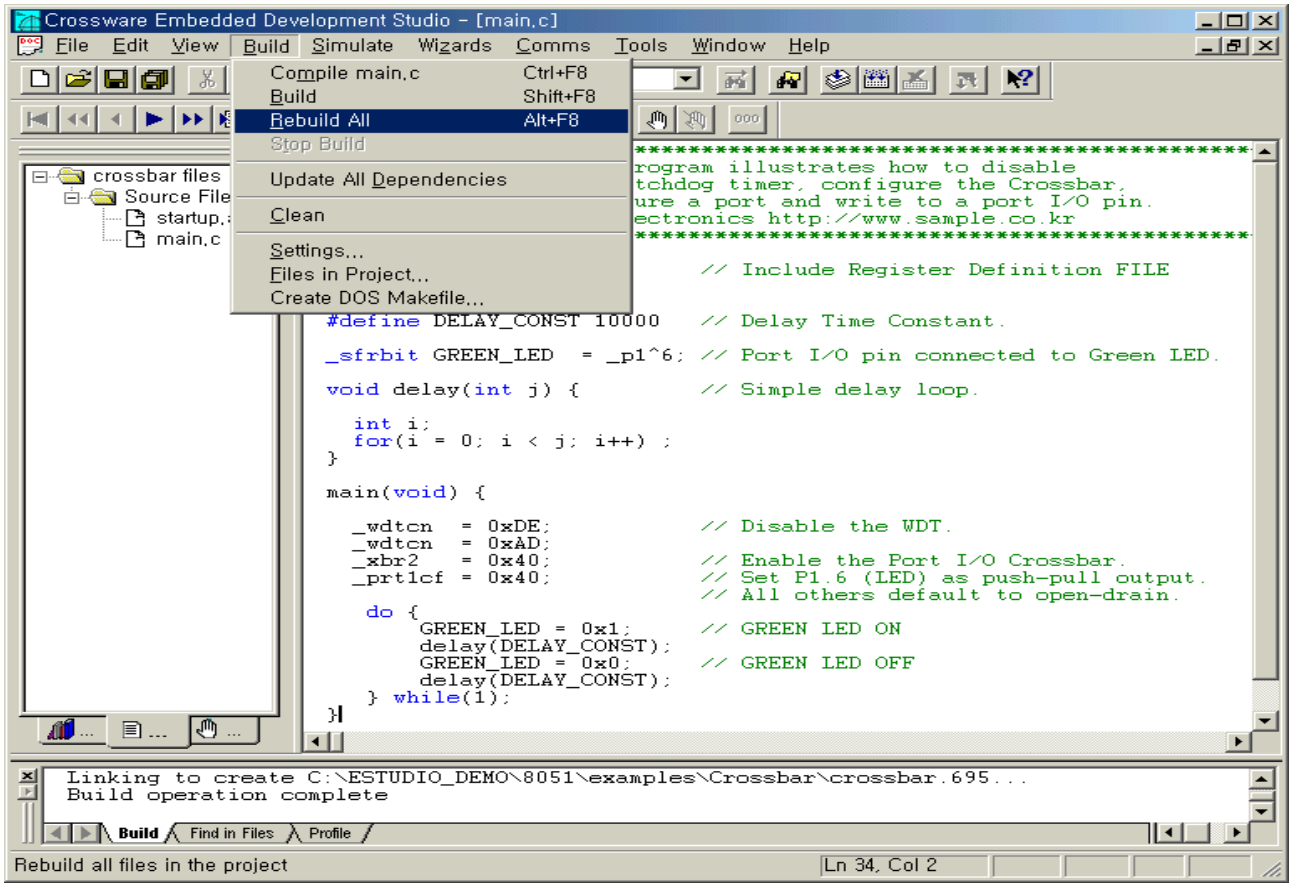


그림 5: 예제 프로그램의 컴파일

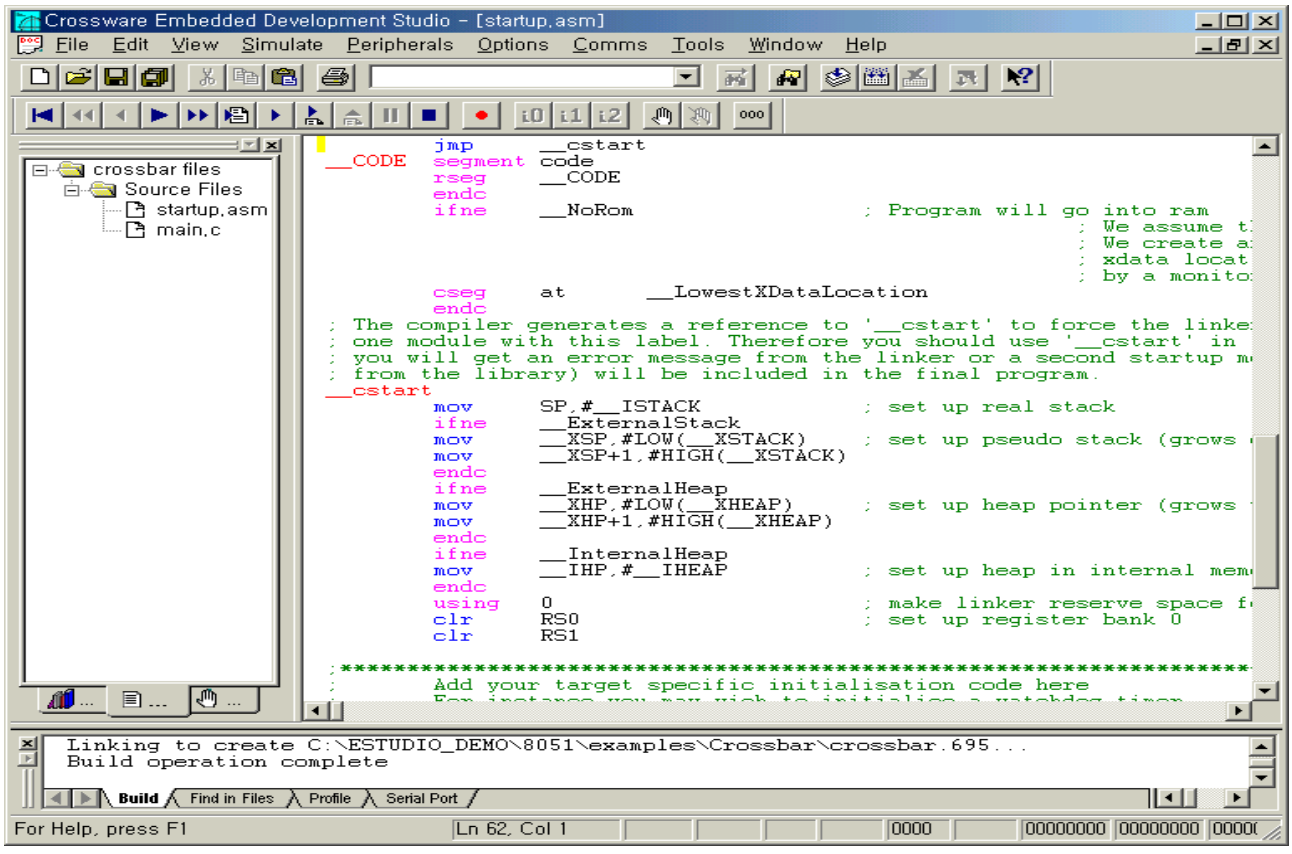


그림 6: 첫번째 스텝실행 아이콘을 클릭한 후의 시뮬레이터 디버그 윈도우

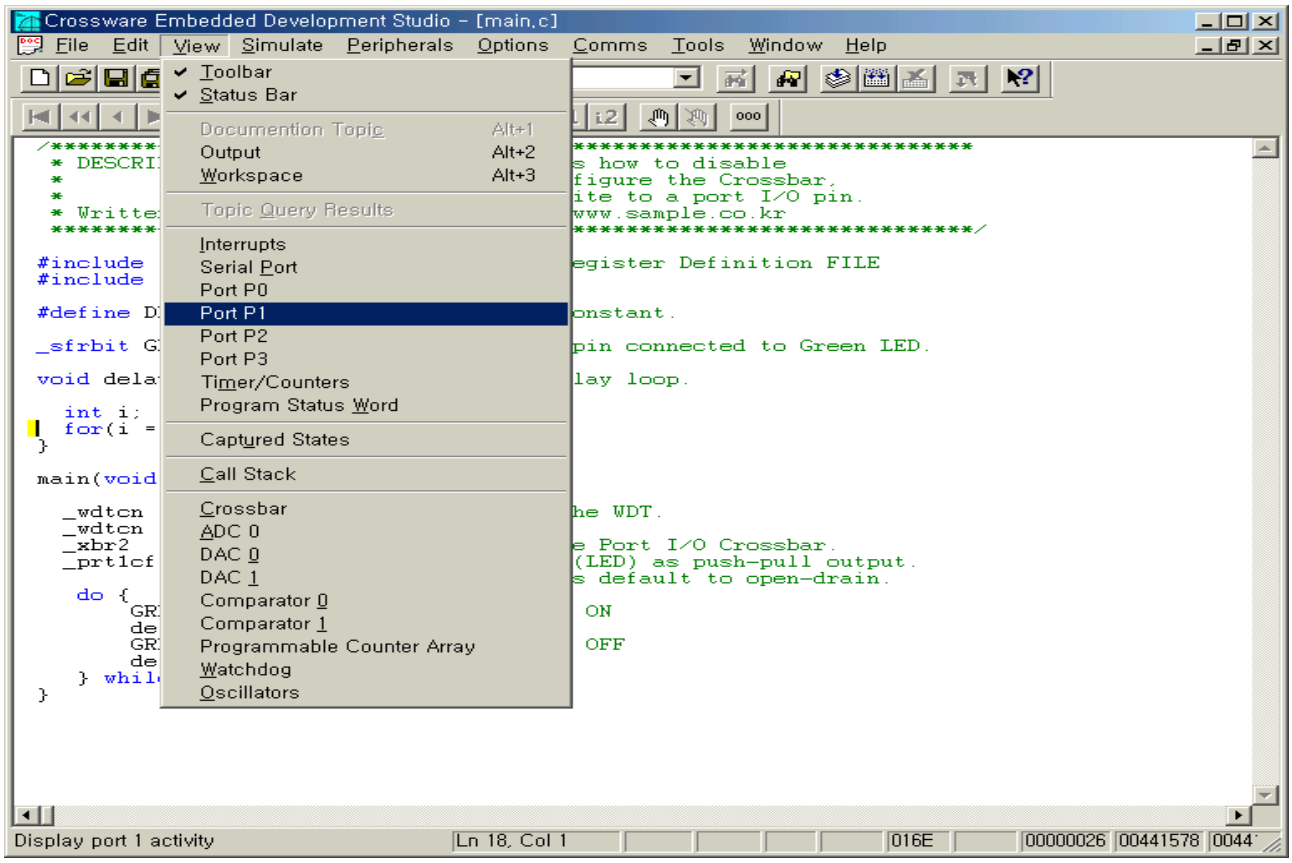


그림 7: Port1 의 그래픽 표시

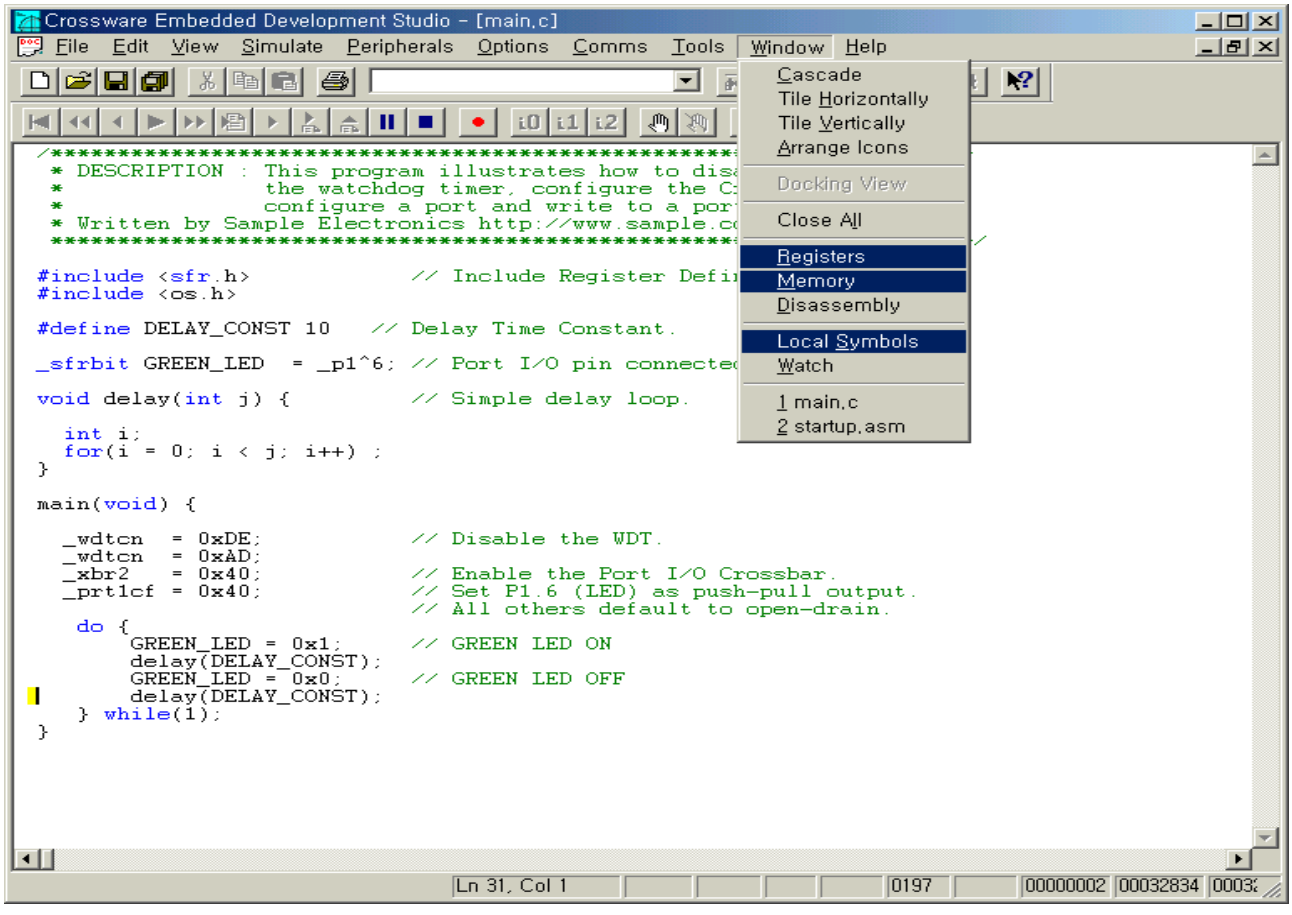


그림 8: 트레이싱 모드 실행중 Register, Memory, Local Symbol 윈도우 오픈

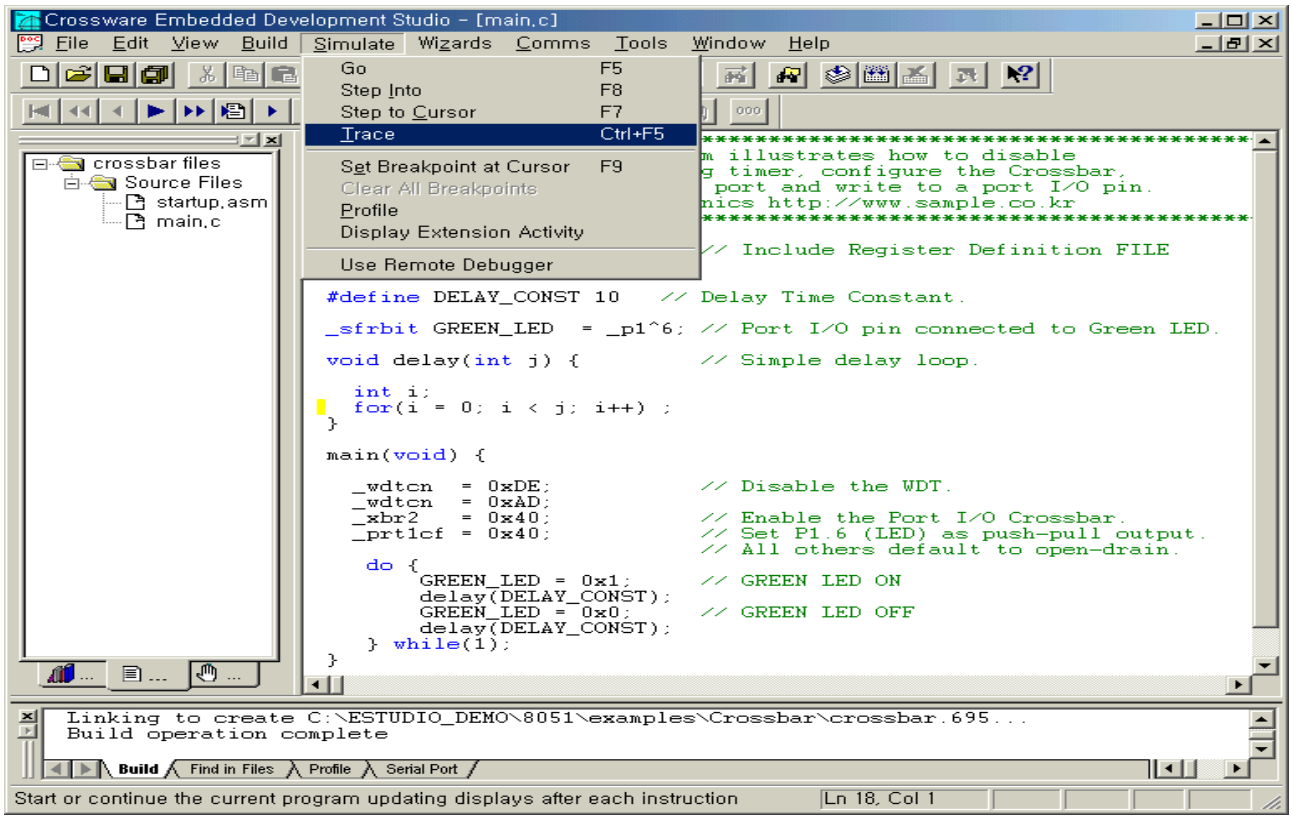


그림 9: 예제 프로그램의 트레이싱 모드 실행

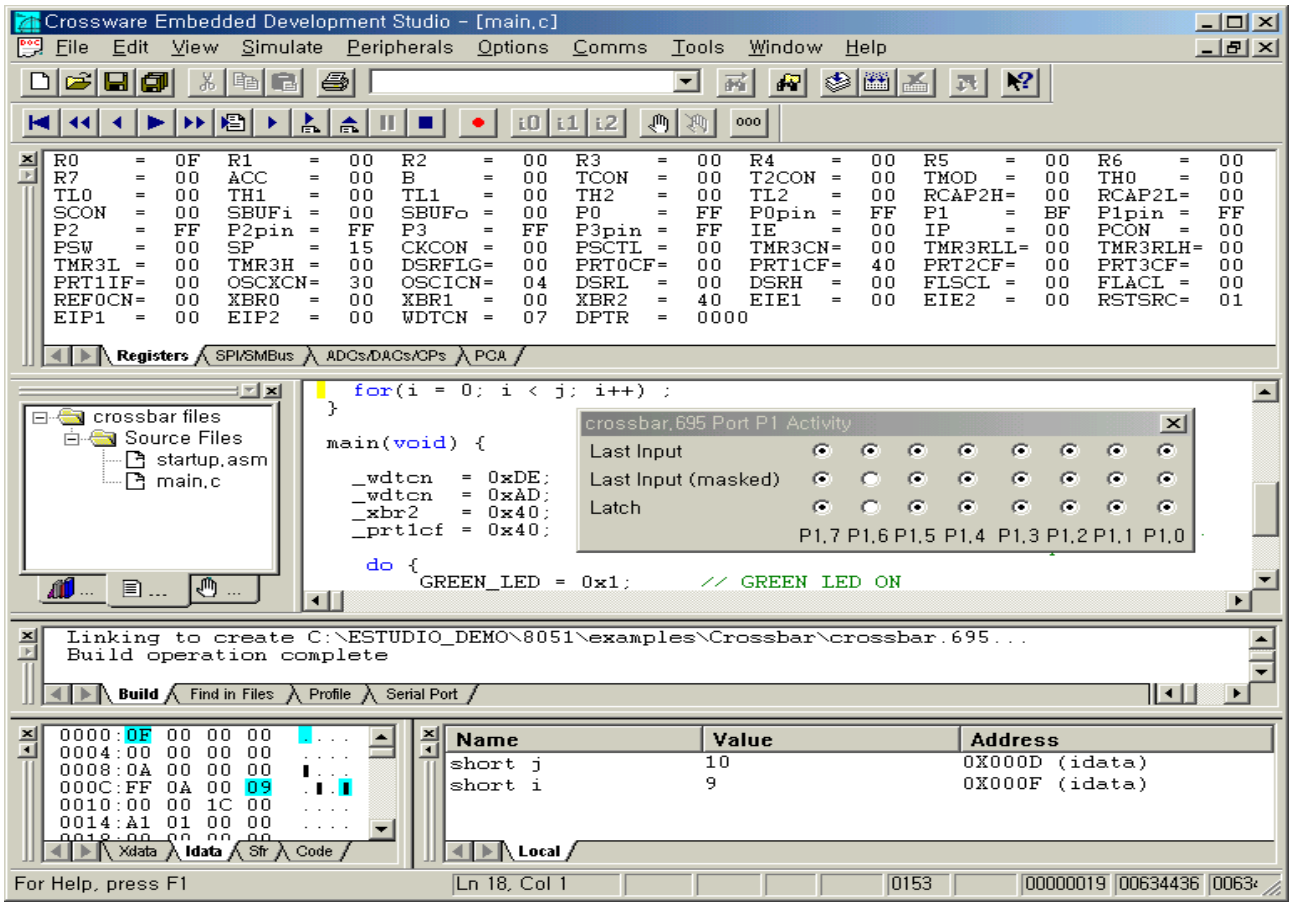


그림 10: 트레이싱 모드 실행화면

스텝실행 아이콘을 클릭하면 그림 6 과 같은 어셈블리 소스 파일이 나타나며 첫번째 명령어에서 노란색 커서가 정지해 있습니다. 이 파일은 startup.asm 파일이며 main 함수가 실행되기 전에 8051 을 초기 설정하는 명령으로 구성되어 있습니다. 예제프로그램은 Port 1 의 P1.6 에 연결된 LED 를 블링크하기 위한 프로그램입니다. Port1 의 각 비트의 동작 상태를 관찰하려면 그림 7 과 같이 윈도우 메뉴바의 View -> Port1 을 클릭하면 됩니다. 그림 8 은 이 예제 프로그램에서 사용된 변수 값의 변화 상태를 감시하고, SFR 레지스터, 내부 메모리 값의 변화를 관찰하기 위하여 그림 8 과 같이 Window-> Registers, Memory, Local Symbols 를 선택합니다. 연속적인 동작상태를 확인하기 위하여 그림 9 와 같이 “Trace” 를 클릭합니다. 그림 10 은 로컬 변수의 변화상태, P1 의 상태, 내부 메모리와 SFR 의 변화상태가 나타나 있습니다. 시뮬레이션을 종료하려면 “Close” 를 클릭합니다.

예제 1 : 새 프로젝트 구성 / Creating a New Projects

- CYNAL 프로세서 C8051F020 을 이용하여 모든 Port 에 부착된 LED 를 점등하는 프로그램 -

어셈블리 언어에 의하여 제어프로그램을 개발할 때 대부분 하나의 어셈블리 소스 파일에 모든 정보를 포함하여 소스파일 하나만 관리하는 방법을 주로 사용합니다. 그러나 C 프로그램으로 개발할 때에는 8051 마이크로 프로세서를 구동하기 위하여 여러 개의 파일이 하나에 작업에 관련되므로 이를 모아서 관리하게 되며 이런 구성 단위를 프로젝트라고 합니다. Embedded Development Studio 는 프로그래머가 자신의 개발작업을 프로젝트(체계적관리) 구성가능 하도록 되어있습니다. Crossware 컴파일러를 인스톨한 초기에는 프로젝트 폴더가 없으므로 최초 1 번은 수작업에 의한 프로젝트를 만들어야 합니다. 그리고 나서 CD 에 있는 예제를 projects 폴더 아래에 복사해 놓고 실행할 수 있습니다.

