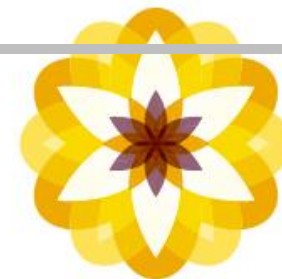
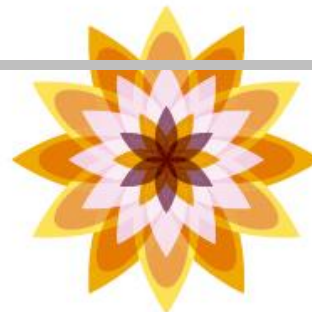


Chapter 01

ARM Cortex-M3 소개



목차

1. 시리얼 통신이란 무엇입니까?
2. 시리얼 통신 규격
3. 신호 배치 및 커넥터
4. 연결 방법
5. 비동기 통신 및 동기식 통신

1. ARM 역사

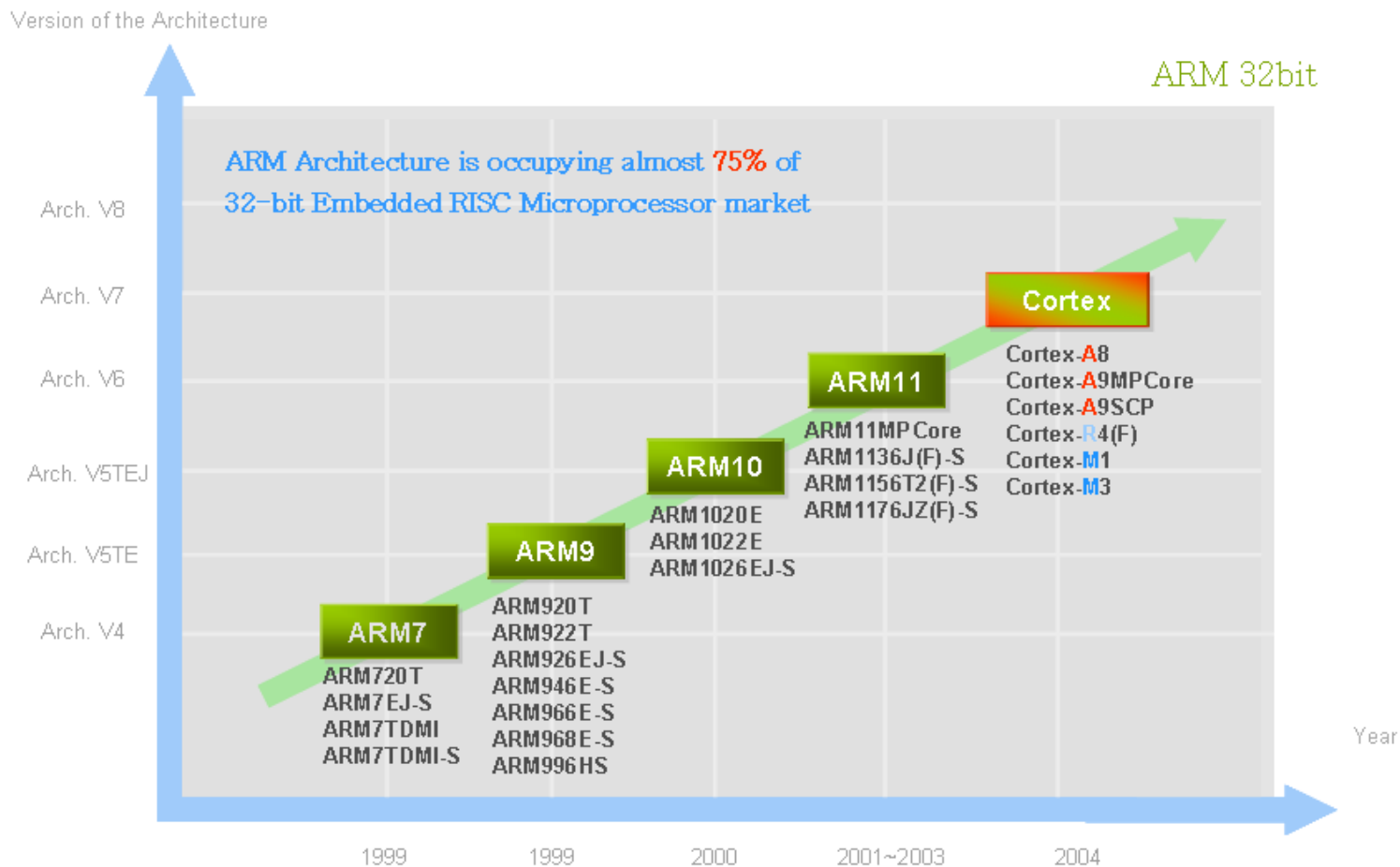
- ARM(Advanced RISC Machine) 아키텍처는 임베디드 기기에 많이 사용되는 32bit RISC프로세서이다.
- 이 프로세서는 저전력을 사용하고 현재 32bit 임베디드 시장에서 독보적인 시장 점유율을 가지고 있다.
- ARM Limited는 현재 영국 케임브리지에 본사가 위치하고 있고 1990년 설립되었다.
- ARM Limited는 프로세서를 제조/판매하는 것으로 유명하지만 이 외에도 리얼뷰 및 케일 브랜드 하에서 여러 가지 소프트웨어 개발 도구도 라이선스하고 있다
- 현재 ARM은 프로세서를 직접 생산하지 않고 코어를 라이선스하고 있으며 이 코어를 라이선스 한 파트너 기업을 통해 필요한 기능을 추가하여 칩에 내장하는 다양한 프로세서 제품군들을 생산하고 있다.
- 이것을 SoC (System-on-Chip)이라 말하기도 한다.

1. ARM 역사

- 2006년 ARM에서 출시된 프로세서의 첫 번째 Cortex 세대인 ARM Cortex-M3 프로세서는 기본적으로 32비트 마이크로프로세서 시장을 타겟으로 설계되었다.
- Cortex-M3 프로세서는 낮은 게이트 수로 훌륭한 성능을 제공할 뿐만 아니라 하이엔드 프로세서에서만 사용할 수 있었던 다양한 특징들이 추가되었다.
- 그 특징들은 아래와 같다.
 - 첫째, 성능 효율성을 높이고 속도, 전력은 이전 수준을 유지하면서도 더 높은 작업이 가능하도록 하였다.
 - 둘째, ARM 특유의 저전력의 구조는 휴대용 제품에서의 강점을 그대로 유지하도록 하였다.
 - 셋째, 결정성을 개선하고 인터럽트 처리 시간을 혁신적으로 단축하였다.
 - 넷째, 코드 집적도를 더욱 높이고 극소형의 메모리에서도 코드가 실장될 수 있도록 하였다.
 - 다섯째, 사용자 편의성을 극대화하기 위해 프로그래밍과 디버깅을 수월하게 하였다.
 - 여섯째, 8비트나 16비트의 시장에 대한 가격 경쟁력을 부여하기 위하여 가격대를 비슷하게 설정하였으며 1달러 미만으로 출시되는 최초의 32비트 마이크로컨트롤러가 되었다.
 - 일곱째, 개발 툴에 대한 선택의 폭을 다양하게 하여 무상 컴파일러에서부터 다양한 개발 툴을 사용하도록 지원하였다.

2. ARM 제품군

- ARM사는 현재 ARM7, ARM9, ARM10, ARM11, Cortex 제품군을 시장에 출시해 놓고 있다.



2. ARM 제품군

■ Cortex

- ARM사 차기 프로세서 제품군인 지금까지의 명명법과는 다르게 cortex는 이름을 가지고 태어났다.
- 그 이유에 대해서는 여러 가지 이야기 있지만 재미있는 이야기 중에 하나가 짝수버전의 ARM제품군들의 판매 및 성과가 좋지 않아서 ARM12라는 이름 대신 Cortex라는 이름의 제품군을 출시했다는 이야기도 있다.
- Cortex 시리즈 제품군의 가장 큰 특징은 기존 8bit, 16bit 프로세서 시장 및 32bit 고성능 프로세서 시장까지 다양한 시장을 공략할 수 있는 제품군으로 구성되어 있다는 것이다.
- Cortex의 제품군으로는 ARM Cortex-A8, ARM Cortex-A9 MPCore, ARM Cortex-A9 Single Core Processor, ARM Cortex-M1, ARM Cortex-M3 and ARM Cortex-R4(F)가 출시되었거나 출시 대기 중이며 NEC, nVIDIA, STMicroelectronics, Samsung, Texas Instruments, Toshiba등의 세계적인 업체들이 현재 라이선스하고 있다.
- Cortex 시리즈 중 먼저 A시리즈에는 ARM Cortex-A8, ARM Cortex-A9 MPCore, ARM Cortex-A9 Single Core Processor가 존재 하며 강력한 임베디드 운영체제와 응용프로그램 사용하기에 최적화한 시리즈이다. A시리즈는 기본적으로 ARM, Thumb의 명령어 세트 뿐 아니라 Thumb-2명령어 세트도 지원하고 있다.
- 두 번째 소개할 시리즈는 R시리즈로 ARM Cortex-R4(F)가 존재하면 리얼타임시스템에 최적화되어있는 시리즈이다. R시리즈 역시 A시리즈처럼 ARM, Thumb의 명령어 세트 뿐 아니라 Thumb-2명령어 세트도 지원하고 있다.
- 마지막으로 M시리즈를 소개하자면 ARM Cortex-M1, ARM Cortex-M3가 존재하며 저렴한 가격대에 프로세서 시장은 목표한 제품이다. 이 제품의 경우 현재 8bit, 16bit 프로세서 시장에서 32bit 시장으로 진출하려는 전자, 전기, 통신 업체들이 보다 싼 비용으로 32bit 제품을 출시 할 수 있게 도와줄 수 있는 시리즈이다.