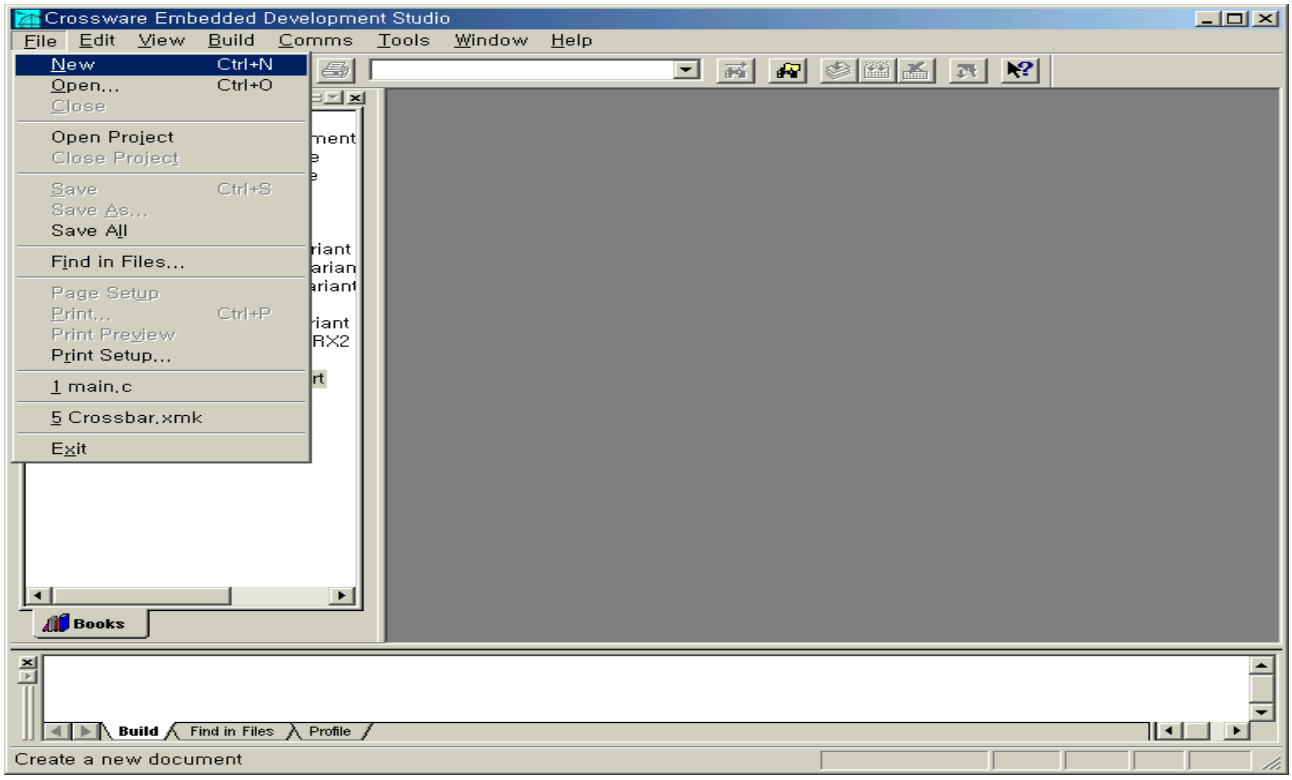


그림 11: 새 프로젝트의 생성

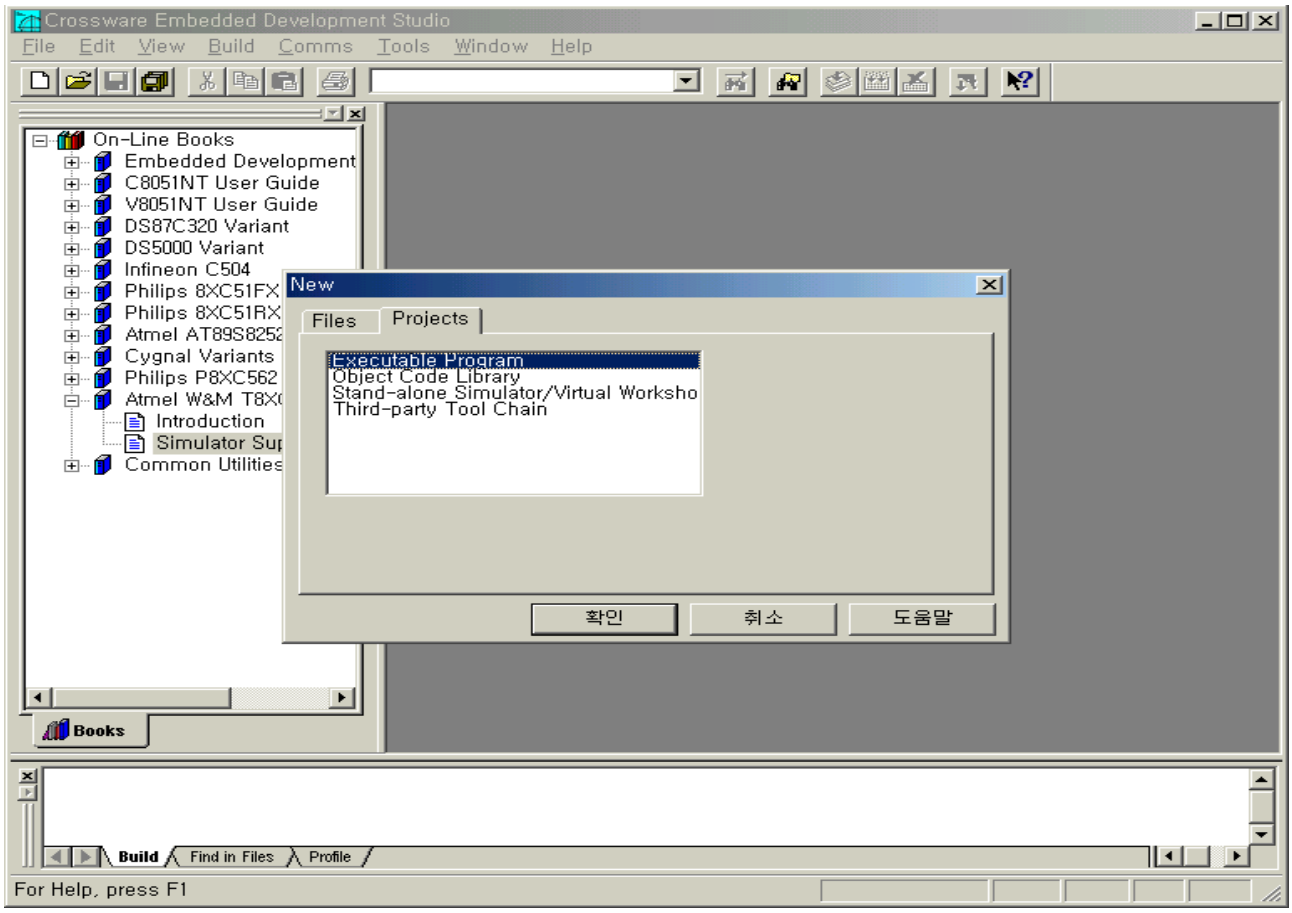


그림 12: 프로젝트 종류 선택

새로운 프로젝트를 설정하는 방법은 New Project Wizard 입니다. New Project Wizard 는 몇 가지의 기본 정보를 선택하는 대화상자이며 간단한 선택을 하면 자동적으로 프로젝트가 생성됩니다. 프로젝트 구성 시 입력한 정보는 나중에 변경이 가능합니다. 그림 11 과 같이 Window 메인메뉴의 File -> New 를 클릭합니다. 첫번째 프로젝트 메뉴가 그림 12 와 같이 표시됩니다. “Project”탭에서 “Executable Program”을 선택합니다. “확인”키를 클릭하면 그림 13 과 같은 프로세서의 종류를 선택합니다. (Intel 의 표준 8051 과 이와 동일한 구조를 갖는 프로세서를 사용할 경우 Generic8051 을 선택합니다. ATMEL 사의 AT89C51 과 AT89C2051 은 Generic8051 을 선택하여야 합니다.) C8051F020 프로세서를 선택합니다. 그림 14 에서 프로젝트 이름을 입력합니다. 그림 15 는 메모리 모델을 지정합니다. Tiny 는 프로그램 메모리가 2K Bytes 이하인 경우입니다. Small 은 프로그램 메모리가 64K Bytes 로 제한은 없으나 스택 영역과 변수 영역의 메모리로 프로세서 내부의 128Bytes 또는 256Bytes 만 사용하는 것입니다. Large 는 프로그램 메모리가 64K Bytes 까지 가능하며 외부에 SRAM 을 64K Bytes 사용가능합니다. Large 메모리 모델에서 내부 메모리는 스택영역으로만 사용하며 변수에 필요한 메모리는 외부 메모리 영역에 배치합니다. 프로그램 메모리를 일반 EPROM 을 사용할 때 이 용량에 맞는 어드레스를 지정하여야 합니다. 그림 16 은 CROSSWARE 또는 사용자가 새로 구성한 라이브러리를 선택할 때 필요합니다. 표준사용 시 선택 사항은 나타나지 않습니다. 기본 라이브러리 이외에 추가적인 라이브러리가 있을 때 선택합니다. 특정한 OEM 타겟 보드에 맞도록 구성된 라이브러리를 선택합니다. 그림 17 은 컴파일후 생성되는 출력 파일 형태를 선택합니다. 출력 파일종류는 **Intel Hex, MOTOROLA S-Record, OMF51, Extended OMF51, IEEE695, Binary Image** 가 있습니다. 대부분의 유니버셜 프로그래머는 **Intel Hex, S-Record** 포맷을 지원합니다. **OMF51** 은 Intel 사에서 제정한

디버깅 정보가 포함된 파일입니다. CYGNAL 의 IDE 에서 **OMF51** 포맷을 지원합니다. Integer 형식으로 8051 의 내부를 관찰할 수 있습니다. IEEE695 는 8051 프로세서의 In-Circuit Emulator 제조업체로 유명한 **Microtec** 사와 **HP** 사가 공동으로 제안하였으며 **ANSI C 에서 사용하는 모든 데이터 유형의 디버그 정보가 포함된 출력파일 포맷**입니다. 이 포맷은 미국 전기/전지 기술자 협회 (IEEE)에 695 번으로 등록되어 8051 인서킷 디버거 장비 제조업체에서 표준으로 사용하는 파일 포맷입니다. 주로 C 와 같은 하이레벨 언어의 디버깅에 사용합니다. OMF51 은 정수 변수만 관찰이 가능하지만 **IEEE695 는 부동소숫점, 스트링, 유니온, 스트럭쳐등 ANSI - C 에서 정의한 모든 변수의 디버깅이 가능**합니다. Crossware Development Suit 을 이용하여 시뮬레이션 또는 하드웨어 디버깅이 가능하려면 IEEE695 파일 포맷을 선택하여야 합니다. 그림 18 은 프로젝트에서 사용되는 언어를 C 언어로 할것인지 8051 어셈블리 언어로 할것인지를 선택합니다.

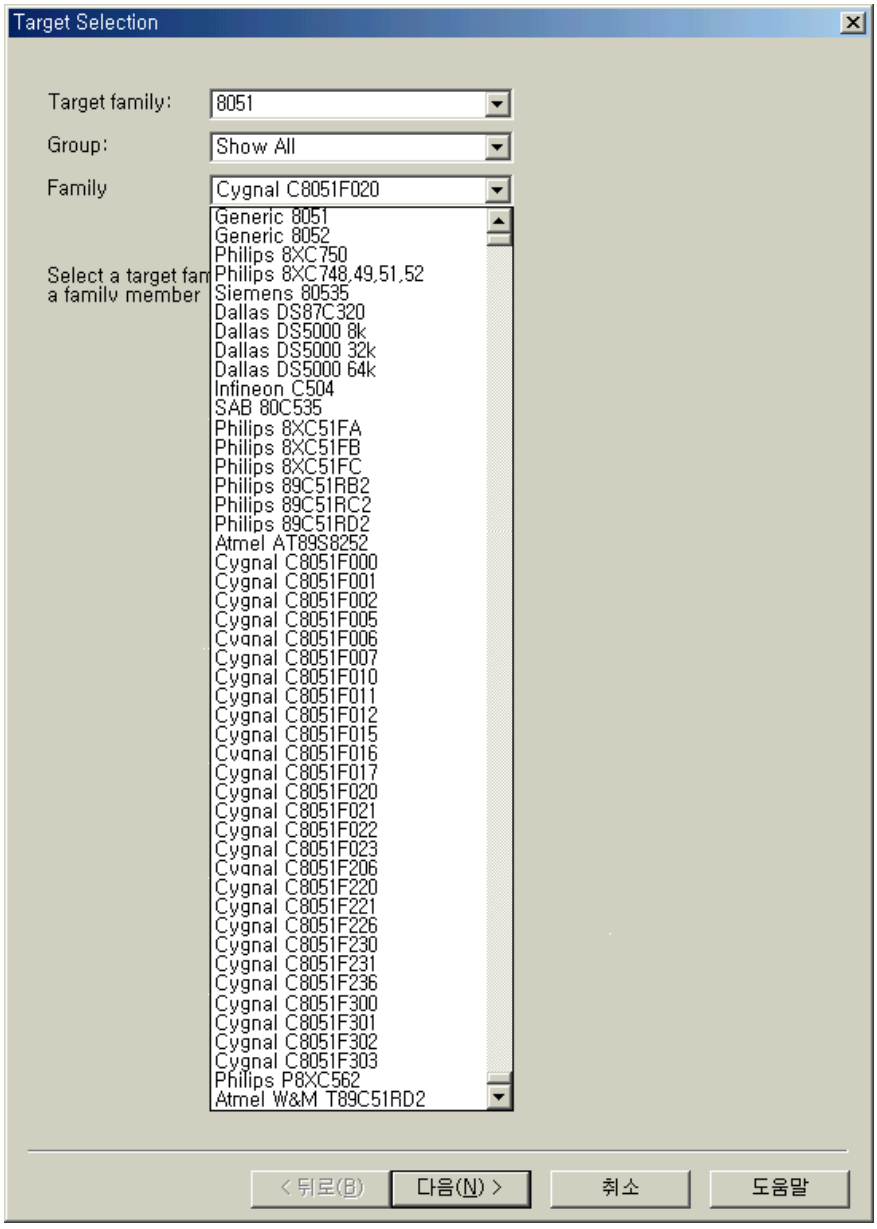


그림 13: 프로세서 종류선택

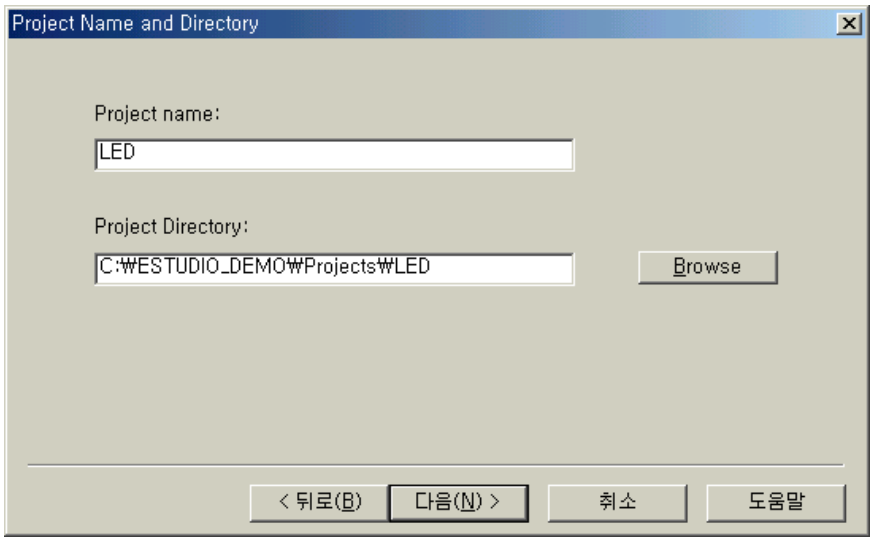


그림 14: 프로젝트 이름 입력

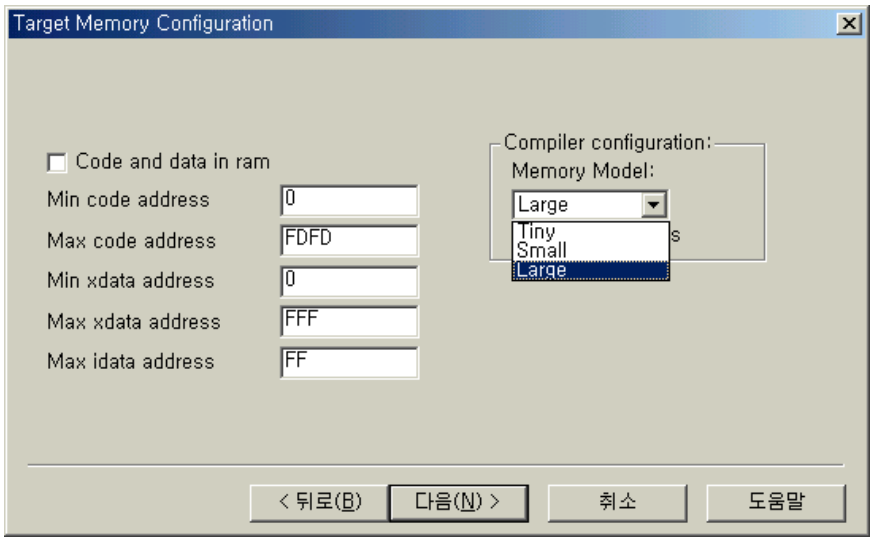


그림 15: 메모리 모델 선택

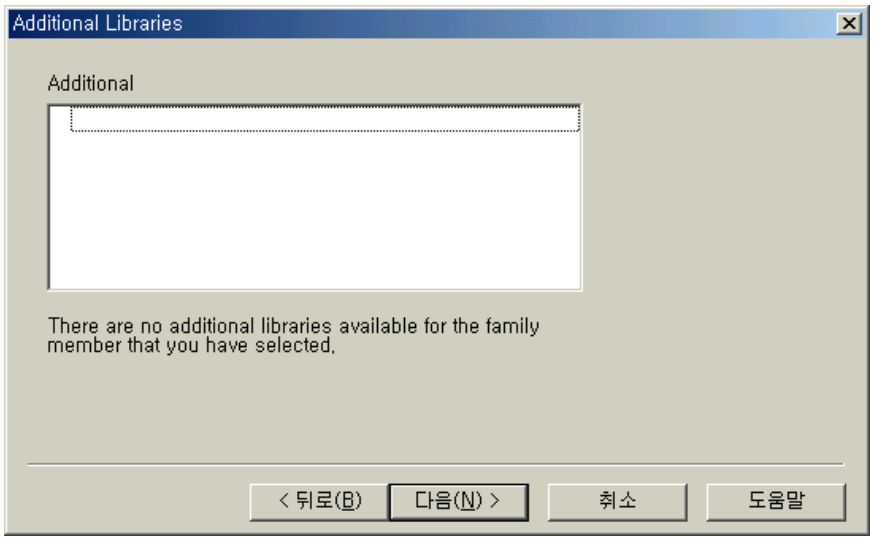


그림 16: 추가 라이브러리 선택

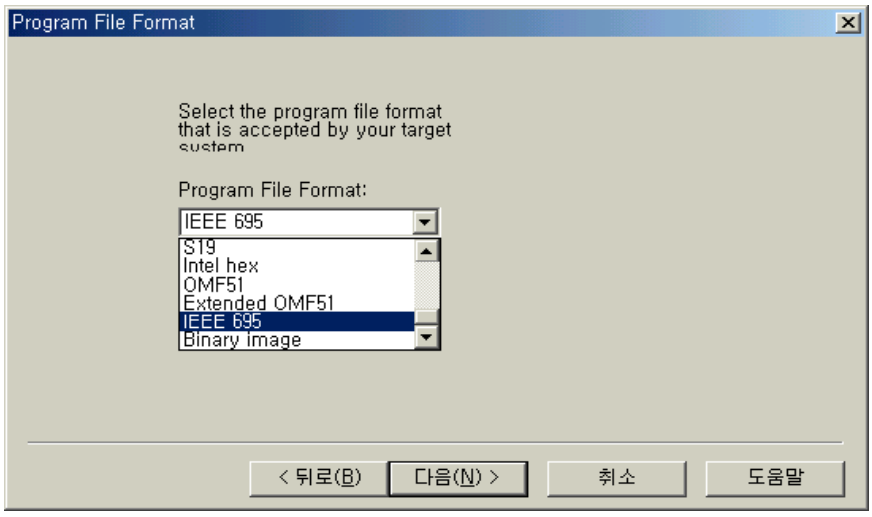


그림 17: 출력 파일 포맷 선택

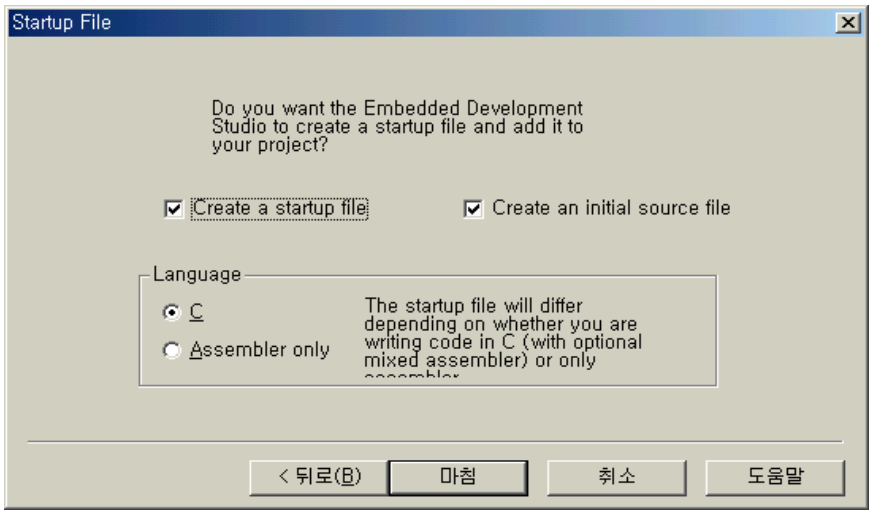


그림 18: 초기 생성 파일 종류선택

Project 구성이 완료되면 startup.asm 과 그림 19 와 같은 main.c 프로그램이 만들어 집니다. 이 상태에서 컴파일이 가능하며 아무런 에러상황도 나타나지 않습니다. 그러나 프로그램을 컴파일하여 헥사 파일을 8051 에 프로그래밍 하여도 8051 은 아무일도 하지 않을 것입니다. CYGNAL 프로세서는 리셋후 즉시 Watchdog 타이머의 사용할 것인지 금지할 것인지를 선택하여야 하며 내부에 내장된 주변장치(Port I/O 포함)가 외부 핀을 사용한다면 CROSSBAR 를 이용하여 핀 할당을 하여야 합니다. Development Suit 은 CYGNAL 프로세서를 사용할 때 프로그램의 초기화를 쉽게하기 위하여 강력한 Wizard 편집기능을 제공합니다. Wizard 기능이란 CYGNAL 프로세서가 내장한 주변장치를 그래픽으로 표시하고 단순히 마우스를 클릭하면 자동적으로 주변장치를 제어하는 초기 설정 코드를 생성하는 기능입니다. 생성된 코드는 xstdsys.c 에 삽입하거나 main.c 프로그램에서 커서가 위치한곳에 만들어 지도록 선택할 수 있습니다. Main.c 파일은 xstdsys.c 를 자동적으로 include 되어 있으므로 따로 지정할 필요는 없습니다. CYGNAL 프로세서를 이용한 하드웨어를 개발할 때 가장 먼저 결정해야할 것은 Watchdog 을 사용할것인지를 결정하여야 합니다. Watchdog 을 사용하지 않을 계획이면 리셋즉시 Watchdog 타이머의 동작을 중지시켜 놓아야 합니다. 표준 8051 프로세서는 Watchdog 기능이 없지만 개선된 8051 프로세서는 대부분 Watchdog 타이머를 내장하고 있습니다. CYGNAL 의 Watchdog 타이머는 다른 8051 과 달리 리셋직후 사용가능 상태로 설정되 있습니다.