2. ARM 제품군

■ Cortex

- Cortex-M3 명령어 세트는 새로운 특징을 많이 가지고 있다. 먼저 최초로 하드웨어 나눗셈 명령어가 ARM 프로세서에서 가능하게 되었다.
- 이전의 명령어 세트에서는 덧셈, 뺄셈, 곱셈만 지원하였다.
- 또 데이터 고속처리 성능을 향상시키기 위한 많은 곱셈 명령어들도 사용 가능하게 되었다.
- 뿐만 아니라 하이-엔드 프로세서에서만 가능하였던 특징인 비정렬 데이터 접근도 지원한다.

2. ARM 제품군

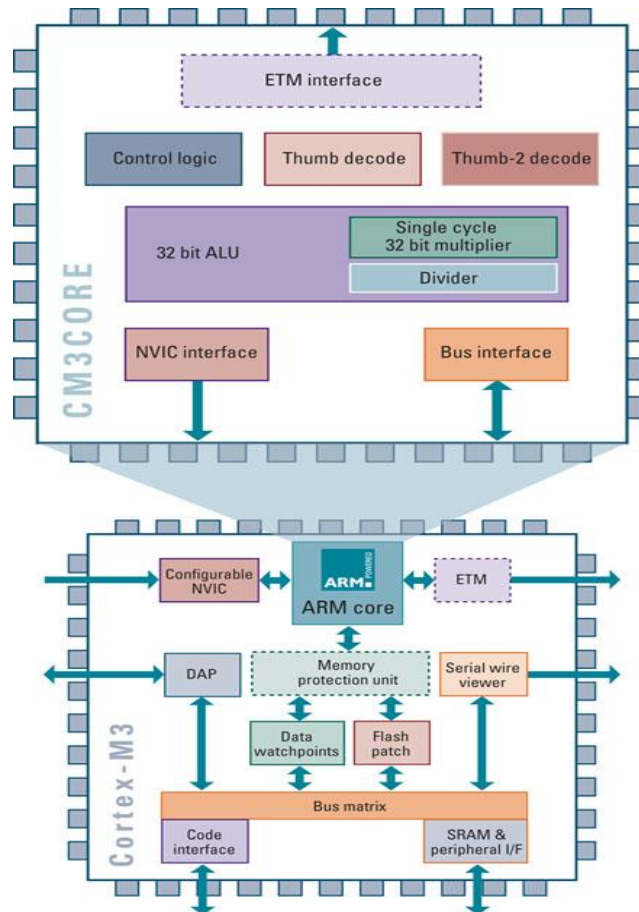
■ Cortex-M3

- ARM Cortex-M3는 32bit RISC 프로세서로써 ARMv7-M 아키텍처 기반으로 제작된 첫 번째 프로세서로써 Actel, Broadcom, Luminary, NXP, STMicroelectronics, Texas Instruments, Toshiba등 14개 업체에서 라이선스하고 있다.
- ARM Cortex-M3는 저가형 MCU시장을 공략하기 위한 모델로써 ARMv7-M아키텍처를 최적화하여 ARM7TDMI보다 30% 저렴한 가격을 형성하지만 ARM7TDMI 보다 뛰어난 성능을 갖추고 있다.
- 기능 적인 능력을 보면 ARM Cortex-M3는 3단계의 파이프라인을 기반으로 하여 하버드 아키텍처로 설계되어 있으며 Thumb-2 명령어세트를 사용하며 기존에 사용하던 ARM 명령어세트는 지원하지 않으며 Thumb코드로 제작되어 있는 코드는 활용이 가능하다.
- 인터럽트는 240개의 물리적 사용 할 수 있으며 이 인터럽트를 256단계의 우선순위로 정의될 수 있다.
- 또한 인터럽트 대기 시간을 최대한 줄이기 위한 Tail Chaining을 제공해 준다.
- 기본적으로 메모리 보호 장치인 MPU를 제공하고 있지만 MMU는 지원하지 않고 있어 가상메모리를 요구하는 임베디드 운영체제를 탑재할 수 없다.
- 또한 고성능의 프로세서의 문제점인 전력문제를 해결하기 위해 많은 노력을 하였고 슬립 모드 또한 지원하고 있다.
- 파이프라인의 특성상 분기 시 버려지는 명령어를 최소화하기 위해 동적 분기 예측 기능을 가지고 있으며 이 기능은 90%이상의 정확도를 가지고 있다.