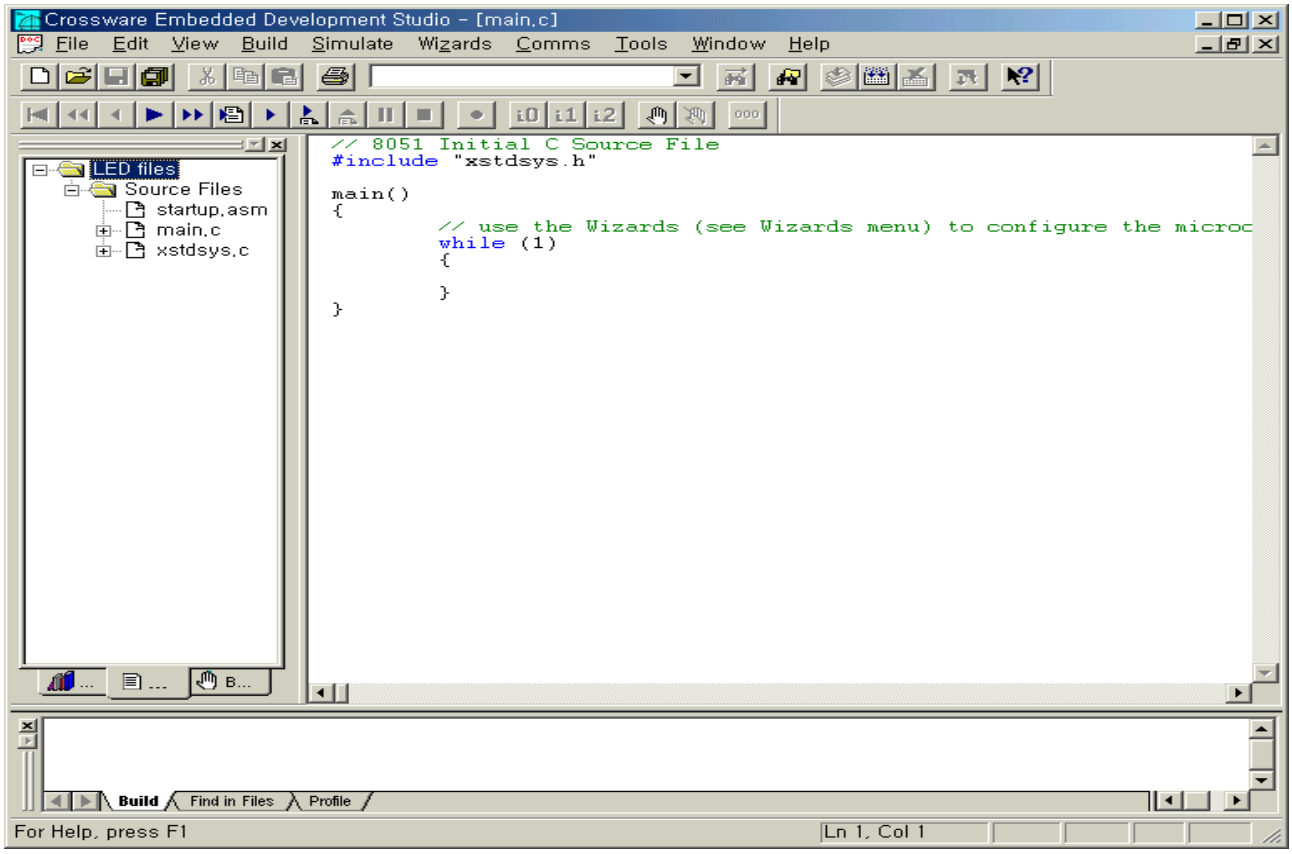


그림 19: 프로젝트 구성후 최초 main.c 프로그램

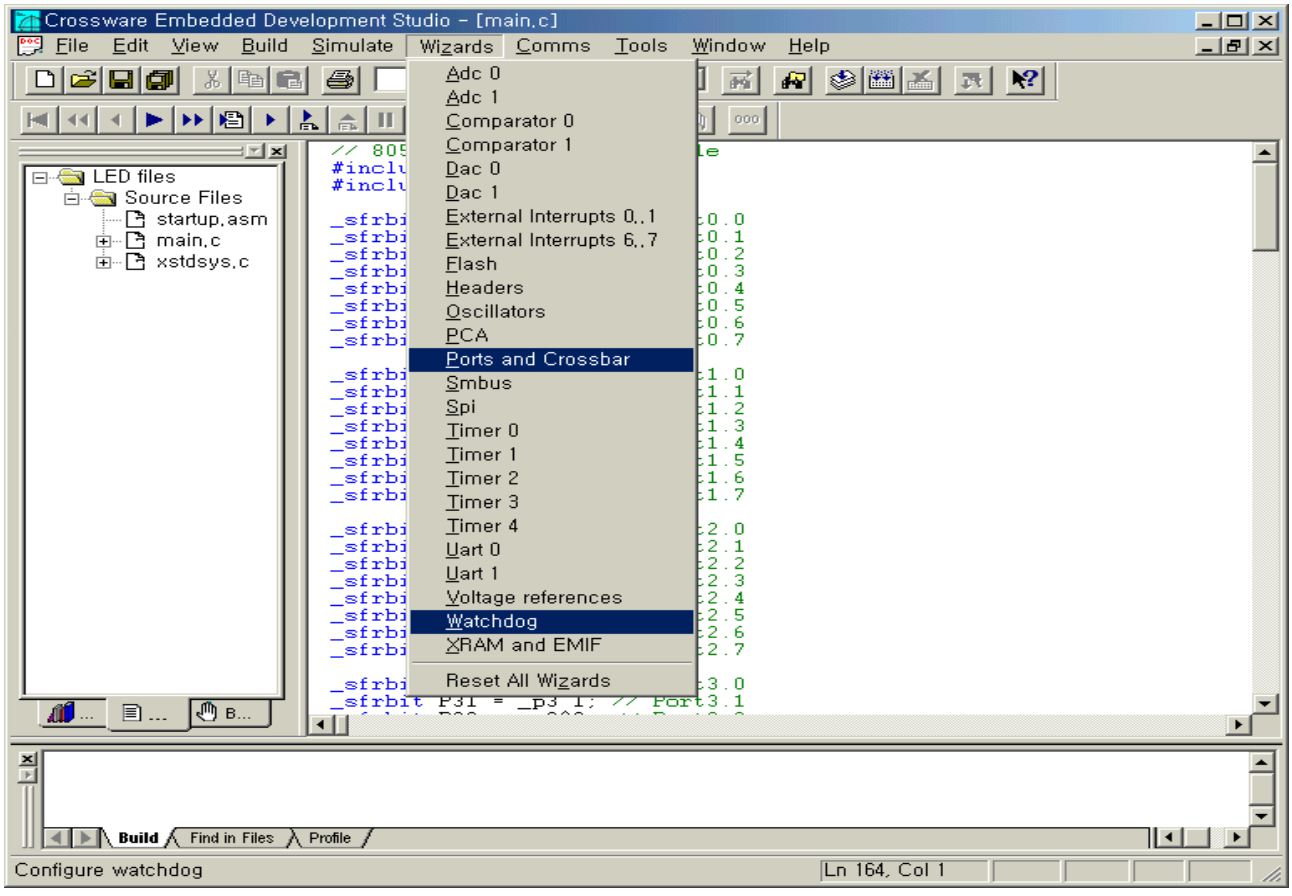


그림 20: Wizard 를 이용한 Crossbar 설정

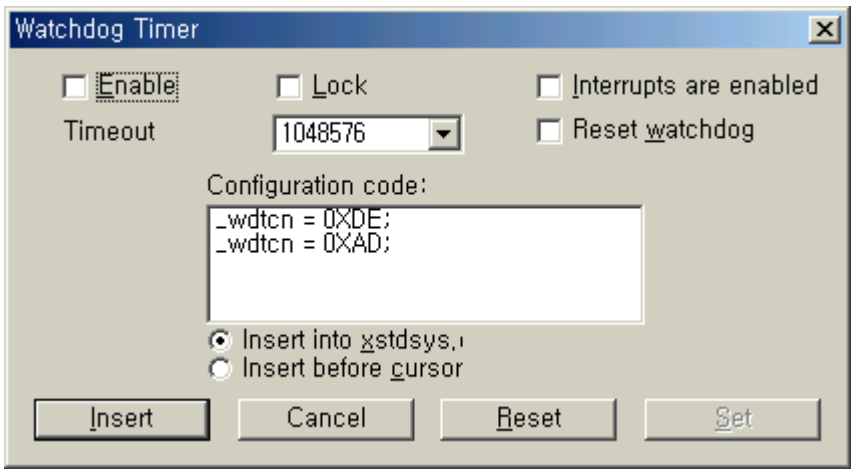


그림 21: Watchdog 설정 윈도우

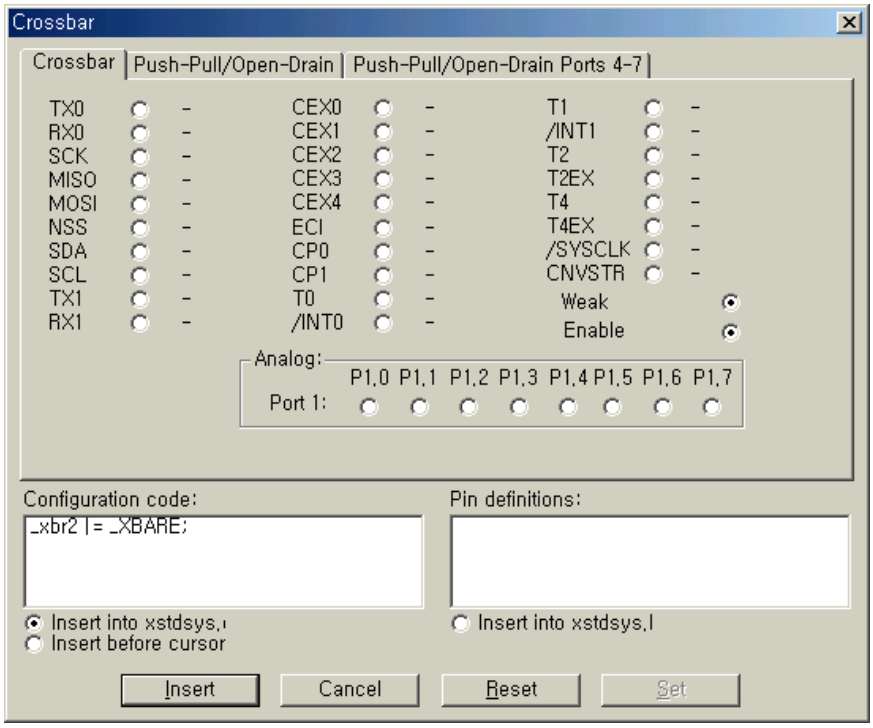


그림 22: 크로스바 설정 메뉴

그림 23: Watchdog 발생 메시지

Watchdog 타이머를 프로그램에서 일정시간 안에 클리어 해주지 않거나 동작금지 상태로 하지 않으면 프로세서는 리셋됩니다. 그림 20 과 같이 윈도우 메뉴바 Wizard-> Watchdog 을 클릭하면 그림 21 과 같은 Watcdog 윈도우가 나타나며 Enable 을 클릭하여 Watchdog 사용금지 상태로 만듭니다. Watchdog 을 사용금지 상태로 프로그램 하지 않으면 Crossware 의 Development Suit 은 시뮬레이션 모드 실행중 그림 23 과 같은 에러 메시지를 표시합니다. 그 다음 그림 20 과 같이 Wizard -> Ports and Crossbar 를 선택합니다. 그림 22 과 같은 윈도우가 표시되며 “Enable”을 체크합니다. Xstdsys.c 파일을 오픈하여 보면 그림 24 와 같

이 Crossbar 사용가능 상태로 만드는 코드가 자동으로 입력된 것을 확인할 수 있습니다.

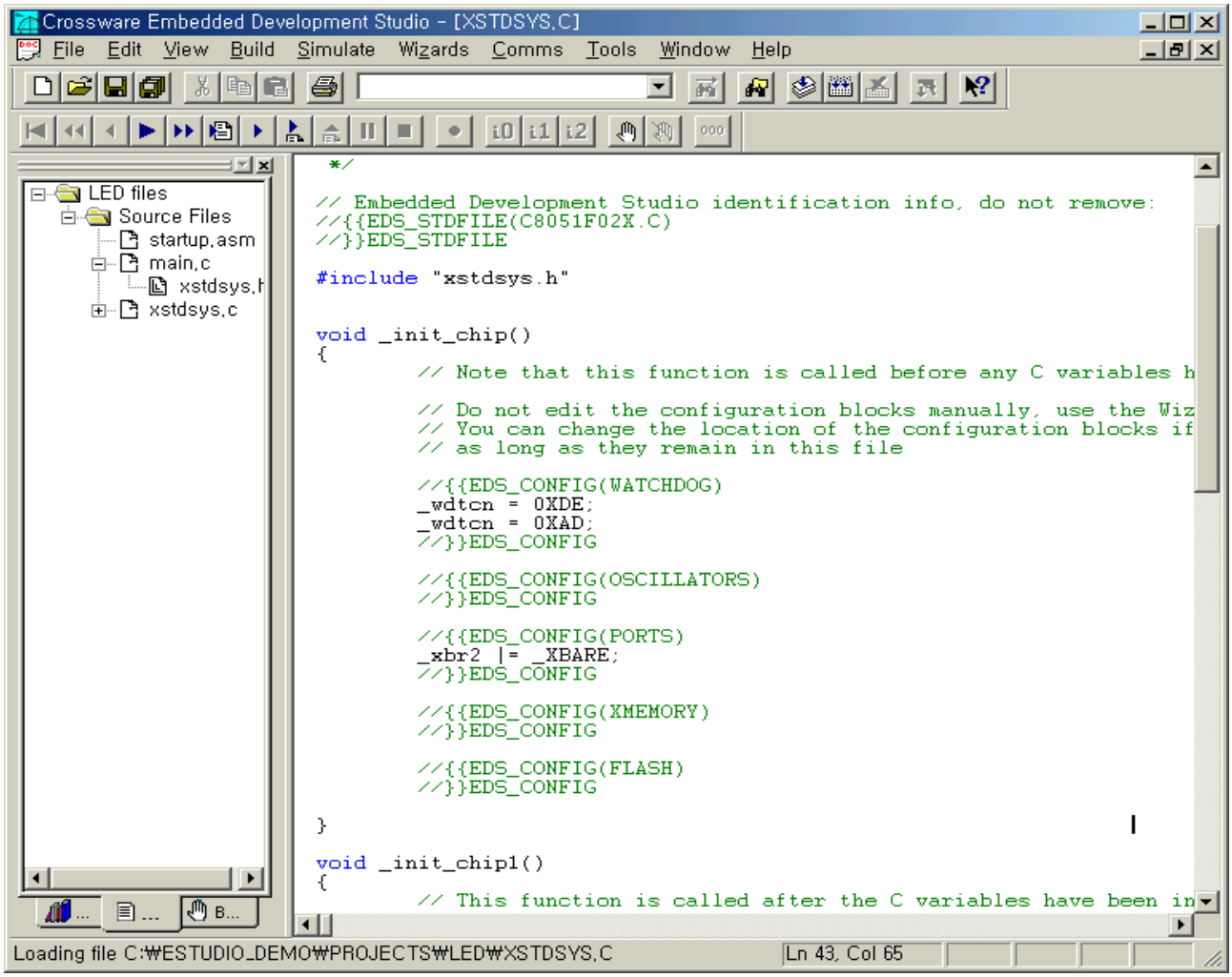


그림 24: Wizard 에 의하여 만들어진 초기화 프로그램

그림 25 은 CYGNAL 프로세서 C8051F020 의 모든 디지털 I/O 포트에 부착된 LED 를 점등하기 위한 예제 프로그램입니다. _sfrbit 를 이용하여 Port I/O 비트 이름을 정의합니다. 코딩이 완료되면 Window 메뉴바의 Build -> Rebuild All 을 클릭하여 컴파일합니다.

```
// 8051 Initial C Source File
#include "xstdsys.h"
#include <sfr.h>

_sfrbit P00 = _p0^0; // Port0.0
_sfrbit P01 = _p0^1; // Port0.1
_sfrbit P02 = _p0^2; // Port0.2
_sfrbit P03 = _p0^3; // Port0.3
_sfrbit P04 = _p0^4; // Port0.4
_sfrbit P05 = _p0^5; // Port0.5
_sfrbit P06 = _p0^6; // Port0.6
_sfrbit P07 = _p0^7; // Port0.7

_sfrbit P10 = _p1^0; // Port1.0
_sfrbit P11 = _p1^1; // Port1.1
_sfrbit P12 = _p1^2; // Port1.2
_sfrbit P13 = _p1^3; // Port1.3
_sfrbit P14 = _p1^4; // Port1.4
_sfrbit P15 = _p1^5; // Port1.5
_sfrbit P16 = _p1^6; // Port1.6
_sfrbit P17 = _p1^7; // Port1.7
```

```
_sfrbit P20 = _p2^0; // Port2.0
_sfrbit P21 = _p2^1; // Port2.1
_sfrbit P22 = _p2^2; // Port2.2
_sfrbit P23 = _p2^3; // Port2.3
_sfrbit P24 = _p2^4; // Port2.4
_sfrbit P25 = _p2^5; // Port2.5
_sfrbit P26 = _p2^6; // Port2.6
_sfrbit P27 = _p2^7; // Port2.7

_sfrbit P30 = _p3^0; // Port3.0
_sfrbit P31 = _p3^1; // Port3.1
_sfrbit P32 = _p3^2; // Port3.2
_sfrbit P33 = _p3^3; // Port3.3
_sfrbit P34 = _p3^4; // Port3.4
_sfrbit P35 = _p3^5; // Port3.5
_sfrbit P36 = _p3^6; // Port3.6
_sfrbit P37 = _p3^7; // Port3.7


main()
{
        // use the Wizards (see Wizards menu) to configure the microcontroller



        while (1)
        {
                  //             01234567
          P00 = 0;   // P0.0 LED On  *
          P01 = 0;   // P0.1 LED On  **
          P02 = 0;   // P0.2 LED On  ***
          P03 = 0;   // P0.3 LED On  ****
          P04 = 0;   // P0.4 LED On  *****
          P05 = 0;   // P0.5 LED On  ******
          P06 = 0;   // P0.6 LED On  *******
          P07 = 0;   // P0.7 LED On  ********

          P07 = 1;   // P0.7 LED Off *******
          P06 = 1;   // P0.6 LED Off ******
          P05 = 1;   // P0.5 LED Off *****
          P04 = 1;   // P0.4 LED Off ****
          P03 = 1;   // P0.3 LED Off ***
          P02 = 1;   // P0.2 LED Off **
          P01 = 1;   // P0.1 LED Off *
          P00 = 1;   // P0.0 LED Off

          P10 = 0;   // P1.0 LED On  *
          P11 = 0;   // P1.1 LED On  **
          P12 = 0;   // P1.2 LED On  ***
          P13 = 0;   // P1.3 LED On  ****
          P14 = 0;   // P1.4 LED On  *****
          P15 = 0;   // P1.5 LED On  ******
          P16 = 0;   // P1.6 LED On  *******
          P17 = 0;   // P1.7 LED On  ********

          P17 = 1;   // P1.7 LED Off *******
          P16 = 1;   // P1.6 LED Off ******
          P15 = 1;   // P1.5 LED Off *****
          P14 = 1;   // P1.4 LED Off ****
          P13 = 1;   // P1.3 LED Off ***
          P12 = 1;   // P1.2 LED Off **
          P11 = 1;   // P1.1 LED Off *
          P10 = 1;   // P1.0 LED Off

          P20 = 0;   // P2.0 LED On  *
          P21 = 0;   // P2.1 LED On  **
          P22 = 0;   // P2.2 LED On  ***
          P23 = 0;   // P2.3 LED On  ****
          P24 = 0;   // P2.4 LED On  *****
          P25 = 0;   // P2.5 LED On  ******
          P26 = 0;   // P2.6 LED On  *******
          P27 = 0;   // P2.7 LED On  ********

          P27 = 1;   // P2.7 LED Off *******
          P26 = 1;   // P2.6 LED Off ******
          P25 = 1;   // P2.5 LED Off *****
          P24 = 1;   // P2.4 LED Off ****
```

```
            P23 = 1;   // P2.3 LED Off ***
            P22 = 1;   // P2.2 LED Off **
            P21 = 1;   // P2.1 LED Off *
            P20 = 1;   // P2.0 LED Off

            P30 = 0;   // P3.0 LED On  *
            P31 = 0;   // P3.1 LED On  **
            P32 = 0;   // P3.2 LED On  ***
            P33 = 0;   // P3.3 LED On  ****
            P34 = 0;   // P3.4 LED On  *****
            P35 = 0;   // P3.5 LED On  ******
            P36 = 0;   // P3.6 LED On  *******
            P37 = 0;   // P3.7 LED On  ********

            P37 = 1;   // P3.7 LED Off *******
            P36 = 1;   // P3.6 LED Off ******
            P35 = 1;   // P3.5 LED Off *****
            P34 = 1;   // P3.4 LED Off ****
            P33 = 1;   // P3.3 LED Off ***
            P32 = 1;   // P3.2 LED Off **
            P31 = 1;   // P3.1 LED Off *
            P30 = 1;   // P3.0 LED Off

           _p4 = 0xfe; // P4.0 LED On  *
           _p4 = 0xfc; // P4.1 LED On  **
           _p4 = 0xf8; // P4.2 LED On  ***
           _p4 = 0xf0; // P4.3 LED On  ****
           _p4 = 0xe0; // P4.4 LED On  *****
           _p4 = 0xc0; // P4.5 LED On  ******
           _p4 = 0x80; // P4.6 LED On  *******
           _p4 = 0x00; // P4.7 LED On  ********

           _p4 = 0x01; // P4.7 LED Off *******
           _p4 = 0x03; // P4.6 LED Off ******
           _p4 = 0x07; // P4.5 LED Off *****
           _p4 = 0x0f; // P4.4 LED Off ****
           _p4 = 0x1f; // P4.3 LED Off ***
           _p4 = 0x3f; // P4.2 LED Off **
           _p4 = 0x7f; // P4.1 LED Off *
           _p4 = 0xff; // P4.0 LED Off

           _p5 = 0xfe; // P5.0 LED On  *
           _p5 = 0xfc; // P5.1 LED On  **
           _p5 = 0xf8; // P5.2 LED On  ***
           _p5 = 0xf0; // P5.3 LED On  ****
           _p5 = 0xe0; // P5.4 LED On  *****
           _p5 = 0xc0; // P5.5 LED On  ******
           _p5 = 0x80; // P5.6 LED On  *******
           _p5 = 0x00; // P5.7 LED On  ********

           _p5 = 0x01; // P5.7 LED Off *******
           _p5 = 0x03; // P5.6 LED Off ******
           _p5 = 0x07; // P5.5 LED Off *****
           _p5 = 0x0f; // P5.4 LED Off ****
           _p5 = 0x1f; // P5.3 LED Off ***
           _p5 = 0x3f; // P5.2 LED Off **
           _p5 = 0x7f; // P5.1 LED Off *
           _p5 = 0xff; // P5.0 LED Off

           _p6 = 0xfe; // P6.0 LED On  *
           _p6 = 0xfc; // P6.1 LED On  **
           _p6 = 0xf8; // P6.2 LED On  ***
           _p6 = 0xf0; // P6.3 LED On  ****
           _p6 = 0xe0; // P6.4 LED On  *****
           _p6 = 0xc0; // P6.5 LED On  ******
           _p6 = 0x80; // P6.6 LED On  *******
           _p6 = 0x00; // P6.7 LED On  ********

           _p6 = 0x01; // P6.7 LED Off *******
           _p6 = 0x03; // P6.6 LED Off ******
           _p6 = 0x07; // P6.5 LED Off *****
           _p6 = 0x0f; // P6.4 LED Off ****
           _p6 = 0x1f; // P6.3 LED Off ***
           _p6 = 0x3f; // P6.2 LED Off **
           _p6 = 0x7f; // P6.1 LED Off *
           _p6 = 0xff; // P6.0 LED Off

           _p7 = 0xfe; // P7.0 LED On  *
```

```
            _p7 = 0xfc; // P7.1 LED On  **
            _p7 = 0xf8; // P7.2 LED On  ***
            _p7 = 0xf0; // P7.3 LED On  ****
            _p7 = 0xe0; // P7.4 LED On  *****
            _p7 = 0xc0; // P7.5 LED On  ******
            _p7 = 0x80; // P7.6 LED On  *******
            _p7 = 0x00; // P7.7 LED On  ********

            _p7 = 0x01; // P7.7 LED Off *******
            _p7 = 0x03; // P7.6 LED Off ******
            _p7 = 0x07; // P7.5 LED Off *****
            _p7 = 0x0f; // P7.4 LED Off ****
            _p7 = 0x1f; // P7.3 LED Off ***
            _p7 = 0x3f; // P7.2 LED Off **
            _p7 = 0x7f; // P7.1 LED Off *
            _p7 = 0xff; // P7.0 LED Off


        }
}
```

그림 25: 테스트 프로그램 소스

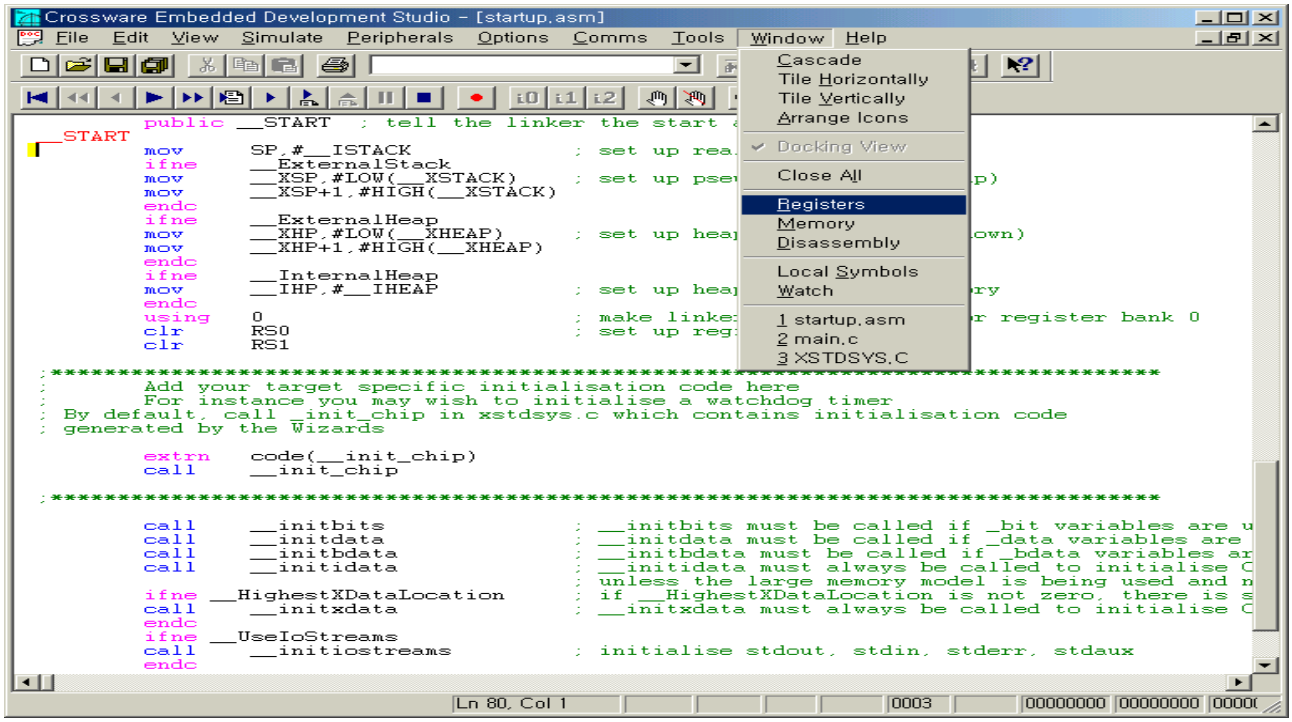


그림 26: 시뮬레이션 모드에서 디버그관련 메뉴

작성한 프로그램의 동작상태를 시뮬레이션 해보기 위하여 Trace 키  를 클릭합니다. 그림 27 과 같은 화면으로 되며 1 라인씩 스텝 실행됩니다. Port 의 상태를 그래픽으로 표시하려면 Window 메뉴바의 View 에서 P0, P1, P2, P3, P4, P5, P6 를 모두 선택하면 그림 28 과 같이 됩니다. 브레이크 포인트 설정은 원하는 위치에 커서를 놓고  를 클릭하면 됩니다. 브레이크 설정은 컴파일 완료 후 설정가능하며 시뮬레이션 /디버깅 모드로 들어간 후에도 설정하는 것이 가능합니다. 이 프로그램은 SE-8051FUP020 보드의 테스트를 하기 위하여 작성된 것입니다. SE-8051FUP 가 EC-2 에 의하여 PC 로 연결되어 있으면 SE-8051FUP 에서 LED 의 점등상태를 볼 수 있습니다. 그림 29 와 같이 Simmulation 모드를 그림 30 과 같은 Debug 모드로 전환하고 트레이스 아이콘  를 클릭하면 그림 31 과 같은 확인용 윈도우가 나옵니다. OK 를 클릭하면 실행코드가 C8051F020 프로세서에 다운로드 된 후 SE-8051FUP 의 LED 가 순차 점등합니다.

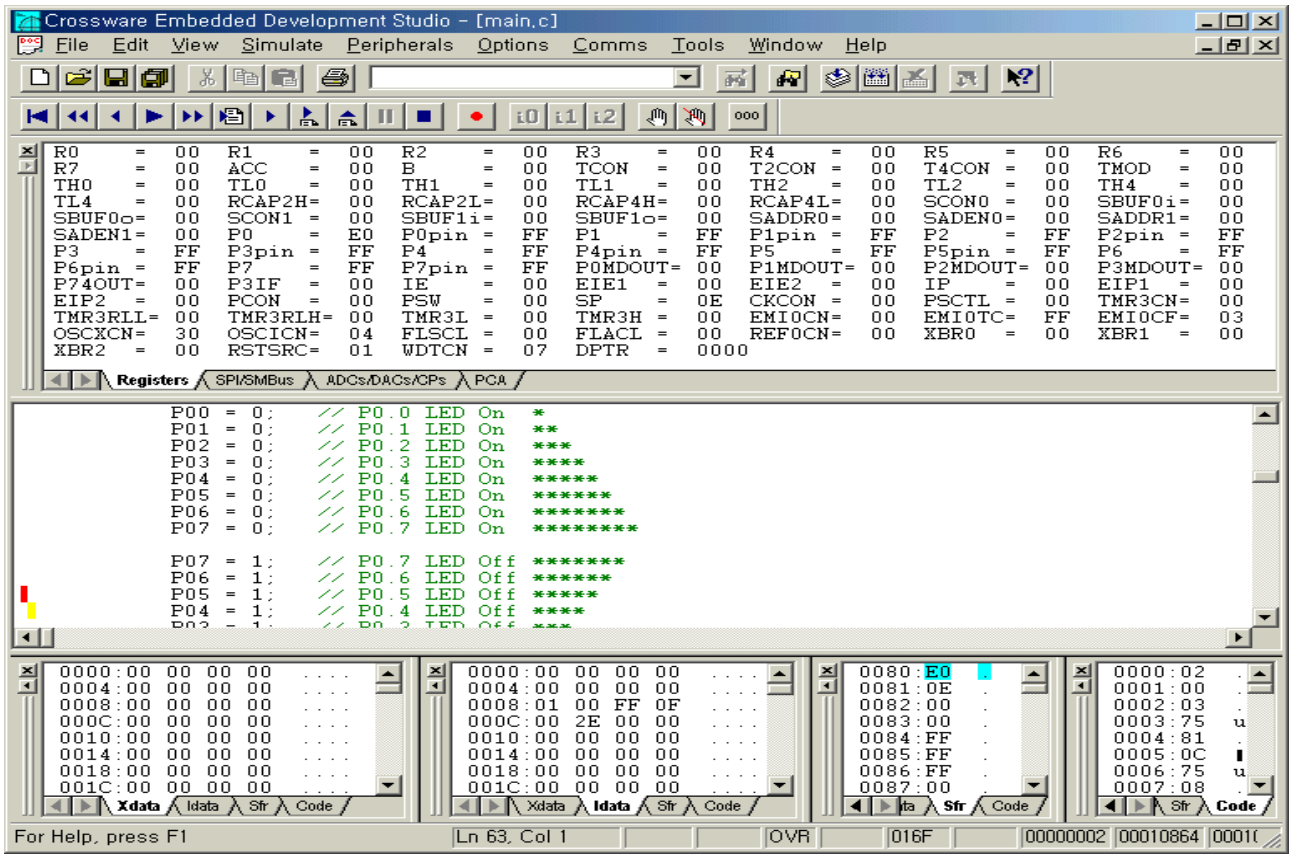


그림 27: 시뮬레이션 모드 실행

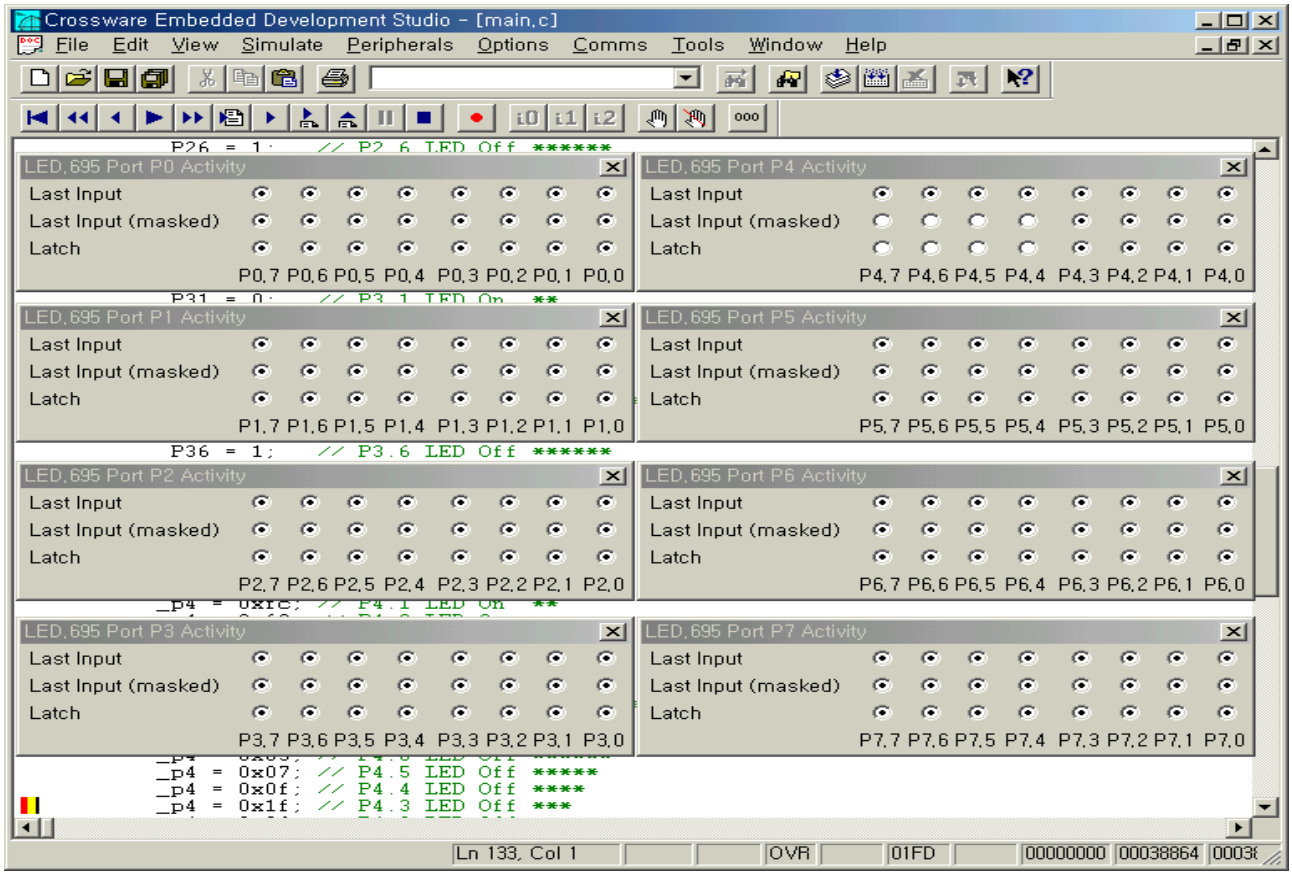


그림 28: Port 의 동작상태를 그래픽 표시

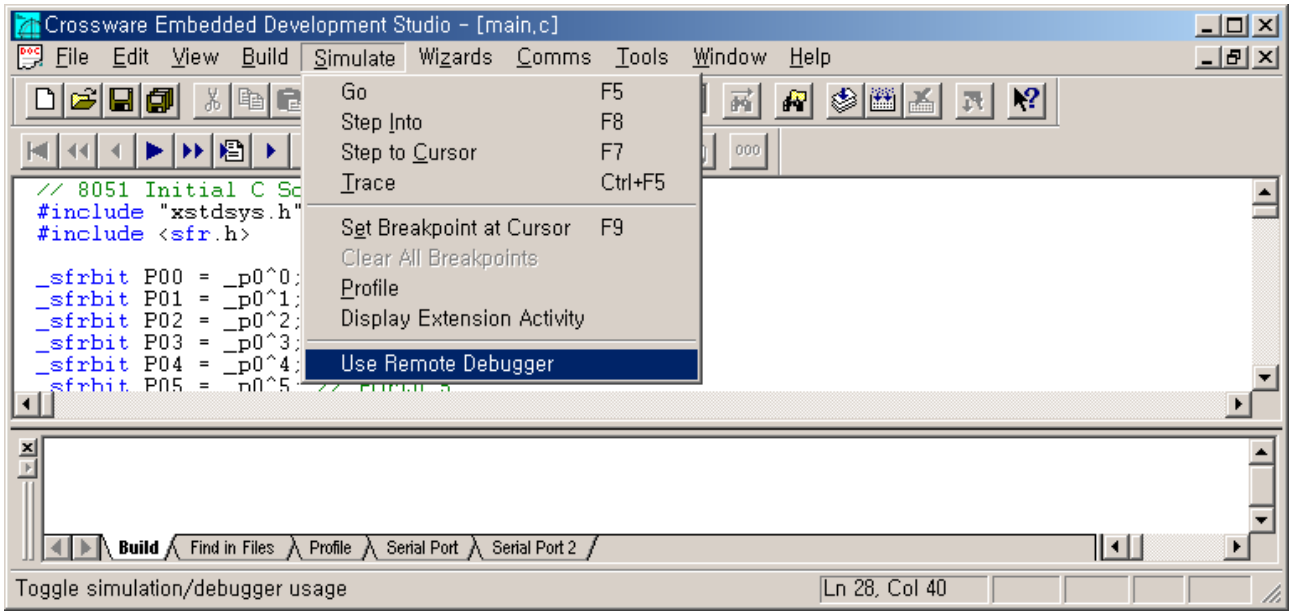


그림 29: Simmulate 모드에서 Debug 모드로 전환

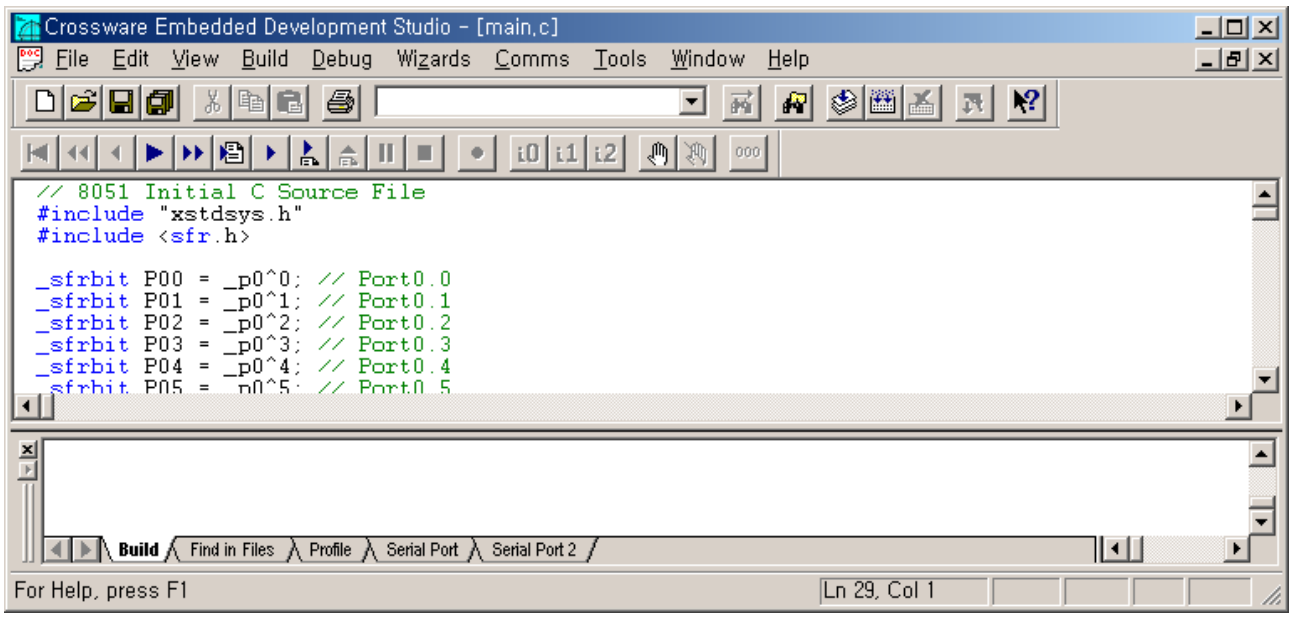


그림 30: Debug 모드

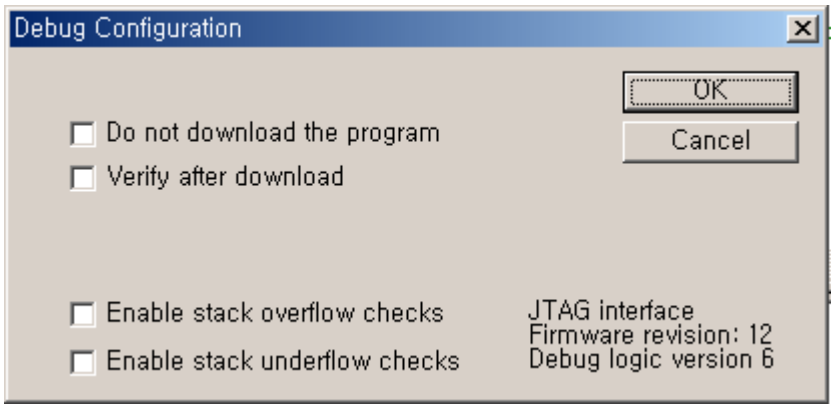


그림 31: 다운로드 확인용 윈도우

예제 2:부동소숫점, 스트링, Structure,Union 변수의 디버깅

Crossware Development Suit 은 IEEE695 포맷 파일을 출력합니다. IEEE695 는 C 에서 사용하는 모든 변수의 디버깅이 가능합니다. 그림 32 는 변수의 형이 비트, 정수, 스트링, 부동소숫점, Structure, Union 의 디버깅 과정을 보여주는 예제입니다. 프로그램을 컴파일한 후 디버깅 모드로 들어가면 그림 33 과 같이 됩니다. 윈도우 메뉴바에서 Windows -> Local Symbols 를 선택하면 main 함수에서 사용하는 모든 로컬 변수가 나타납니다. 숫자는 십진수로 표시하거나 마우스를 Value 위치에서 오른쪽 마우스 버튼을 클릭하면 16 진수 로 표시할 수 있습니다. 이 예제 프로그램은 16 비트 정수 변수를 상위 바이트 하위바이트로 나누어 P1 과 P2 에 출력하는 방법을 보여주며 사용가능한 연산자의 응용 예와 변수의 선언방법을 설명합니다. Crossware Demo 에서 동작할 수 있으며 트레이스 모드 사용으로 변수의 변화 상태를 단계적으로 확인할 수 있습니다.

```
/************************************************************
 Description:
 이 프로그램은 Crossware ANSI C 컴파일러에서
 지원하는 변수 형식과 연산자의 사용 예를 보인것입니다.
 비트변수, 정수변수, 부동소숫점, 스트링 그리고 Structure 와
 Union 변수의 변화 상태를 관찰할 수 있습니다.

 Witten by SAMPLE Electronics co.      http://www.SAMPLE.co.kr
 ************************************************************/

#include <sfr.h>                          // ( 1)
#include <string.h>                        // ( 2)
#include "xstdsys.h"                       // ( 3)
                                           // ( 4)
#define PI 3.141592                        // ( 5) 상수 지정
                                           // ( 6)
_sfrbit P00 = _p0^0;                       // ( 7) Port 0 Bit 0
                                           // ( 8)
typedef union {                            // ( 9) Union 형식 선언
  unsigned int wordData;                   // (10) 16 비트 wordData 는
  struct {                                 // (11) 2 개의 8 비트 byte_low 와
    unsigned char byte_low;                // (12) byte_high 가 공통 메모리를 사용
    unsigned char byte_high;               // (13)
  } byteData;                              // (14)
} WORDAREA;                                // (15)
                                           // (16)
typedef struct {                           // (17) Struct 형식 선언
  char name[10];                           // (18) 스트링 변수
  int age;                                 // (19) 정수 변수
  double value;                            // (20) 64 비트 배 정도형 부동 소숫점
} MEMBER;                                  // (21)
                                           // (22)
float area;                                // (23) 전역변수 정의
                                           // (24)
void main() {                              // (25)
                                           // (26)
  _bit yellow;                             // (27) 비트 변수 선언
  _bit blue;                               // (28)
  _bit red;                                // (29)
```

```
  char banana;                          // (30) 8 비트 부호 변수 선언
  char cherry;                          // (31)
  unsigned char apple;                  // (32) 8 비트 무부호 변수선언
  int truck;                            // (33) 16 비트 부호 정수 변수 선언
  unsigned int car;                     // (34) 16 비트 무부호 정수 변수  선언
  long int train;                       // (35) 64 비트 정수 변수
  double radius = 9.87654321;           // (36) 64 비트 배 정도형 부동 소숫점
  WORDAREA adcresult;                   // (37) Union 형식의 변수 정의
  MEMBER sample;                        // (38) Struct 형식의 변수 정의
                                        // (39)
  yellow = 0;                           // (40) 비트 변수
  blue = 1;                             // (41)
  red = yellow | blue;                  // (42) 논리 OR 연산
  red = yellow & blue;                  // (43) 논리 AND 연산
  P00 = red;                            // (44) Port P0.0 을 0 으로 설정
                                        // (45)
  banana = 0xff;                        // (46)
  apple = 0x55;                         // (47)
  cherry = apple ^ banana;              // (48) 배타적(Exclusive) OR 연산
  cherry = ~cherry;                     // (49) 비트 반전 (1's Compliment)
  banana = !cherry;                     // (50)
  apple = !banana;                      // (51)
                                        // (52)
  truck = 0xcccc;                       // (53)
  car = truck << 5;                     // (54) 왼쪽으로 5 비트 쉬프트
  car = truck >> 2;                     // (55) 오른쪽으로 2 비트 쉬프트
                                        // (56)
  train = 0;                            // (57)
  --train;                              // (58) Pre Decrement
  truck++;                              // (59) Post Increment
  cherry--;                             // (60) Post Decrement
  ++apple;                              // (61) Pre Increment
                                        // (62)
  adcresult.wordData = 0x55AA;          // (63) 16 비트 데이터 지정
                                        // (64)
  _p1 = adcresult.byteData.byte_low;    // (65) 16 비트의 하위 바이트를 P1 에 출력
  _p2 = adcresult.byteData.byte_high;   // (66) 16 비트의 상위 바이트를 P2 에 출력
                                        // (67)
  adcresult.byteData.byte_low = 0x34;   // (68) 8 비트 하위 데이터 지정
  adcresult.byteData.byte_high = 0x12;  // (69) 8 비트 상위 데이터 지정
                                        // (70)
  strcpy(sample.name, "Crossware");     // (71) 스트링 복사
  sample.age = 7;                       // (72)
  sample.value = -1.23456789e-123;      // (73)
  area = PI + radius - 5.4321;          // (74) 가산(+), 감산(-)
  area = radius * radius * PI / 3.21;   // (75) 승산(X), 제산(/)
                                        // (76)
  truck = 12;                           // (77)
  car = 12 % 10;                        // (78) 나머지 값의 계산
                                        // (79)
  while (1){ ; }                        // (80) 무한루프
}                                       // (81)
```