2. ARM 제품군

■ Cortex-M3

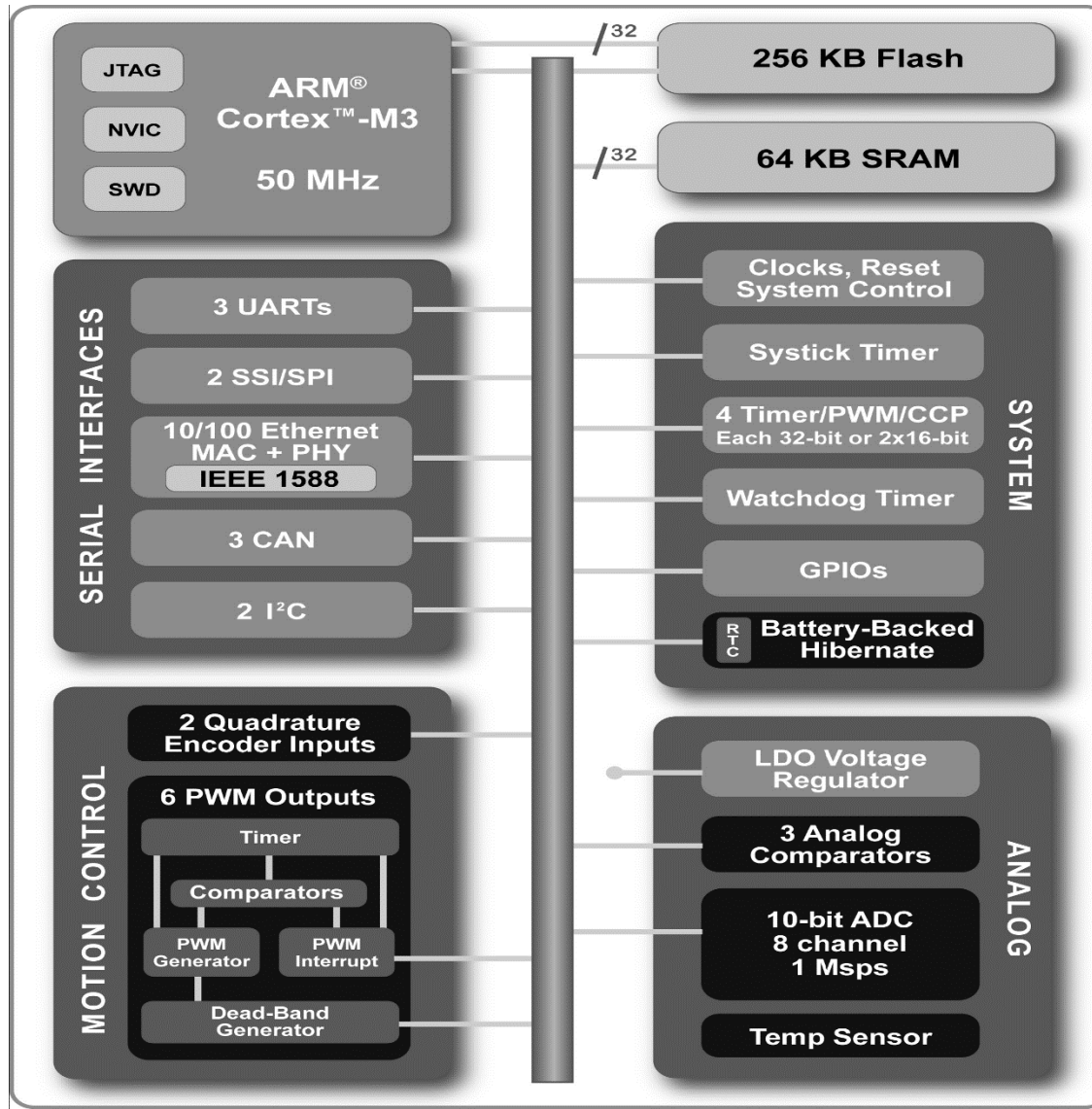
- ARM Cortex-M3 활용분야를 보면 8비트 비용을 가지고 32비트를 성능을 요구하는 MCU업체를 주 대상으로 하며 Bluetooth, Zigbee등의 저전력 작업과 복잡한 스택을 지원해야 하는 무선 네트워킹 분야, 보완 신뢰성 등이 중요한 자동차 및 산업용 제어 시스템, 높은 안정성을 요구하는 의료 계측분야, 복잡한 수학적 알고리즘을 사용하여 높은 성능을 기대하는 시스템등에 활용 가능할 것이다.



3. LM3S8962 Micro Controller

- Luminary Micro Stellaris(ARM Cortex-M3 기반) 제품군은 가격에 민감한 임베디드 마이크로컨트롤러 어플리케이션에 고성능의 32비트 컴퓨팅을 제공한다.
- 8비트와 16비트 제품과 유사한 가격대에서 32비트 성능을 제공하면서 시스템의 크기는 더욱 작게 설계할 수 있는 장점이 있다.
- Stellaris 제품군은 저렴한 가격임에도 불구하고 제품이 요구하는 충분한 프로세싱 성능과 네트워크 연결기능 등이 통합된 효율적인 성능을 제공한다.
- Stellaris LM3S8000 시리즈는 10/100 이더넷 미디어 액세스 제어(MAC) 및 물리적(PHY) 계층과 함께 컨트롤러 영역 네트워크(CAN) 기술을 결합한 제품이다.
- LM3S8962 마이크로컨트롤러는 ARM의 Cortex-M3를 채택하여, ARM용 개발툴을 활용할 수 있고, Thumb 명령어 및 Thumb-2 명령어를 지원함으로써 효율적인 메모리활용을 통한 비용절감 효과도 기대할 수 있다.