그림 32: Structure, Union 변수를 이용하는 예제 프로그램

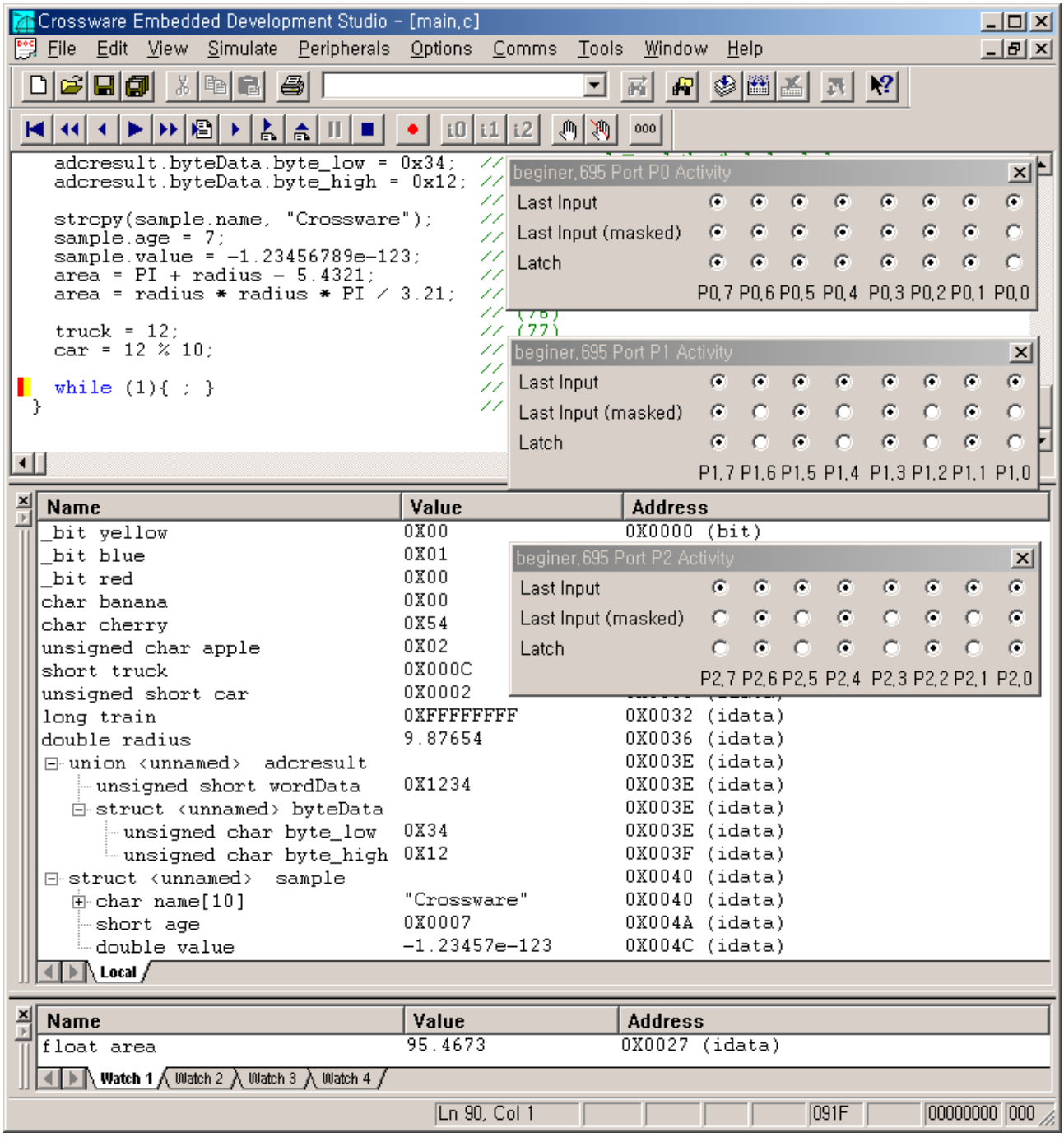


그림 33: Structure, Union 변수, 지역변수(main 함수), 전역변수 의 디버깅

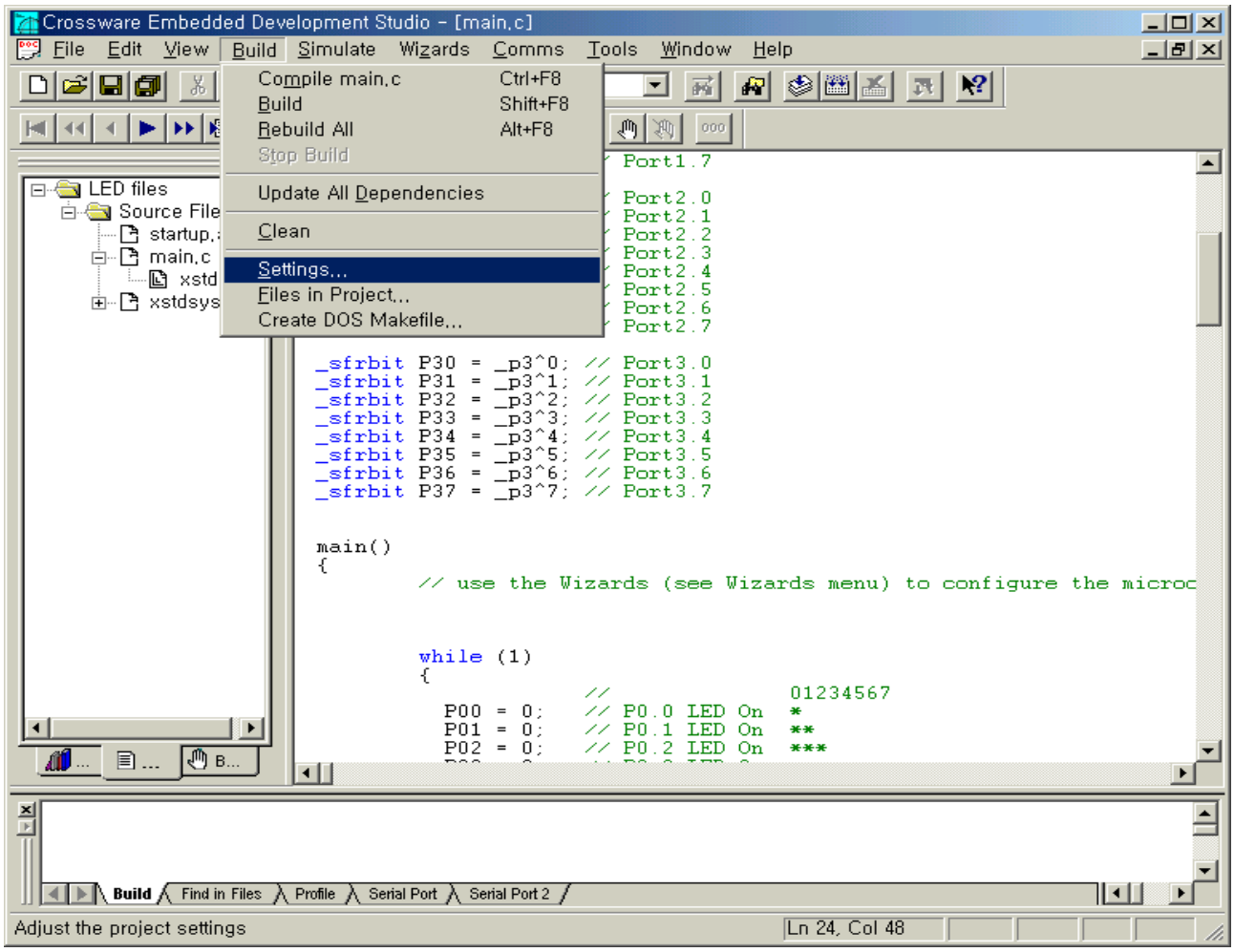


그림 34: 프로젝트 구성 요소 변경

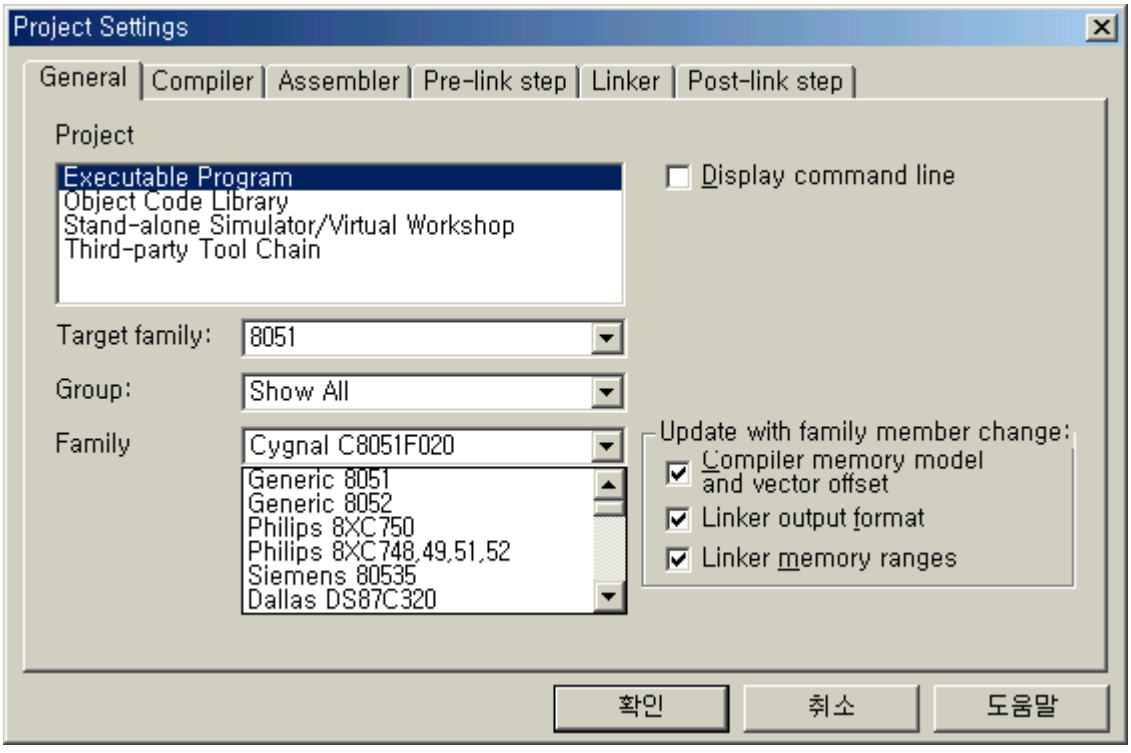


그림 35: Project Setting 윈도우

예제 3: 타이머 인터럽트의 사용 예제

- 이 예제는 Timer 3 의 오버 플로우 인터럽트를 이용하여 일정한 시간 간격으로 엔코더의 A,B 상 신호를 읽어 카운트하여 표시하는 예를 보인 것입니다. Crossware Demo 에서는 컴파일을 되지만 4K Bytes 이상 헥사파일이 출력되므로 다운로드하여 실행할 수는 없습니다. -

엔코더는 회전각을 검출하기위한 것을 로터리 엔코더라고 하며 직선 변위량을 검출하기 위한 것을 리니어 엔코더라고 합니다. 엔코더의 동작 원리는 로터리와 리니어 방식 모두 동일합니다. 엔코더는 2 개의 트랙에 서로 90 도의 위상이 차이 나도록 홈을 판 후 각각의 트랙에서 빛을 쏘아 A 와 B 신호를 출력하도록 제작되어 있습니다. 엔코더 카운터는 수신하여 수신된 신호 A, B 의 위상 차이로 이동 방향을 판단하며 펄스 수에 의하여 움직임 량을 측정합니다. 이동 방향이 바뀌는 경우에는 A,B 의 위상이 바뀌게 되어 있습니다. A, B 신호를 State Diagram 으로 만들어 나타낸 것이 그림 37 입니다. S0 는 A=0, B=0 S1 은 A = 1, B = 0 S2 는 A = 1, B = 1 그리고 S3 는 A=1, B=0 으로 정의합니다. 스테이트 다이어그램을 보면 S0 -> S1 -> S2 -> S3 로 입력된 A, B 가 변화하면 카운터 값이 증가되며 S3 -> S2 -> S1 -> S0 로 변화하면 감소하는 것입니다. 여기서 각각의 State 는 바로 옆 State 만으로 이동 해야만 되는 것을 알 수 있습니다. 만약 S0 에서 S2 로는 상태 변화가 일어나면 엔코더 시스템에 문제가 있는 경우입니다. 이런 원리를 잘 이해하면 A,B Quadrature 신호를 분해하는 하드웨어 또는 소프트웨어를 구성할 수 있습니다. 상용화된 직선거리 측정 시스템으로는 영국의 RENISHAW, 독일의 HEIDENHAIN, 일본의 SONY, 미국의 ANILAM 사가 있습니다. 그림 36 은 미국의 US Digital 사에서 생산하는 간단한 자동화용으로 사용하기 편리한 리니어 스케일이며 0.05MM 의 분해능을 가지고 있습니다. FND (7 세그멘트 LED) 8 자리로 1 개의 이송 축에 대하여 다이나믹 디스프레이 방식을 이용한 엔코더 카운터 표시장치를 만든 것입니다. -123.45 MM 임을 표시하고 있습니다. A, B 상은 기계가 빠르게 이송할 때 주파수가 높게 나오므로 A,B 상을 분해하는 디지털 하드웨어 회로를 만들거나 전용 카운터 IC 를 사용하여야 합니다. 이번 실험은 스케일의 분해능이 0.05 MM 로 공작기계에서 일반적으로 사용되는 0.001 MM 의 정밀도 보다도 낮은 것이므로 소프트웨어 방식에 의하여 A, B 상을 분해하여도 충분하며 CYGNAL 프로세서는 25 MIPS 의 고속 동작이 가능하므로 이번에 제작한 디지털 카운터 시스템은 실용화가 가능합니다. 그림 39 은 실험 제작한 디지털 카운터의 회로도입니다. 프로그램은 C 언어로 작성 되었으며 Crossware 8051 ANSI C Compiler 를 사용하여 컴파일 한것입니다. 디스프레이는 Timer3 를 이용하여 주기적으로 타이머 인터럽트를 걸리도록 하였으며 인터럽트가 걸렸을 때 다이나믹 방식으로 7 세그멘트 LED 를 구동하고 동시에 A, B 상을 입력하여 분해한 후 이동량을 알아내도록 구성되어있습니다. 이번 실험에 사용된 리니어 스케일은 미국 US Digital 사에서 제작한 플라스틱 재질의 Metric 스트립이며 헤드 암프는 미국 Agilent 사의 HEDS-9200 을 사용한 것입니다. CPU 는 미국 CYGNAL 사의 고속 프로세서 C8051F005 를 사용하여 22.1184MHz 로 동작합니다. 이 예제를 보면 7 세그멘드 FND 의 다이나믹 디스프레이 처리 방법을 다른 시스템에도 응용할수 있습니다. 8 : 1 드라이브 방식(순간적으로는 1 개의 FND 만 점등되어있음)으로 구성되어 있습니다.

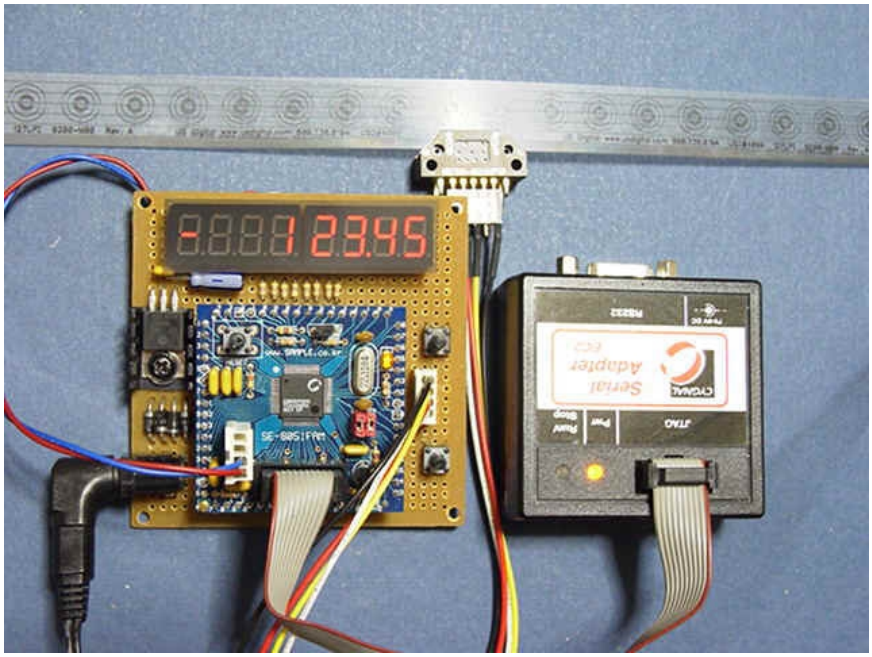
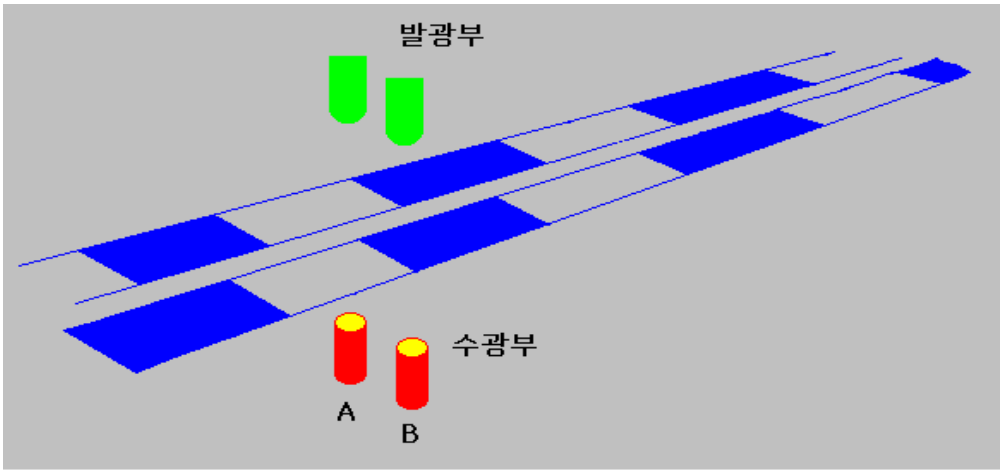
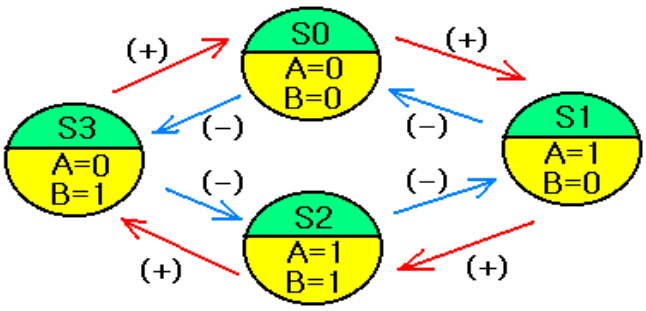