3. LM3S8962 Micro Controller



4. 개발환경 구축

■ Target Board 설치

- Target Board를 아래 그림과 같이 연결하여 설치 한다.
- Target Board에 전원 케이블을 연결하고(USB 전원으로 대체 가능), PC와 USB 케이블을 이용하여 연결하게 된다.
- 이 때 USB 케이블은 JTAG기능과 USB2UART 2가지 기능을 제공하게 된다.



4. 개발환경 구축

■ 드라이버 설치

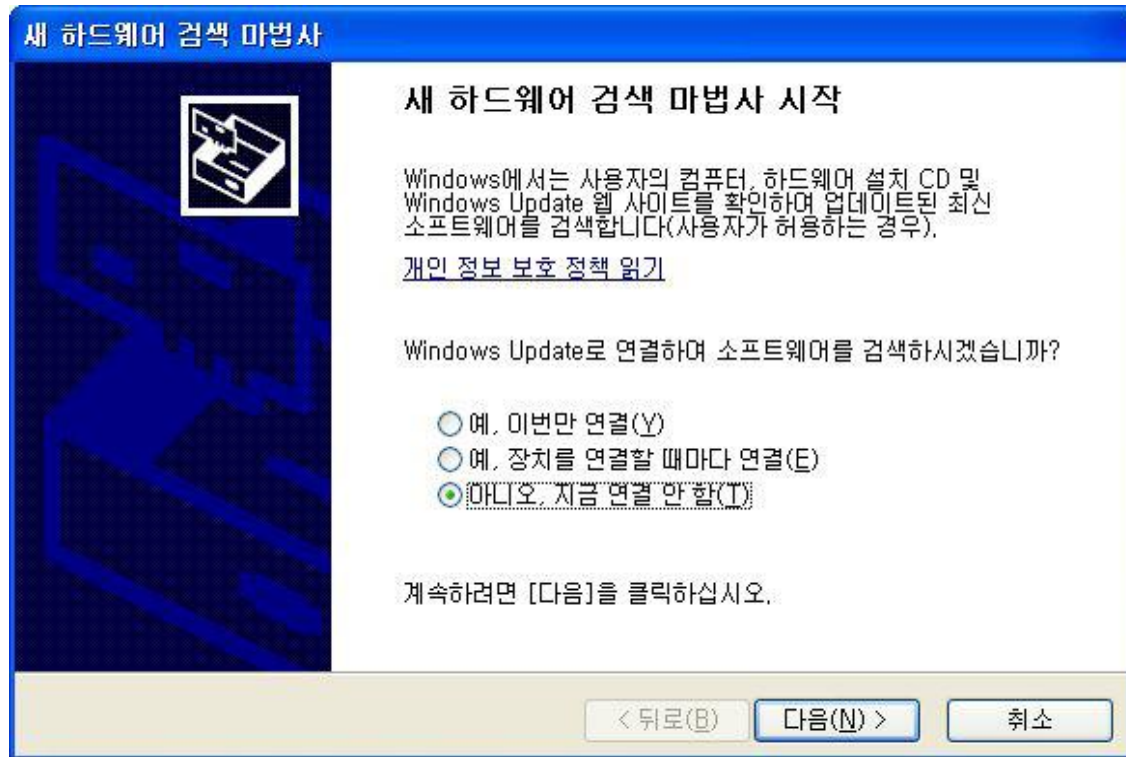
- Target Board는 USB를 통해서 JTAG 기능과 함께 USB2UART 기능을 추가로 제공한다.
- 따라서 JTAG 관련하여 디바이스 드라이버를 한 번 설치하고, USB2UART 와 관련하여 디바이스 드라이버를 한 번(한 번은 USB 드라이버, 한 번은 가상 콤포트 드라이버) 총 2번을 설치해야 한다.
 - ① 압축된 디바이스 드라이버를 제공 받아 압축을 해제한다.
 - ② Target Board를 PC 와 USB 케이블로 연결하면 다음과 같이 장치를 인식한다.