그림 36: 제작한 리니어 엔코더 카운터





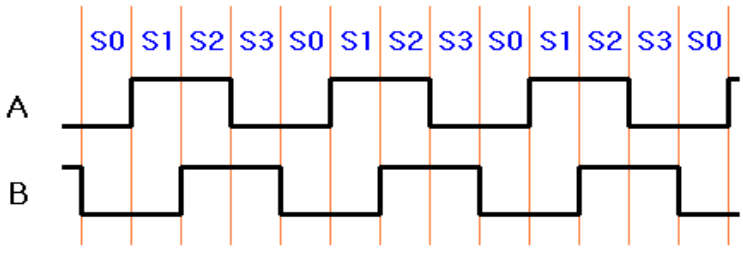


그림 37: 엔코더 동작원리

```
/*************************************************************
 *  Linear / Rotary Encoder Quadrature Counter               *
 *  (c)SAMPLE Electronics co. http://www.SAMPLE.co.kr        *
 *                                                           *
 *  Compiled on Crossware 8051 ANSI C Compiler               *
 *                                                           *
 *  이 프로그램은 로터리/리니어 엔코더에서 출력되는          *
 *  A / B 상 신호를 카운트하여 디스프레이 하는               *
 *  예를 보인것 입니다.                                      *
 *  리니어 엔코더는 US Digital 사의 Metric Type 리니어       *
 *  스트립이며 헤드암프는 Agilent 사의 HEDS-9200 을          *
 *  사용하였습니다. 정밀도는 0.05 MM 입니다.                 *
 *  A / B 상 신호는 소프트웨어에 의하여 4 체배 방식으로     *
 *  분해하여 진행 방향과 카운트 양을 계산합니다.             *
 *  디스프레이는 8 자리의 7 세그멘트 LED 를 다이나믹         *
 *  방식으로 구동합니다.                                     *
 *                                                           *
 *  Related Web Site                                         *
 *  http://www.USDIGITAL.com                                 *
 *  http://www.CROSSWARE.com                                 *
 *  http://www.CYGNAL.com                                    *
 *                                                           *
 * 1. Linear or Rotary Encoder Pattern                       *
 *   ======================================================  *
 * A |#########|         |#########|         |#########|     *
 *   ------------------------------------------------------  *
 * B      |#########|         |#########|         |#####     *
 *   ======================================================  *
 *                                                           *
 * 2. Quadrature Signal                                      *
 *   +---------+         +---------+         +---------+     *
 * A |         |         |         |         |         |     *
 *  -+         +---------+         +---------+         +-    *
 *                                                           *
 *        +---------+         +---------+         +------    *
 * B      |         |         |         |         |          *
 *  ------+         +---------+         +---------+          *
 *          S2   S3   S0   S1   S2   S3   S0   S1           *
 *                                                           *
 * 3. State Diagram                                          *
 *                                                           *
 *        /--S0---W            /--S1---W                     *
 *        | A = 0 | ---(+)---->| A = 1 |                     *
 *        | B = 0 |<---(-)---- | B = 0 |                     *
 *        W-------/            W-------/                     *
 *          /|W |               /|W |                       *
 *           |  |                |  |                       *
 *          (+)(-)              (-)(+)                      *
 *           |  |                |  |                       *
 *           | W|/               | W|/                      *
 *        /--S3---W            /--S2---W                     *
 *        | A = 0 | ---(-)---->| A = 1 |                     *
 *        | B = 1 |<---(+)---- | B = 1 |                     *
 *        W-------/            W-------/                     *
```

```
 *                                                          *
 *                                                          *
 *      P1.0 : Quadrature A                                 *
 *      P1.1 : Quadrature B                                 *
 *                                                          *
 *                                                          *
 *                                                          *
 *              P3.0 P3.1 P3.2 P3.3 P3.4 P3.5 P3.6 P3.7     *
 *              +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+     *
 *              |  | |  | |  | |  | |  | |  | |  | |  |     *
 *              +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+     *
 *              |  | |  | |  | |  | |  | |  | |  | |  |     *
 *              +--+ +--+ +--+ +--+ +--+ +--+ +--+ +--+     *
 *                 +    +    +    +    +    +    +    +     *
 *                                                          *
 ************************************************************/

// 8051 Initial C Source File
#include <stdio.h>
#include <stdlib.h>
#include <sfr.h>
#include <os.h>
//
//                  7 Segment LED 패턴 데이터
//                  hgfedcba
#define DIG0 0x3F // 00111111b  ; 0           P2.0
#define DIG1 0x06 // 00000110b  ; 1        +-----a-----+
#define DIG2 0x5B // 01011011b  ; 2        |           |
#define DIG3 0x4F // 01001111b  ; 3   P2.5 f           b P2.1
#define DIG4 0x66 // 01100110b  ; 4        |           |
#define DIG5 0x6D // 01101101b  ; 5        |   P2.6    |
#define DIG6 0x7D // 01111101b  ; 6        +-----g-----+
#define DIG7 0x27 // 00100111b  ; 7        |           |
#define DIG8 0x7F // 01111111b  ; 8   P2.4 e           c P2.2
#define DIG9 0x67 // 01100111b  ; 9        |           |
                  //            ;         |   P2.3    |
#define DIGM 0x40 // 01000000b  ; -        +-----d-----+ * h P2.7
#define DIGP 0x80 // 10000000b  ; .
#define DIGB 0x00 // 00000000b  ; "Blank"

#define LBLANK 5      // 원쪽으로 부터 5 칸을 공백처리한다.
#define RESOLUTION 5  // 정밀도 곱 상수

#define QS0 0x00   // 00 00   S0 S0+    ; . 무변화
#define QS1 0x01   // 00 01   S0 S1+    ; + 증가
#define QS3 0x03   // 00 11   S0 S2+    ; ?
#define QS2 0x02   // 00 10   S0 S3+    ; - 감소

#define QS4 0x04   // 01 00   S1 S0+    ; - 감소
#define QS5 0x05   // 01 01   S1 S1+    ; . 무변화
#define QS7 0x07   // 01 11   S1 S2+    ; + 증가
#define QS6 0x06   // 01 10   S1 S3+    ; ?

#define QSC 0x0C   // 11 00   S2 S0+    ; ?
```

```
#define QSD 0x0D   // 11 01   S2 S1+    ; - 감소
#define QSF 0x0F   // 11 11   S2 S2+    ; . 무변화
#define QSE 0x0E   // 11 10   S2 S3+    ; + 증가

#define QS8 0x08   // 10 00   S3 S0+    ; + 증가
#define QS9 0x09   // 10 01   S3 S1+    ; ?
#define QSB 0x0B   // 10 11   S3 S2+    ; - 감소
#define QSA 0x0A   // 10 10   S3 S3+    ; . 무변화

const char segment_pattern[] = { DIG0, DIG1, DIG2, DIG3, DIG4, DIG5, DIG6, DIG7, DIG8, DIG9 };
const char segment_select[8] = { 0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE };

char segment_buffer[8];
unsigned char segment = 0;
unsigned char pq;
long int counter = 0;  // 인터럽트 루틴 내부에서만 사용됩니다.
long int task = 0;
bit flag = 0;

void _InitSerialPort(void);

void _interrupt IVN_TIMER3 segment_display() { // Timer 3 오버플로우 인터럽트

  unsigned char t, cq;

     _tmr3cn &= 0x7F;               // Timer 3 Overflpw Flag Clear

     _p3 = segment_buffer[segment];
     _p2 = segment_select[segment];

      segment += 1;
      segment &= 0x07;

            cq = _p1 & 0x3;
            t = (pq << 2) | cq;
            pq = cq;

            switch(t) {
              case QS1: counter++; break;  // 증가
              case QS2: counter--; break;  // 감소
              case QS4: counter--; break;  // 감소
              case QS7: counter++; break;  // 증가
              case QSD: counter--; break;  // 감소
              case QSE: counter++; break;  // 증가
              case QS8: counter++; break;  // 증가
              case QSB: counter--; break;  // 감소
            }

        if(flag == 0) {
          task = counter;
          flag = 1;
        }
}
```

```
void display(long int d) {

char s;
char i;
char t[10];

        if(d < 0) {                 // 부호를 저장해놓고
          s = -1;                   // 음수인 경우 양수로 변환한다.
          d = labs(d);
        } else {
            s = 0;
          }

        sprintf(t,"%8li",d);        // 8 개의 자리수로 스트링 프린트

        for (i = 0; i < 8; i++) {   // 7 세그멘트 패턴으로 변환
           t[i] = segment_pattern[(t[i] & 0x0F)];
        }

        for(i = 0; i < LBLANK; i++) { // 왼쪽 수가 "0" 일때 제거
         if (t[i] == DIG0) {
             t[i] = DIGB;
         }else break;
        }

        if(s < 0) { t[0] = DIGM; }  // 음수인경우 "-" 표시

        t[LBLANK] |= DIGP;          // 소수점 "." 표시

        do {
          if(segment == 0) {
            for ( i = 0; i < 8; i++) {
              segment_buffer[i] = t[i];
            }
            break;
          }
        }while (1);
}

void main(void) {

       _wdtcn = 0xDE;       // Watch Dog Disable
       _wdtcn = 0xAD;       //
//     _oscxcn = 0x65;      // XFCN = '101' for 11.0592 MHz crystal
       _oscxcn = 0x66;      // XFCN = '110' for 22.1184 MHz crystal
       _oscicn = 0x08;      // 내장 오실레이터 금지하고 외부 크리스탈 사용
       _xbr0 = 0x04;        // UART Enable
       _xbr2 = 0x40;        // Crossbar Enable
       _prt3cf = 0xFF;      // Port 3 Push-Pull Mode

       _ref0cn = 0x03;      // Internal Band Gap On Reference Buffer On
       _dac0cn = 0x80;      //

       _tmr3rll = 0xA0;     // Timer 3 Auto Re-load register low
```

```
        _tmr3rlh = 0xFF;      // Timer 3 Auto Re-load register high
        _tmr3l = 0x00;        // Timer 3 Start Value Low
        _tmr3h = 0x00;        // Timer 3 Start Value High
        _tmr3cn |= 0x04;      // Timer 3 Run
        _eie2 |= 0x01;        // Timer 3 Interrupt Enable

        _InitSerialPort();    // 11.0592MHz -> 19200 BPS
                              // 22.1184MHz -> 38400 BPS

         do{
           if(flag){
            task = task * RESOLUTION; // 정밀도 끝수를 곱한다.
            display(task);
            flag = 0;
           }
         }while(1);

}
```

그림 38: 소프트웨어로 AB 상을 분해하여 엔코더의 진행 방향과 펄수를 표시하는 프로그램

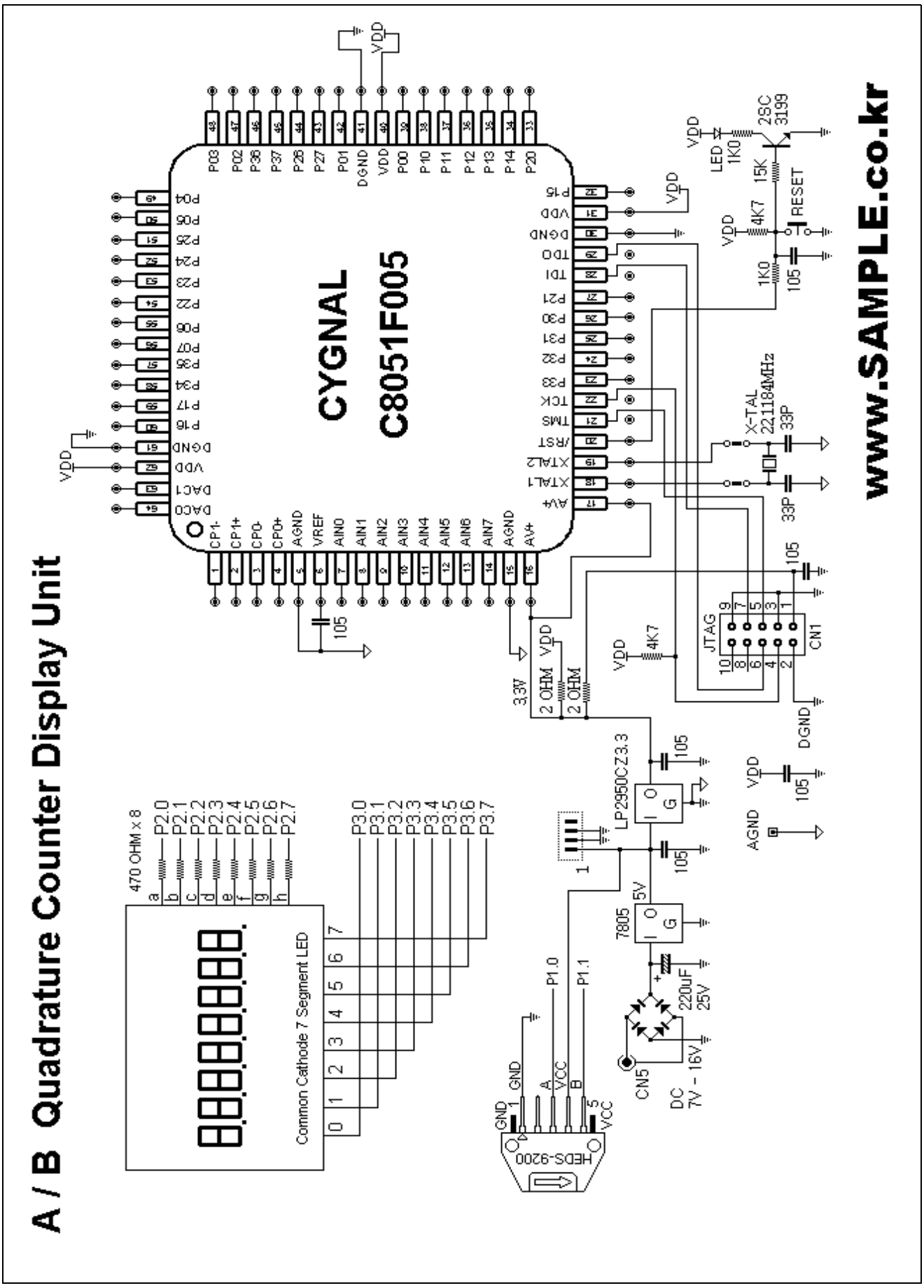


그림 39: 엔코더 카운터 회로도

예제 4: 시리얼 포트를 이용한 데이터 입출력 실험

8051 마이크로프로세서는 시리얼 입출력 포트를 가지고 있습니다. CYGNAL 의 C8051F020, C8051F021 은 2 개의 시리얼 포트를 가지고 있습니다. 이 예제는 시리얼 포트를 이용하여 PC 와 스트링 데이터를 교신하는 프로그램 입니다. 새 프로젝트를 만든후 그림 44 와 같이 프로그램 합니다. 컴파일에 이상이 없으면 CYGNAL 프로세서 (SE-8051FUP 또는 Development Kit)에 프로그램을 다운로드 합니다. CYGNAL 프로세서와 PC 의 시리얼 포트를 연결하기 위하여 프로세서의 Power 를 OFF 합니다. 그리고 EC-2 에 연결된 시리얼 케이블을 CYGNAL 프로세서에 연결합니다. 그리고 Power 를 ON 합니다.(EC-2 와 시리얼 케이블을 연결하거나 분리할 때 반드시 Power 를 OFF 하여야 안전합니다.) 그림 40, 그림 41 과 같이 시리얼 포트를 설정합니다. 통신속도를 4800 으로 설정합니다. 그리고 그림 42, 그림 43 과 같이 터미널 에뮬레이터를 동작합니다. CYGNAL 프로세서 (SE-8051FUP 또는 Development Kit)를 리셋하면 그림 43 과 같이 초기 메시지 스트링이 표시됩니다. 그리고 PC 에 키보드에서 문자를 입력하고 “ENTER”키를 누루면 입력한 문자가 CYGNAL 프로세서로 입력된후 다시 PC 로 출력되어 터미널 에뮬레이터에 나타납니다.

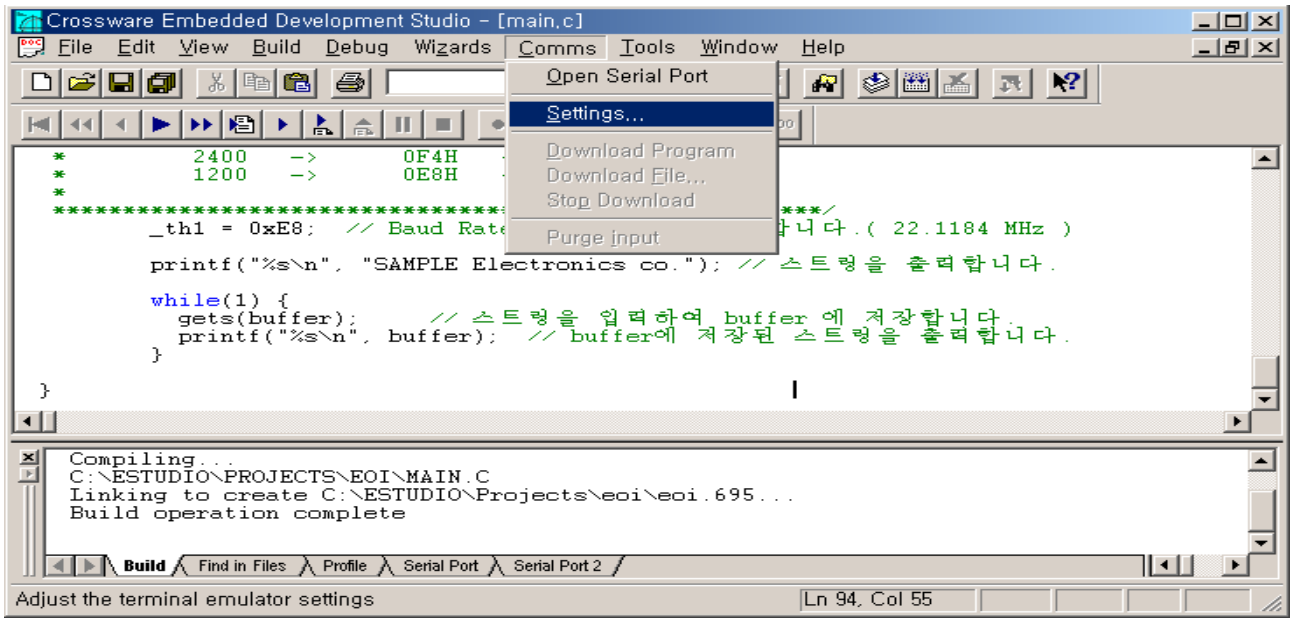


그림 40: 터미널 에뮬레이터 파라메터 설정

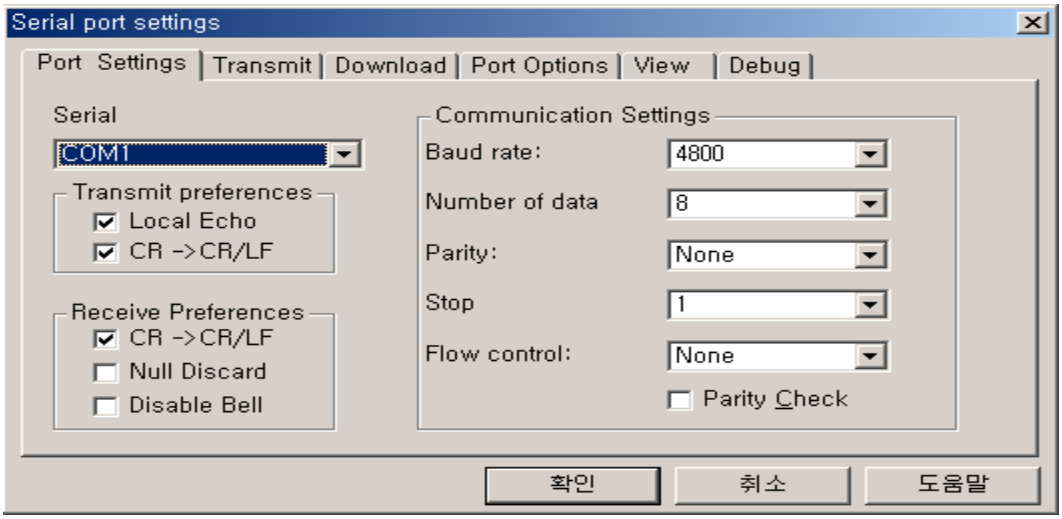


그림 41: 통신 속도 설정

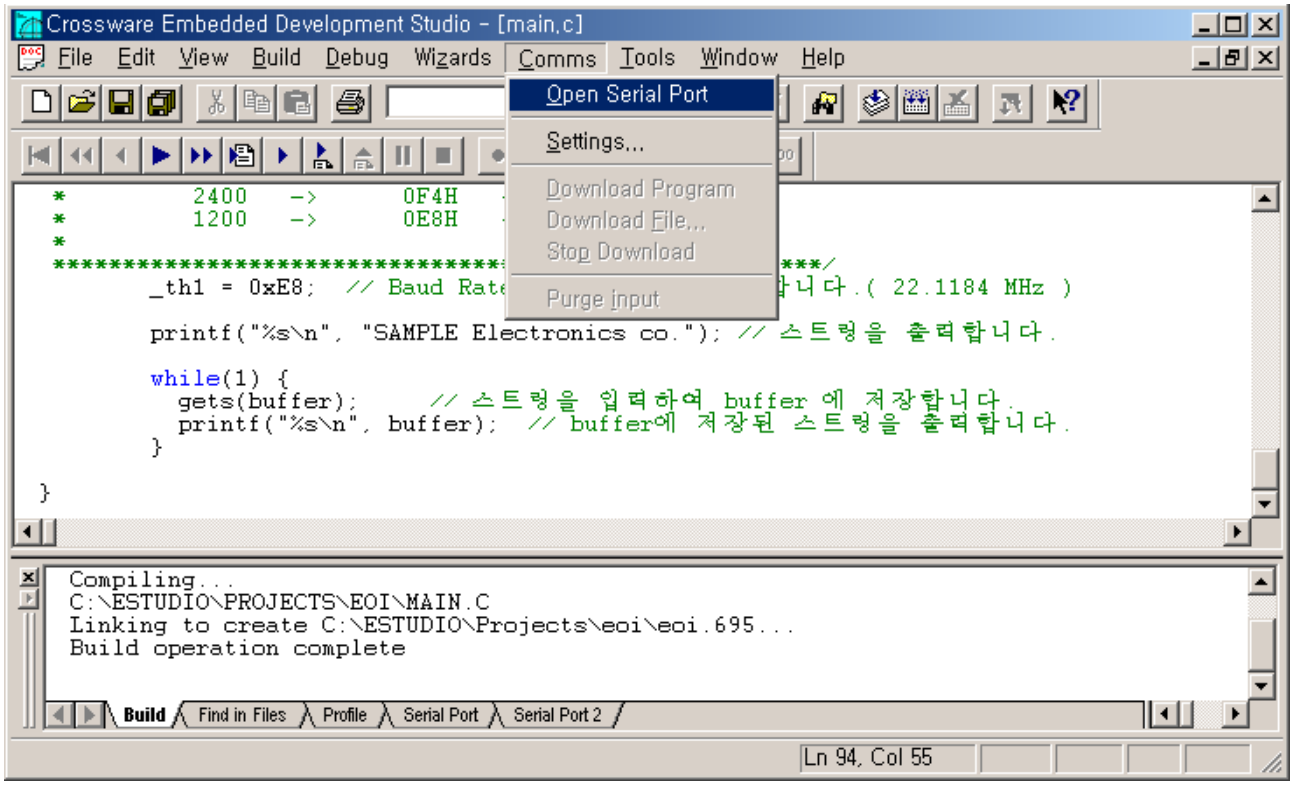


그림 42: 터미널 에뮬레이터

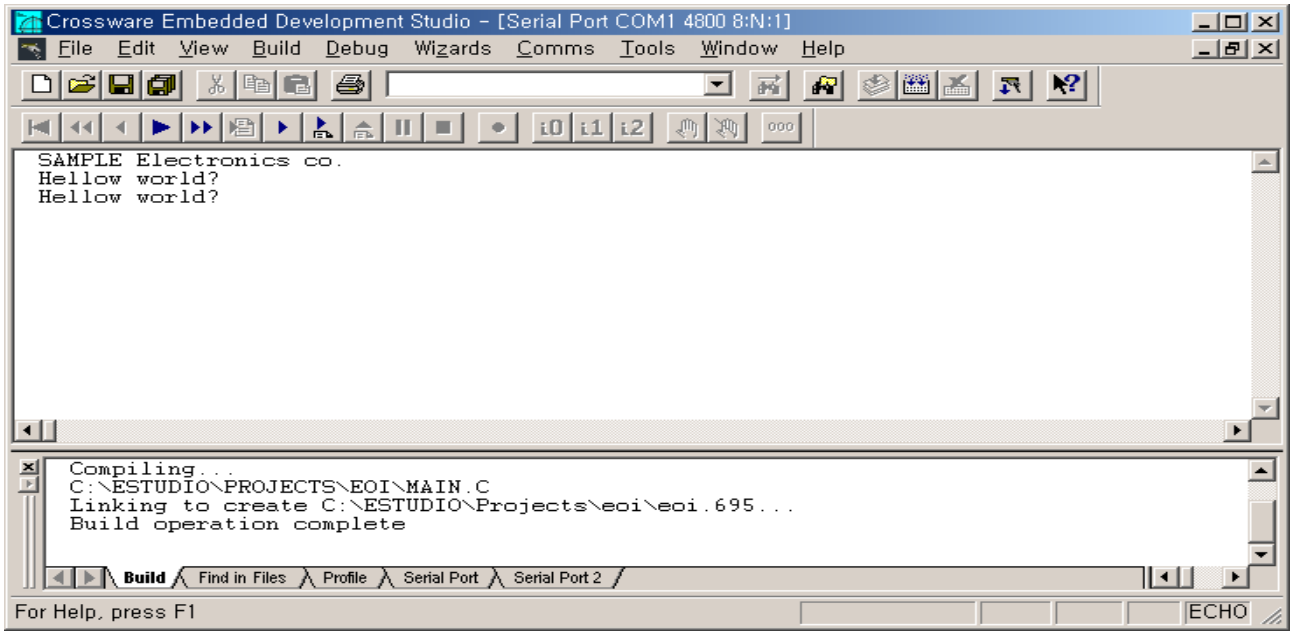


그림 43: 터미널 에뮬레이터 동작 윈도우

```
/* Interrupt driver 8051 serial communications
   (c) 1995 - 1997 Crossware Associates */


#include <os.h>
#include <conio.h>
```

```
#include <sfr.h>


#define TRUE 1
#define FALSE 0

static unsigned char m_nReceivedCharacter;
static unsigned char m_bByteReceived;
static unsigned char m_bByteTransmitting;

void _interrupt IVN_SERIALPORT _using 2 _SerialInterrupt()
{
        if (_ri)
        {
                // receive interrupt if here
                _ri = 0;
                m_nReceivedCharacter = _sbuf;
                m_bByteReceived = TRUE;
        }
        else if (_ti)
        {
                // transmit interrupt if here
                _ti = 0;
                m_bByteTransmitting = FALSE;

        }
}

// with interrupts
// no xon/xoff

int putch(int c)
{
        if (c == '\n')
        {
                //putch('\r');
                // avoid recursion so that works with reentrant and non-reentrant libraries
                while (m_bByteTransmitting);      // wait for previous byte to complete
                _sbuf = '\r';
                m_bByteTransmitting = TRUE;
        }
        while (m_bByteTransmitting);      // wait for previous byte to complete
        _sbuf = c;
        m_bByteTransmitting = TRUE;
        return c;
}

int getch()
{
        while (!m_bByteReceived);// wait for a byte if not already there
        m_bByteReceived = FALSE;
        return (int)m_nReceivedCharacter;
}
```

```
int kbhit()
{
        return (int)m_bByteReceived;      // return true if character waiting in buffer
}



void _InitSerialPort()
{
        m_nReceivedCharacter = 0;
        m_bByteReceived = FALSE;
        m_bByteTransmitting = FALSE;
        _scon = 0X50;    // SCON serial port control - 8 bit uart (1 stop bit)
        _tl1 = 0XFD;             // Timer1 lo
        _th1 = 0XFD;     // 9600 baud
        _pcon |= 0X80;   // 19200 baud (SMOD == 1)
        _tmod = 0X20;    // 8 bit autoreload for timer 1 ( baud rate generator)
        _tcon = 0;
        _tr1 = 1;                // start baud rate generator
        _es = 1; // enable serial port interrupts
        _ea = 1; // enable interrupts
}
```