4. 개발환경 구축

■ 드라이버 설치

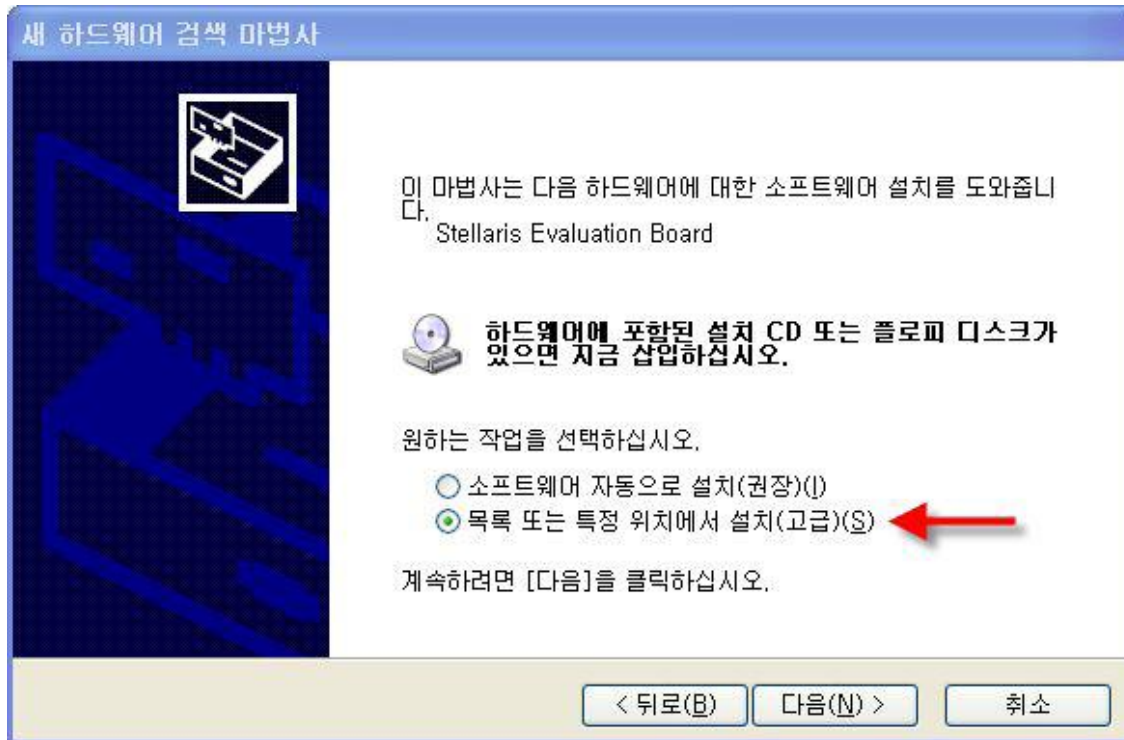
- ③ 장치가 인식되면 일반적인 하드웨어 디바이스 드라이버 설치 과정이 동일하게 아래와 그림과 같이 진행하면 된다.
- ▶ [예, 장치를 연결할 때마다 연결]을 선택한다.



4. 개발환경 구축

■ 드라이버 설치

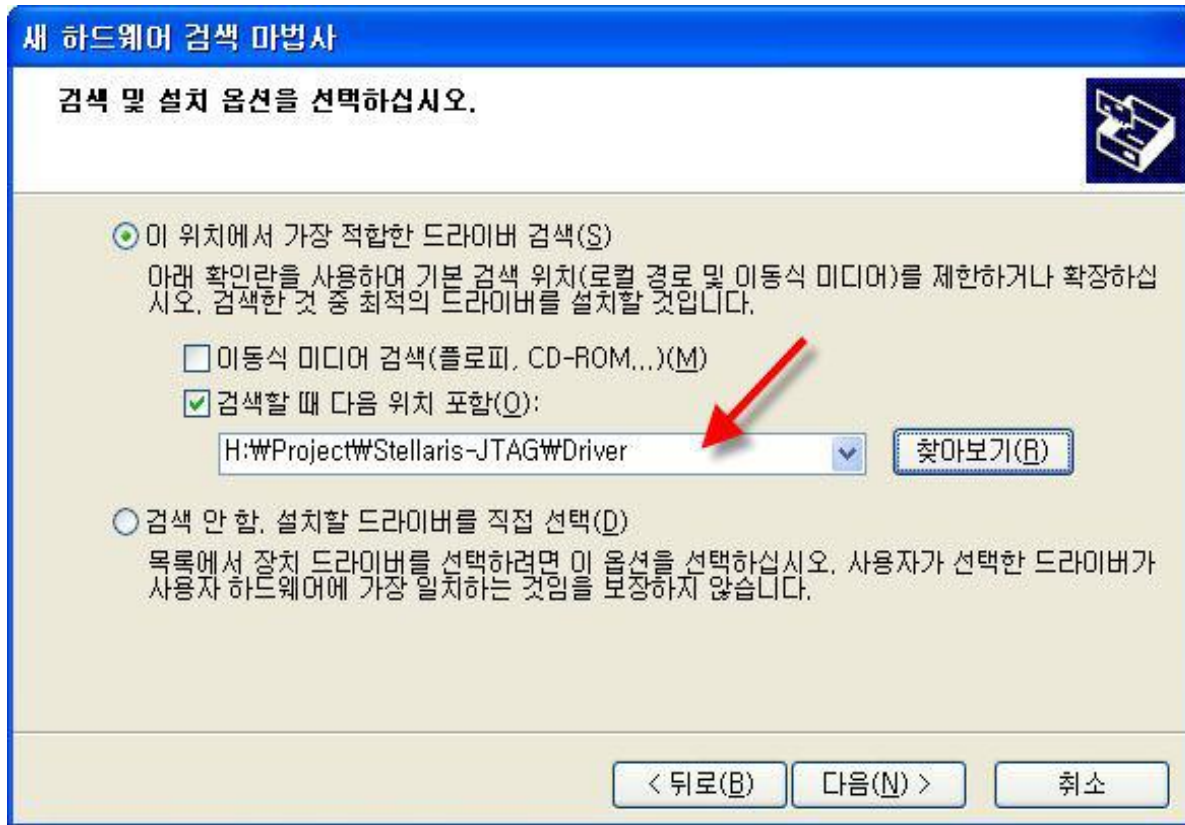
- [목록 또는 특정 위치에서 설치(고급)(S)] 선택한다.



4. 개발환경 구축

■ 드라이버 설치

- 아래 그림과 같이 옵션값을 선택하고 [찾아보기] 버튼을 클릭하여 압축이 풀려진 디바이스 드라이버 폴더를 선택한다.



4. 개발환경 구축

■ 드라이버 설치

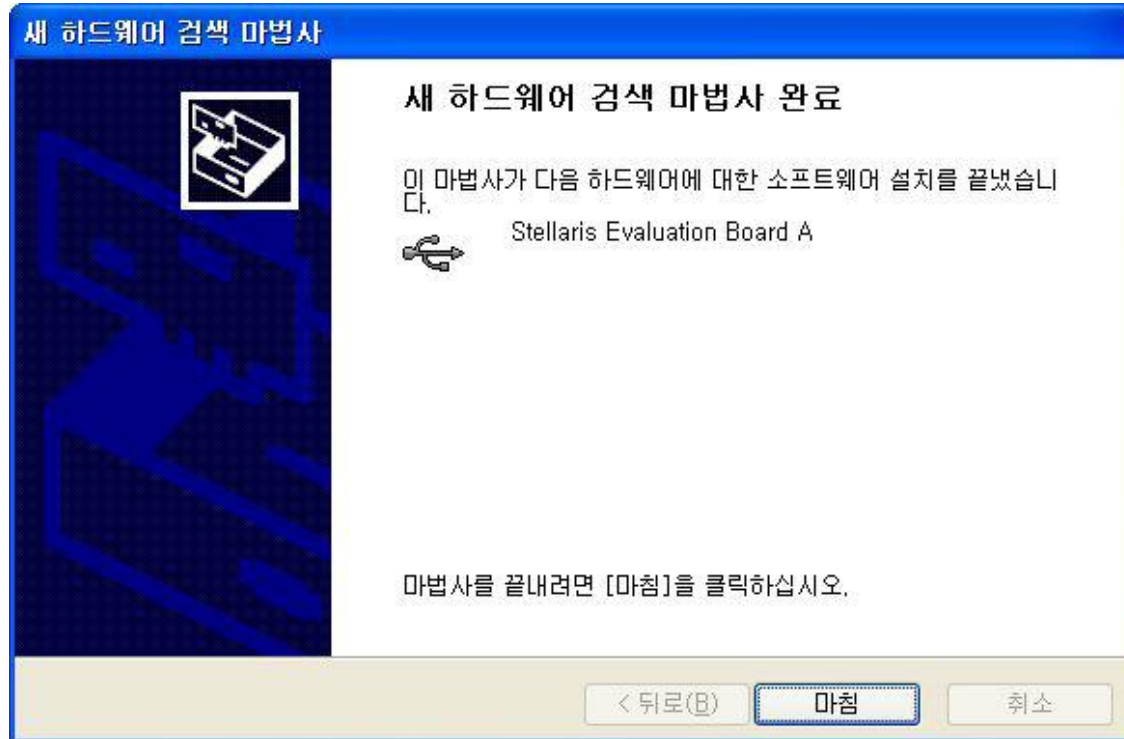
- ▶ 디바이스 드라이버를 설치한다.