그림 44: source.c 에 있는 InitSerialPort 함수

```
/*****************************************************
 * 이 프로그램은 CYGNAL 프로세서의 시리얼 포트를 이용하는
 * 예제 입니다. 시리얼 포트를 통하여 입력된 스트링
 * 데이터를 외부로 출력합니다.
 *
 * 실험 하드웨어 : 25MIP CYGNAL 프로세서를 사용한
 *         Evaluation Kit 또는 SE-8051FUP
 *
 * 외부 오실레이터 주파수 : 22.1184MHz
 *
 * Step 1: 새프로젝트를 만든후 이 파일을 입력하여
 *         컴파일/다운로드 합니다.
 * Step 2: Evaluation Kit의 Power 를 OFF 하고
 *         시리얼 포트를 PC 의 Com1 에 연결합니다.
 * Step 3: Crossware Windows 메뉴바의 Comms -> Settings ->
 *         Port Setting을 1 로 하고 Baud를 4800 으로 설정합니다.
 * Step 4: Comms -> Open Serial Port를 클릭하면 터미널
 *         에뮬레이터 창이 나옵니다.
 * Step 5: Evaluation Kit 의 POWER를 연결하면 터미널
 *         에뮬레이터 윈도우에 "SAMPLE Electronics co."가 표시됩니다.
 * Step 6: PC 의 키보드에서 문자키를 여러개 입력한후 ENTER키를
 *         누르면 PC에서 입력한 문자가 Evaluation Kit 의
 *         시리얼 포트로 전송되며 다시 PC 로 반송되어
 *         터미널 에뮬레이터에 나타납니다.
 *
 *****************************************************/

// 8051 Initial C Source File

#include <stdio.h>
```

```
#include <sfr.h>            // Include Register Definition FILE

void _InitSerialPort(void);

main()
{
char buffer[100];           // 스트링 저장용 버퍼 영역

        _wdtcn = 0xDE;       // Watch Dog 타이머의 동작을 금지합니다.
        _wdtcn = 0xAD;
        _oscxcn = 0x66;      // 외부에 연결된 오실레이터
        _oscicn = 0x08;      // 주파수를 사용합니다.
        _xbr0 = 0x04;        // UART0 를 사용가능하게 합니다.
        _xbr2 = 0x40;        // Crossbar 를 ON 합니다.

        _InitSerialPort();   // 시리얼 포트 UART 를 초기화 합니다.

/*********************************************************
 *         SERIAL PORT INITIALIZE
 *         BAUD RATE TABLE
 *
 *        _InitSerialPort() 함수의 Baud Rate는 XTAL 주파수가
 *         22.1184MHz 일때 19200 BPS 이며
 *         11.0592MHz 일때 9600 BPS 로 기본 설정되어 있습니다.
 *         Baud Rate 를 변경하려면 _InitSerialPort()함수를
 *         먼저 실행한후 TH1 을 변경하여 주면 됩니다.
 *
 *         Baud Rate 는
 *         (1) XTAL1 에 입력되는 주파수,
 *         (2) PCON 레지스터의 SMOD 비트
 *         (3) Timer 1 의 TH1 레지스터의 설정되는
 *             값에 의하여 결정됩니다.
 *
 *         Xtal = 22.1184 MHz 일때 TH1 설정값
 *
 *         SMOD = 0     TH1 Reload   SMOD = 1
 *
 *        57600    ->      0FFH   <-    115200
 *        19200    ->      0FDH   <-     28800
 *         9600    ->      0FAH   <-     19200
 *         4800    ->      0F4H   <-      9600
 *         2400    ->      0E8H   <-      4800
 *
 *         Xtal = 11.0592 MHz 일때 TH1 설정값
 *
 *         SMOD = 0     TH1 Reload   SMOD = 1
 *
 *        28800    ->      0FFH   <-     57600
 *         9600    ->      0FDH   <-     19200
 *         4800    ->      0FAH   <-      9600
 *         2400    ->      0F4H   <-      4800
 *         1200    ->      0E8H   <-      2400
 *
 *********************************************************/
```

```
        _th1 = 0xE8;  // Baud Rate 를 4800 으로 변경 합니다.( 22.1184 MHz )

        printf("%sWn", "SAMPLE Electronics co."); // 스트링을 출력합니다.

        while(1) {
          gets(buffer);     // 스트링을 입력하여 buffer 에 저장합니다.
          printf("%sWn", buffer);  // buffer 에 저장된 스트링을 출력합니다.
        }

}
```

그림 45: RS-232 시리얼 입출력 실험

예제 5: 인터럽트 INT0 와 INT1 의 사용법

인터럽트는 정상적인 프로그램을 실행하다가 급한 작업을 실행하기 위하여 현재의 프로그램 실행을 중지하고 별도로 작성해둔 인터럽트 서비스 루틴을 실행하는 것입니다. 대표적인 예는 PC 의 키보드에서 키가 눌려지면 데이터가 전송되어 스크린에 나타나는 것를 구성한 것입니다. 프로세서가 키보드로부터 언제 전송되어 올지도 모를 데이터를 받기 위하여 기다릴 수가 없으므로 인터럽트(시리얼 포트 데이터 수신 인터럽트)를 이용하여 메인 프로그램을 실행하다가 데이터가 들어오면 인터럽트 서브루틴을 실행하여 데이터를 메모리에 저장하여 놓고 메인 프로그램으로 복귀합니다. 인터럽트의 이용은 매우 중요하며 고급 제어 시스템일수록 복잡하고 정교하게 인터럽트를 사용합니다. 외부 핀을 이용한 인터럽트의 경우 프로세서에게 인터럽트를 처리해줄 것을 요청하는 방법으로 LEVEL 인터럽트와 EDGE 인터럽트의 2 가지 방법이 있습니다. LEVEL 인터럽트란 프로세서에게 인터럽트를 걸 때 0 레벨을 입력하는 방법이며 EDGE 인터럽트는 논리 레벨이 변경되었을 때 (1 에서 0)인터럽트를 발생하는 것입니다. 인터럽트를 시작하는 관점에서만 보면 양쪽이 동일한 것 같지만 큰 차이가 있습니다. 첫째로 인터럽트를 요청하는 펄스의 폭 입니다. LEVEL 입력으로 인터럽트를 구성하면 인터럽트 펄스폭이 그림 48 과 같이 인터럽트 검사 주기 보다 작을 경우 인터럽트는 실행되지 않습니다. 그러나 EDGE 모드에서는 논리 변화(1 에서 0 으로) 상태가 생기면 인터럽트가 발생하는 것이므로 인터럽트 요청신호의 폭이 작아도 인터럽트 서비스 루틴은 확실하게 실행됩니다. 두번째로 인터럽트 요청 펄스폭이 길 때 입니다. 인터럽트 서브루틴 실행중에 INTx 가 1 로 되면 관계없습니다만 인터럽트 서비스 루틴이 실행 완료된 이후에도 INTx 가 0 상태를 유지하고 있다면 그림 46 과 같이 다시 인터럽트 실행 서브루틴이 반복해서 실행됩니다. 인터럽트 요청은 한번 이지만 INTx 레벨이 1 로 복귀하지 않는다면 프로세서는 주 프로그램으로 복귀하지 못하고 무한정 동일한 인터럽트 서브루틴을 반복 실행하게 됩니다. (INTx 가 1 이 될때까지) EDGE 인터럽트 모드에서는 INTx 핀이 0 상태를 계속 유지하거나 1 상태를 유지 하더라도 관계없습니다. 즉 그림 47 과 같이 인터럽트 실행은 INTx 가 1 에서 0 으로 변화된 순간 발생하며 한번만 인터럽트 서브스루틴이 실행됩니다. LEVEL 인터럽트는 그림 49 와 같이 여러 개의 인터럽트 발생원을 1 개의 인터럽트 핀으로 처리할 때 사용합니다. 인터럽트 발생원 A, B, C 중 한나만 인터럽트가 발생하거나 또는 동시에 A, B, C 가 모두 인터럽트 요청이 발생하면 프로세서는 인터럽트 서비스 루틴을 실행하기 시작합니다. 인터럽트 서비스 루틴은 먼저 A 가 인터럽트를 요청하였는지 확인합니다. A 가 인터럽트를 요청했다는 것이 확인되면 A 에 해당되는 인터럽트 서비스루틴을 실행하고 인터럽트 발생원 A 가 INTx 로 인터럽트 신호를 내보내지 않도록 합니다. 그리고 메인 프로그램으로 복귀합니다. 그런데 아직도 인터럽트 요구 신호가

있다면 다시 인터럽트 서비스루틴이 실행하여 B, C 에 대하여 같은 방법으로 인터럽트 서비스 루틴을 처리합니다. 모든 인터럽트 요청에 대한 서비스가 완료되었다면 인터럽트 신호는 1 이되어 메인 프로그램으로 복귀할수 있게 됩니다. 일반적으로 16/32 비트 프로세서는 LEVEL 인터럽트만 있어서 프로세서 외부에서 여러 개의 인터럽트를 처리하는 하드웨어를 가지고 동작하도록 구성되어 있습니다. 마이크로컨틀롤러는 EDGE 인터럽트 핀을 여러 개 가지고 있습니다. 표준 8051 은 인터럽트 핀이 2 개 있으며 설정에 의하여 LEVEL 모드 EDGE 모드를 선택할 수 있습니다. 표준 8051 은 P3.2 가 INT0 이며 P3.4 가 INT1 으로 지정되어 있습니다. CYGNAL 프로세서의 경우 Crossbar 의 구성에 따라 INT0, INT1 의 핀 배치가 달라집니다

그림 46: LEVEL 인터럽트 동작원리

그림 47: EDGE 인터럽트 동작원리

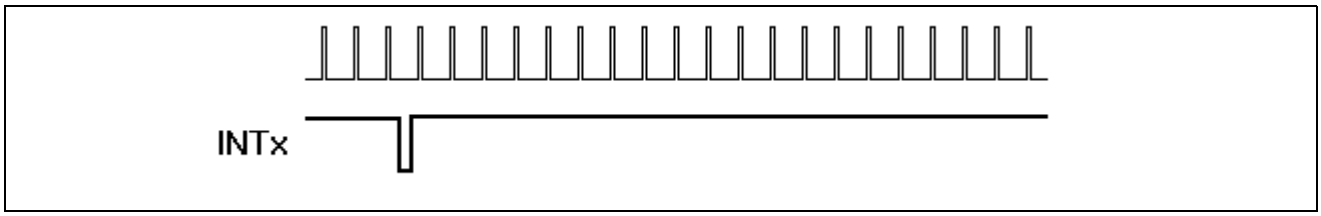


그림 48: LEVEL 모드에서 짧은 인터럽트 요청 신호

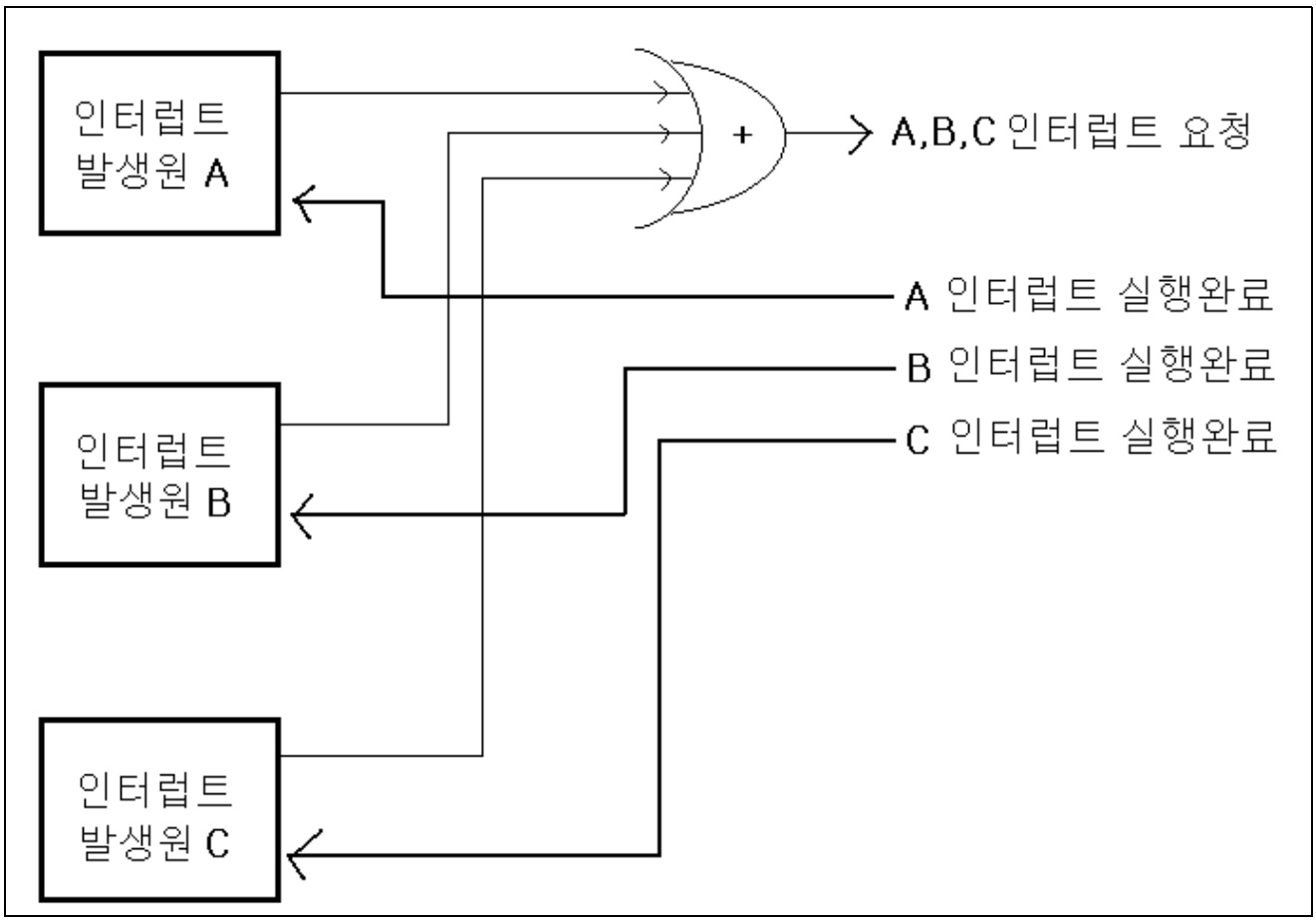


그림 49: 여러 개의 인터럽트로 구성

CROSSWARE 의 Wizard 기능을 이용하면 간편하게 인터럽트 처리 프로그램을 작성하는 것이 가능 합니다. 메뉴바에서 FILE-> NEW 를 클릭하여 프로젝트를 구성합니다. 프로젝트 구성이 완료되면 초기 main.c 가 만들어집니다. 이번 예제는 Wizards 기능을 이용하여 인터럽트의 동작을 확실히 배울수 있도록 프로그램합니다.

STEP 1: Watchdog Timer 동작금지 - 커서를 main()함수 안의 첫번째 라인으로 놓습니다. 그리고 윈도우 메뉴에서 Wizards-> Watchdog 을 클릭하면 그림 59 와 같은 Watchdog Timer 윈도우가 나옵니다. “Enable” 버튼을 클릭하여 Watchdog 을 사용하지 않는 조건으로 설정합니다. 그리고 생성된 Configuration Code 가 현재 편집중인 파일의 커서 위치로 삽입되도록 “Insert before cursor”를 선택하고 “Insert” 버튼을 클릭합니다.

STEP 2: Crossbar 설정 - 커서를 main()함수 안의 두번째 라인으로 놓습니다. 그리고 윈도우 메뉴에서 Wizards-> Ports and Crossbar 를 클릭하면 그림 60 의 윈도우가 나옵니다. /INT0, /INT1, Enable 을 선택합니다. 그리고“Insert before cursor” 를 선택하고 “Insert”버튼을 클릭합니다.

STEP 3: INT0, INT1 동작모드 설정 - 커서를 main()함수 안의 세번째 라인으로 놓습니다. 그리고 윈도우 메뉴에서 Wizards-> External Interrupt 0,1 를 클릭하면 그림 61 의 윈도우가 나옵니다. Mode 에서 Falling Edge 를 선택합니다. Interrupt 를 Enable 로 선택하고 Global Interrupt 도 선택합니다. 이제 INT1 탭을 클릭하여 그림 62 가 나오면 Interrupt 를 Enable, Mode 를 Falling 으로 선택합니다. 그리고“Insert before cursor” 를 선택하고 “Insert”버튼을 클릭합니다.

STEP4: 인터럽트 서비스 루틴 작성 - 그림 65 와 같이 INT0, INT1 의 인터럽트 서비스 루틴을 입력합

니다. CYGNAL 프로세서는 표준 8051 에 비하여 많은 인터럽트 발생원을 가지고 있습니다. 각각의 번호는 그림 64 의 Include 파일 os.h 파일을 참고합니다.

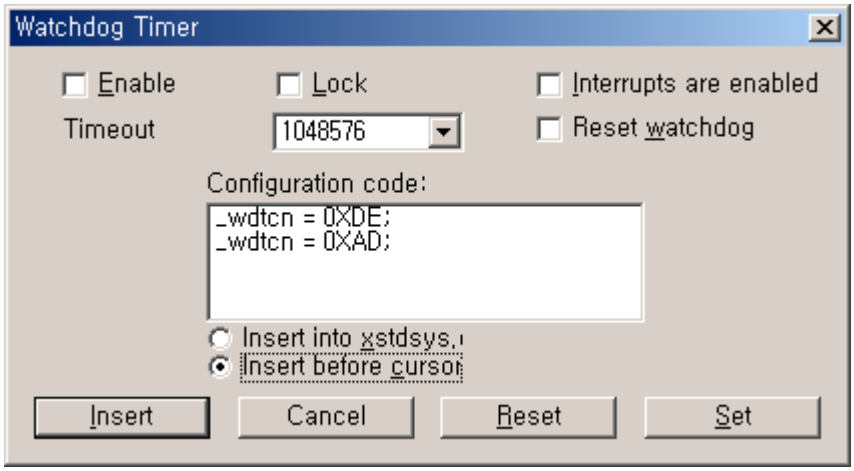


그림 59: Watchdog 타이머 동작금지

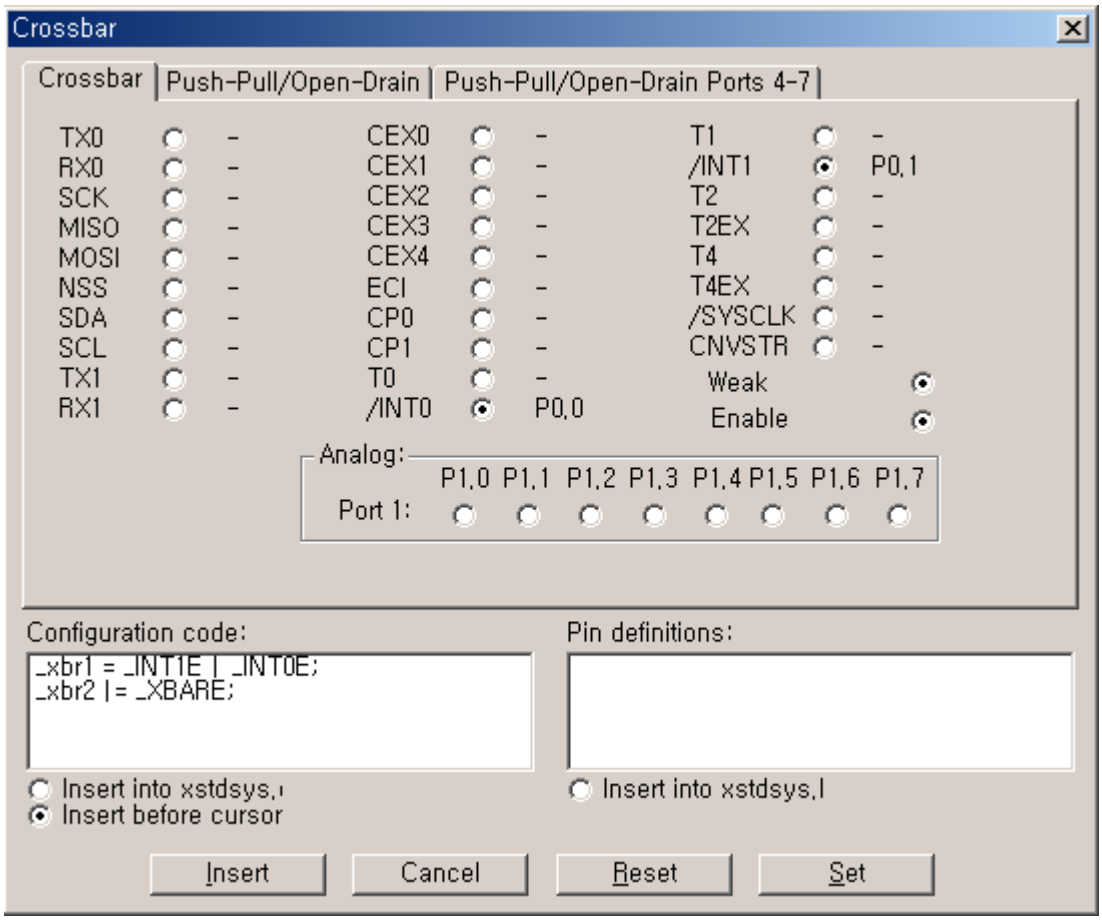


그림 60: 크로스바 설정

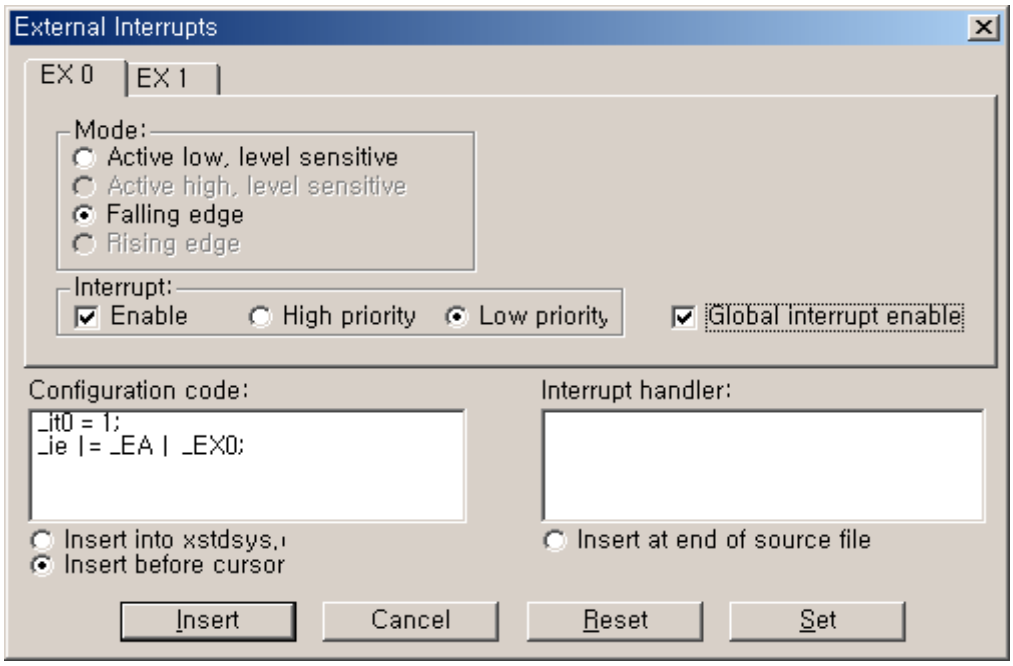


그림 61: INT0 동작 모드 설정

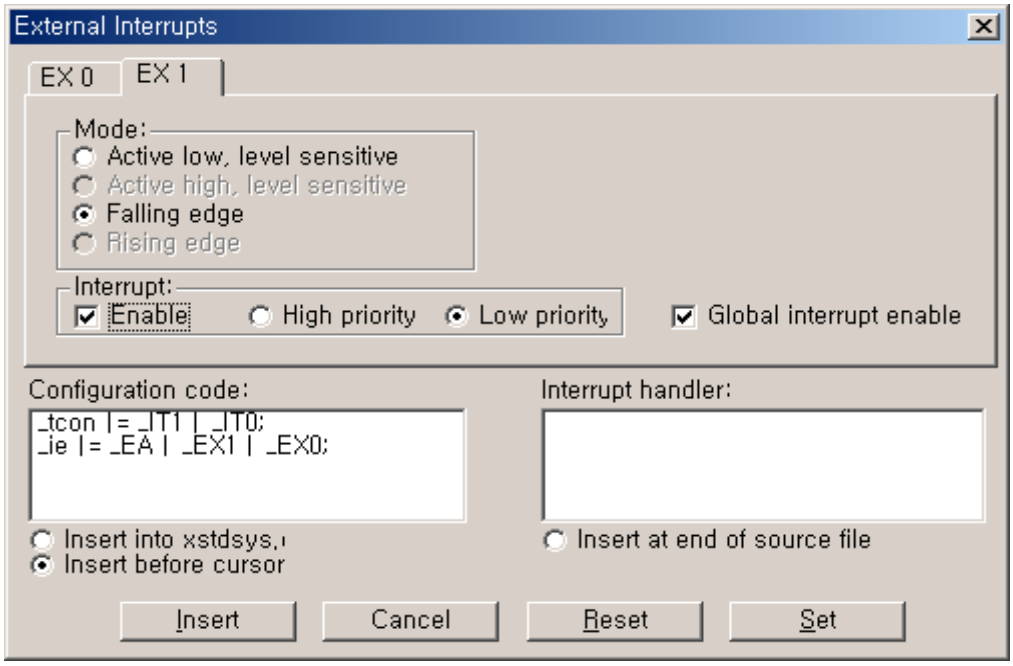


그림 62: INT1 동작 모드 설정

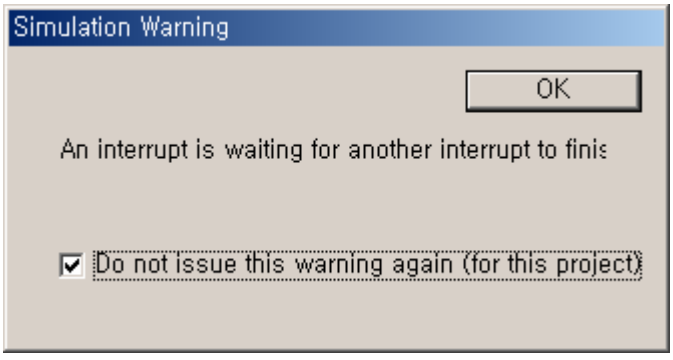


그림 63: 인터럽트 경고 메시지 윈도우

```
/* Copyright (c) 2001 Crossware Associates */

/* Interrupt vector numbers */

/* Definitions for _interrupt n for Cygnal chips
 *
 * For example:
 *  void _interrupt IVN_TIMER1 Timer1InterruptHandler()
 *  {
 *  }
 */

#ifdef __C8051F02X

#define IVN_INTERRUPT0      0
#define IVN_TIMER0          1
#define IVN_INTERRUPT1      2
#define IVN_TIMER1          3
#define IVN_SERIALPORT      4
#define IVN_SERIALPORT0     4
#define IVN_TIMER2          5
#define IVN_SPI             6
#define IVN_SMBUS           7
#define IVN_ADC0WINDOW      8
#define IVN_PCA0            9
#define IVN_CP0FALLING      10
#define IVN_CP0RISING       11
#define IVN_CP1FALLING      12
#define IVN_CP1RISING       13
#define IVN_TIMER3          14
#define IVN_ADC0CONVERSION      15
#define IVN_TIMER4          16
#define IVN_ADC1CONVERSION  17
#define IVN_INTERRUPT6      18
#define IVN_INTERRUPT7      19
#define IVN_SERIALPORT1     20
#define IVN_OSC             21

#elif defined(__C8051F0XX)

#define IVN_INTERRUPT0      0
#define IVN_TIMER0          1
#define IVN_INTERRUPT1      2
#define IVN_TIMER1          3
#define IVN_SERIALPORT      4
#define IVN_SERIALPORT0     4
#define IVN_TIMER2          5
#define IVN_SPI             6
#define IVN_SMBUS           7
#define IVN_ADC0WINDOW      8
#define IVN_PCA0            9
#define IVN_CP0FALLING      10
#define IVN_CP0RISING       11
#define IVN_CP1FALLING      12
```

```
#define IVN_CP1RISING       13
#define IVN_TIMER3          14
#define IVN_ADC0CONVERSION      15
#define IVN_INTERRUPT4      16
#define IVN_INTERRUPT5      17
#define IVN_INTERRUPT6      18
#define IVN_INTERRUPT7      19
#define IVN_OSC             21

#elif defined(__C8051F2XX)

#define IVN_INTERRUPT0      0
#define IVN_TIMER0          1
#define IVN_INTERRUPT1      2
#define IVN_TIMER1          3
#define IVN_SERIALPORT      4
#define IVN_SERIALPORT0     4
#define IVN_TIMER2          5
#define IVN_SPI             6
#define IVN_ADC0WINDOW      8
#define IVN_CP0FALLING      10
#define IVN_CP0RISING       11
#define IVN_CP1FALLING      12
#define IVN_CP1RISING       13
#define IVN_ADC0CONVERSION      15
#define IVN_INTERRUPT4      16
#define IVN_INTERRUPT5      17
#define IVN_INTERRUPT6      18
#define IVN_INTERRUPT7      19
#define IVN_OSC             21

#elif defined(__C8051F3XX)

#define IVN_INTERRUPT0      0
#define IVN_TIMER0          1
#define IVN_INTERRUPT1      2
#define IVN_TIMER1          3
#define IVN_SERIALPORT      4
#define IVN_SERIALPORT0     4
#define IVN_TIMER2          5
#define IVN_SMBUS           6
#define IVN_ADC0WINDOW      7
#define IVN_ADC0CONVERSION      8
#define IVN_PCA0            9
#define IVN_CP0FALLING      10
#define IVN_CP0RISING       11

#else

#error  No appropriate variant defined

#endif
```

그림 64: CYGNAL 프로세서용 os.h 헤더파일

```
/************************************************************
 이 프로그램은 CYGNAL 프로세서에서 INT0, INT1 을 이용하는
 예제입니다. CROSSWARE 의 Wizards 를 이용하여 인터럽트 사용환경
 구성을 보여줍니다.
 Step 1: Watchdog 타이머를 금지합니다.
 Step 2: Crossbar 를 설정합니다. (INT0, INT1 사용, Crossbar ON)
 Step 3: 인터럽트 INT0, INT1 의 동작모드를 Falling Edge 로 선택합니다.
 Step 4: 인터럽트 INT0, INT1 를 1 로 설정하고 EA 를 1 로하여여
         모든 인터럽트가 가능 상태가 되게 합니다.
 다운로드가 완료되면 P0.0(INT0), P0.1(INT1)을 GND 레벨로 입력하면
 Port1 과 Port2 의 상태가 변화합니다.

SAMPLE Electronics co.                  http://www.SAMPLE.co.kr
************************************************************/
// 8051 Initial C Source File
#include "xstdsys.h"

void _interrupt IVN_INTERRUPT0 interrupt_0() { // INT0 인터럽트 서비스 루틴

        _p1 = 0xff;          // P1 의 LED 를 Off
        _p2 = 0x00;          // P2 의 LED 를 On
}

void _interrupt IVN_INTERRUPT1 interrupt_1() { // INT1 인터럽트 서비스 루틴

        _p1 = 0x00;          // P1 의 LED 를 On
        _p2 = 0xff;          // P2 의 LED 를 Off
}

main()
{
        _wdtcn = 0XDE;                // Watch dog 타이머 금지
        _wdtcn = 0XAD;
        _tcon |= _IT1 | _IT0;         // Int0, Int1 의 Falling Edge 모드설정
        _ie |= _EA | _EX1 | _EX0;     // Int0, Int1 의 사용가능 상태 설정
        _xbr1 = _INT1E | _INT0E;      // Int0, Int1 을 P0.0, P0.1 에 배치
        _xbr2 |= _XBARE;              // Crossbar ON

        _p1 = 0x00;                   // P1 의 LED 를 On
        _p2 = 0x00;                   // P2 의 LED 를 On

        // use the Wizards (see Wizards menu) to configure the microcontroller
        while (1)
        {

        }
}
```

그림 65: 인터럽트 실험용 프로그램

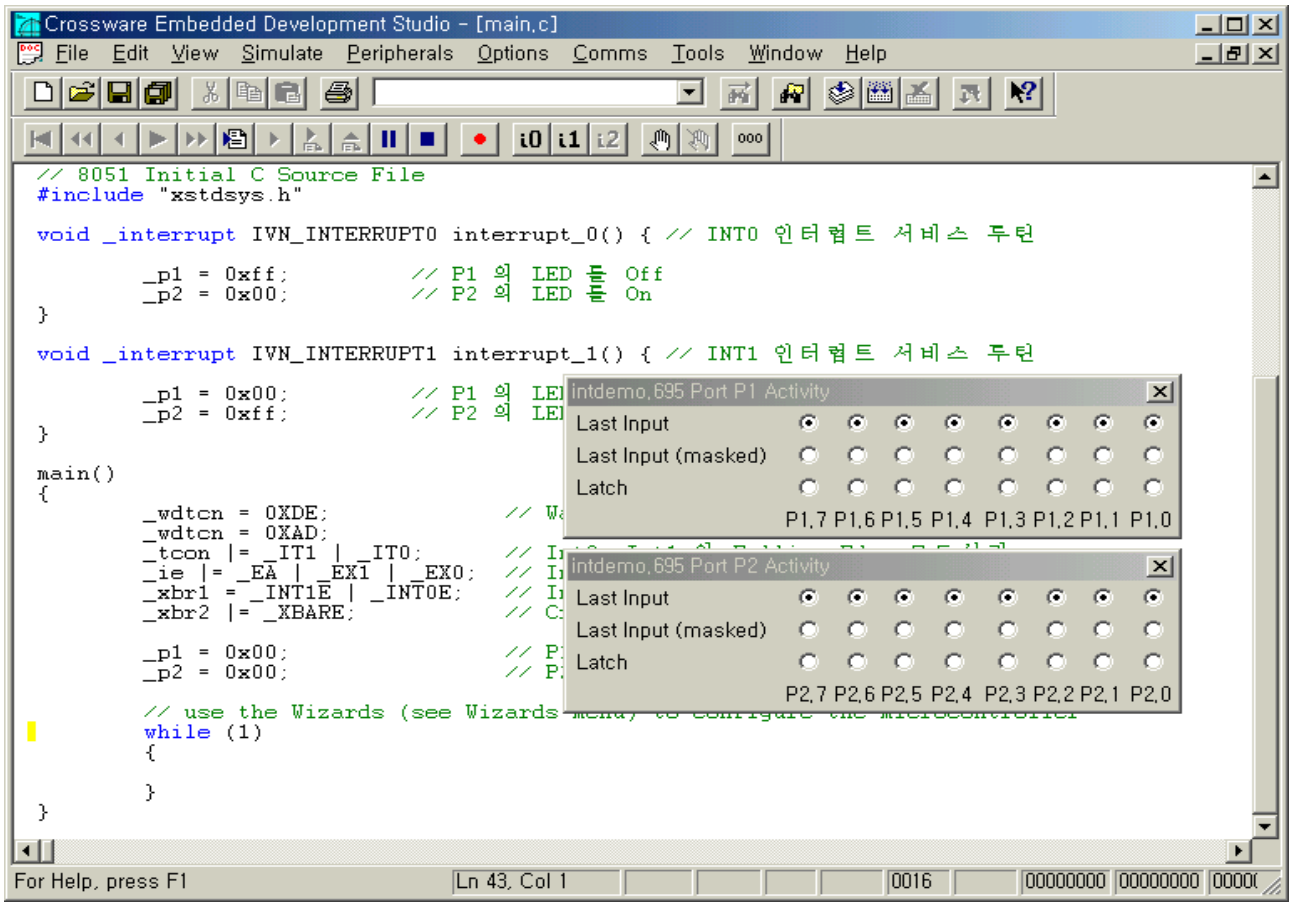


그림 66: 인터럽트가 걸리기전 상태 P1 = 0x00, P2 = 0x00

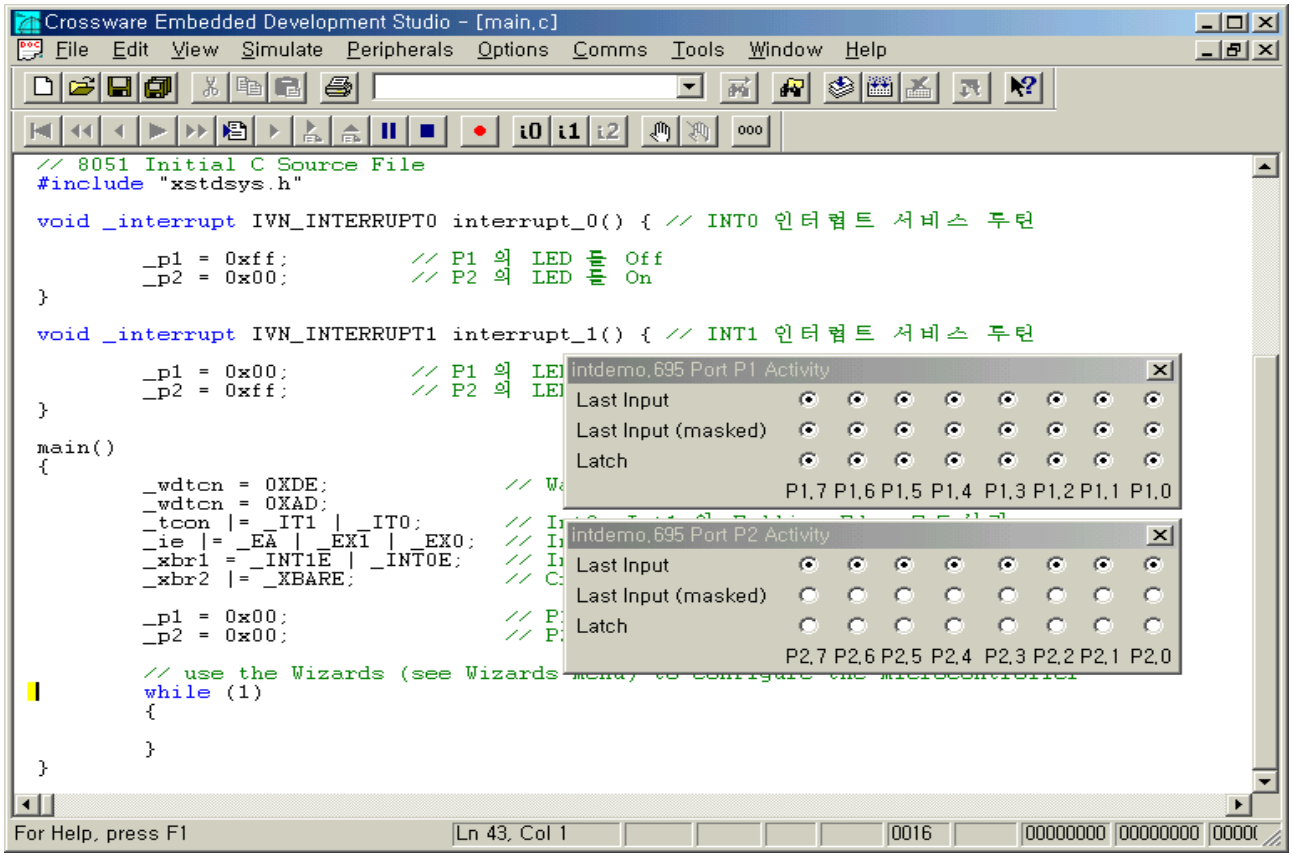


그림 67: INT0 발생시 P1 = 0xFF, P2 = 0x00

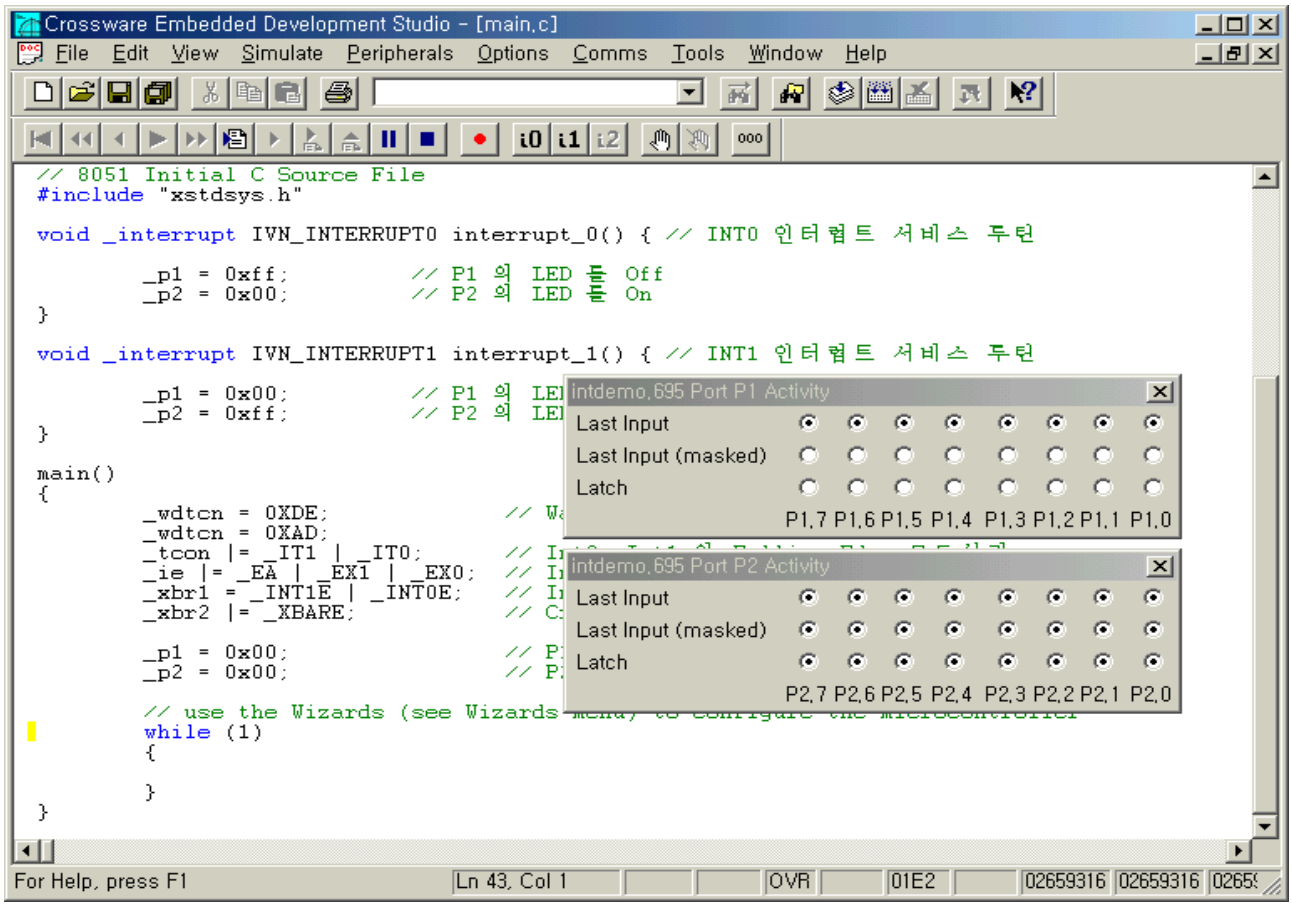


그림 68: INT1 발생시 P1 = 0x00, P2 = 0xFF

프로그램이 완성되었으면 컴파일합니다.  를 클릭하여 프로그램을 실행하면 그림 66 과 같이 됩니다. P1 과 P2 가 모두 0 인 것을 알 수 있습니다. INT0 신호에 해당하는 신호는  를 클릭하면 그림 67 과 같이 됩니다. INT0 의 인터럽트 서비스루틴에서 P1 의 LED 는 OFF(0xff), P2 의 LED 는 ON(0x00) 한 것입니다.  를 클릭하면 그림 68 과 같이 P1 의 LED 는 ON (0x00), P2 의 LED 는 OFF0xff) 됩니다. 인터럽트 시뮬레이션 시 그림 63 과 같은 경고 메시지 윈도우가 나오면 경고 메시지가 다시 나오지 않도록 체크하고 “OK”를 클릭합니다. 시뮬레이션을 정지하려면  를 클릭하면 됩니다. 이 프로그램을 SE-8051FUP 에서 실험하려면 Simmulate 를 Debug 로 전환하여 실행 아이콘  를 클릭하면 됩니다. SE-8051FUP 의 UART1 컨넥터에 테스트 클립 커넥터를 연결하고 중간(GND 에 연결됨) 클립은 P0.0 과 P0.1 에 접촉하여 인터럽트를 걸어보면 P1 과 P2 의 LED 점등 상태가 각각의 인터럽트 서비스 루틴의 실행 결과로 변화 됩니다.

The Crossware C8051NT is an ANSI standard C compiler that generates code for the 8051 family of microcontrollers. It provides numerous extensions that allow access to 8051 specific features so that you can write your code completely in the C language without the need to resort to assembler code. It supports both small and large memory models so you can create code for all 8051 variants whether or not they have external ram.

- The ANSI C compiler protects your investment by ensuring future portability of your C source.
- 8051 specific extensions enable you to access all of the resources of the 8051 and its variants from C.
- Smart pointers which can work out for themselves which memory area they are pointing to. This enables you to avoid inefficient 3 byte pointers yet still write ANSI C compatible code.
- Generates fast in-line code with a minimum of library calls for high speed performance on your target board or optionally uses library routines to minimize code size for those parts of your program where speed is less important.
- Full type checking across modules traps programming errors and ensures that your C variables, function arguments, structure members, etc are consistent across your source files and with the appropriate libraries.
- Source level debug output in IEEE695 format.
- Pre-written library routines, including high speed 32 bit and 64 bit floating point arithmetic.
- Register bank independent code.
- Easy to use C interrupt support with optional register bank switching.
- Small and large memory models.
- Place variables in any memory area.
- Supports all 8051 variants.
- Wide range of output formats.
- Integrated relocatable assembler.
- Highly user friendly *Embedded Development Studio* integrated development environment (see separate data sheet).

Data output for Embedded Development Studio browser.

CROSSWARE 의 8051 C 컴파일러는 IEEE695 파일을 출력합니다. IEEE695 는 8051 개발 장비 제조 회사인 Microtec 과 HP 가 공동으로 제정한 디버깅 정보가 포함된 파일 규격입니다. 미국 전기/전자 기술자 협회(IEEE)에 IEEE695 로 등록되었으며 모든 MDS/ICE 제조 업체에서 표준으로 지원합니다. IEEE695 는 비트, 스트링, 부동소숫점, 스트럭쳐, 유니온, 배열 변수를 정의한 데이터 포맷 그대로 디버깅 할 수 있습니다. 부동소수점 데이터를 헥사값으로 표시하면 디버깅 작업은 사실상 무의미 합니다. IEEE695 는 부동소숫점 데이터는 십진수로 표시합니다.

매뉴얼에는 8051 마이크로 프로세서에서 C 언어를 응용할 수 있는 예제가 포함되어 있습니다.

예제 1: 새 프로젝트의 구성법

새 프로젝트를 구성하는 방법에 관하여 설명되어 있습니다. 예제 프로그램은 CYGNAL C8051F020 프로세서의 64 개의 입출력 포트에 연결된 LED 를 순차적으로 ON/OFF 합니다.

예제 2: 부동소숫점, 스트링, Structure, Union 변수, 지역, 전역 변수의 디버깅

예제 프로그램은 Crossware ANSI C 컴파일러에서 지원하는 변수 형식과 연산자의 사용법을 보여 줍니다. 비트변수, 정수변수, 부동소숫점, 스트링 그리고 Structure 변수 디버깅과 Union 변수를 이용하여 16 비트 정수를 상위바이트 하위바이트로 나누어 P1, P2 에 출력하는 방법에 대하여 설명합니다.

예제 3: 타이머 인터럽트의 사용

정해진 시간 주기에 프로그램을 시행할 때 타이머 인터럽트를 사용합니다. 리니어 로터리 엔코더에서 출력되는 A, B 상 신호를 소프트웨어적인 방법으로 분해하여 8 단 7 세그멘트 FND 에 다이나믹 방식으로 디스프레이하는 엔코더 카운터를 제작합니다.

예제 4: 시리얼 포트를 이용한 데이터 입출력 실험

인터페이스 방식으로 가장 많이 사용되는 RS-232 시리얼 인터페이스를 이용하는 방법에 관하여 설명합니다. PC 의 시리얼 포트(Com1 포트) 와 연결하여 데이터를 주고 받는 실험입니다. Crossware 의 시뮬레이터를 이용하여도 데이터 입출력 실험이 가능합니다.

예제 5: 인터럽트 INT0 와 INT1 의 사용법

인터럽트의 개념에 대하여 설명합니다. 인터럽트는 LEVEL 인터럽트 방식과 EDGE 인터럽트 방식이 있으며 이 두가지 방식에 대한 분명한 차이점과 8051 마이크로 프로세서에서 외부 Port 핀 입력을 이용한 인터럽트 사용법이 설명되어 있습니다.