4. 개발환경 구축

■ 드라이버 설치

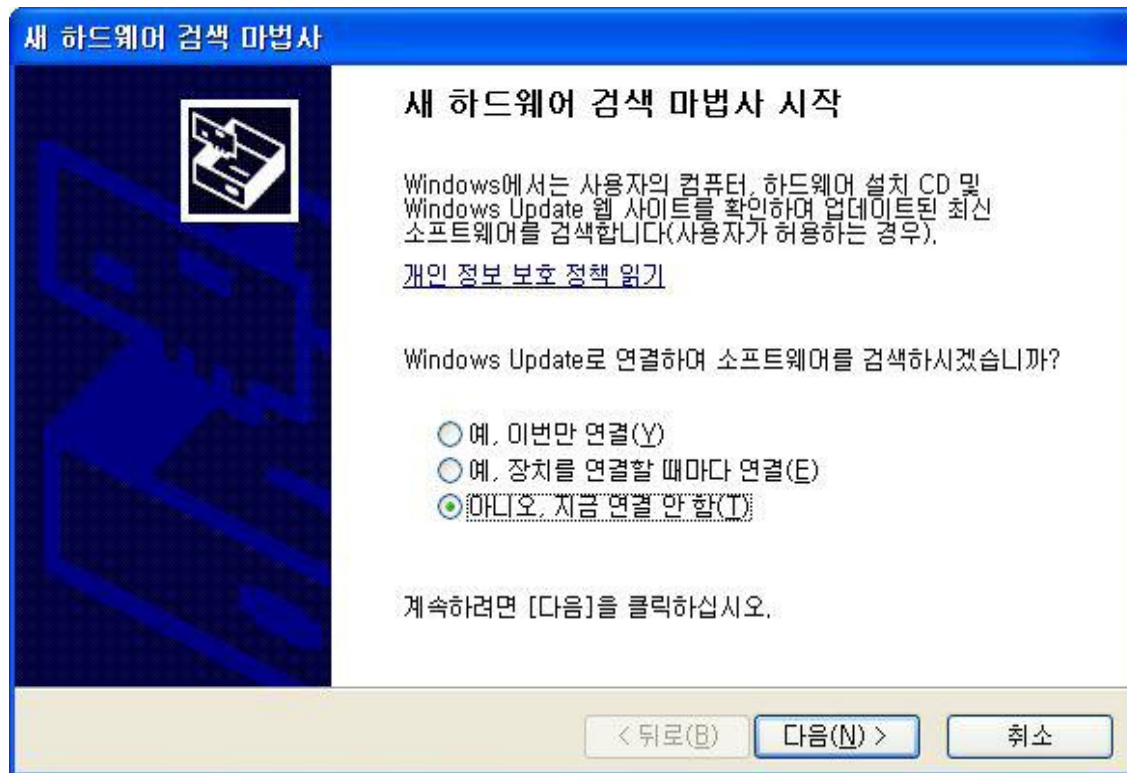
- 디바이스 드라이버가 설치가 완료 되었으면 장치 관리자에서 Stellaris Evaluation Board A가 설치되어 있음을 확인 할 수 있다.



4. 개발환경 구축

■ 드라이버 설치

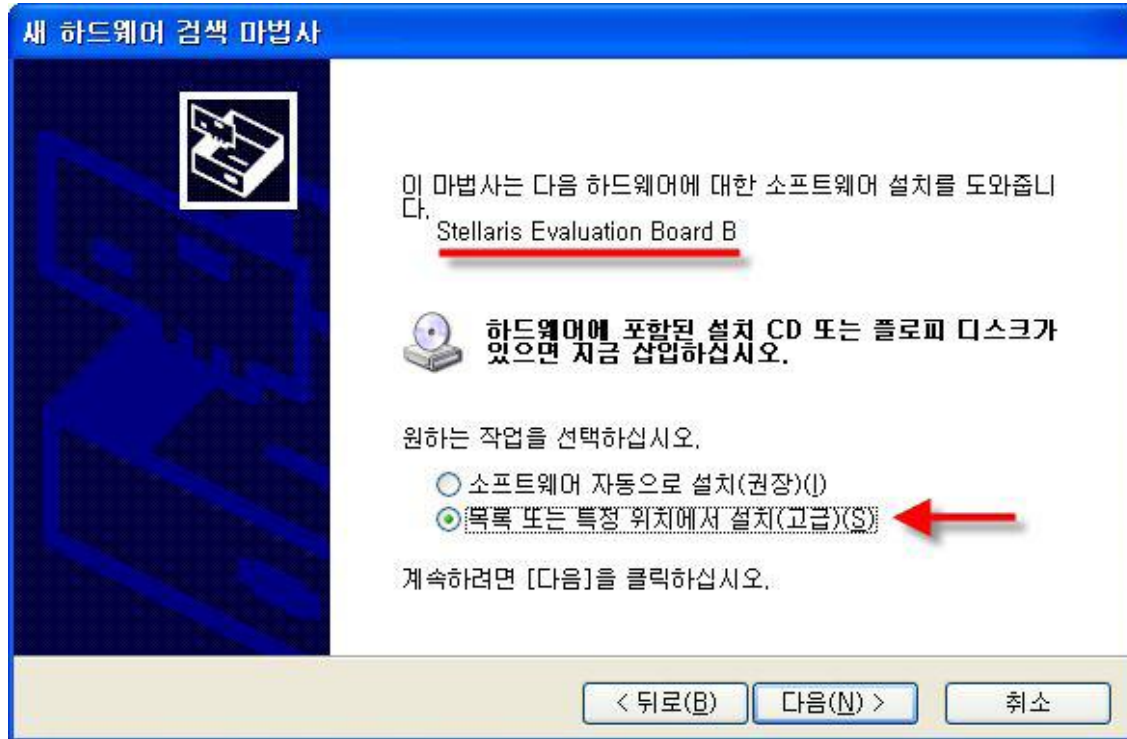
- 다음 단계는 Stellaris Evaluation Board B를 다음에 단계를 거쳐 설치한다.
- [예, 장치를 연결할 때마다 연결]을 선택한다.



4. 개발환경 구축

■ 드라이버 설치

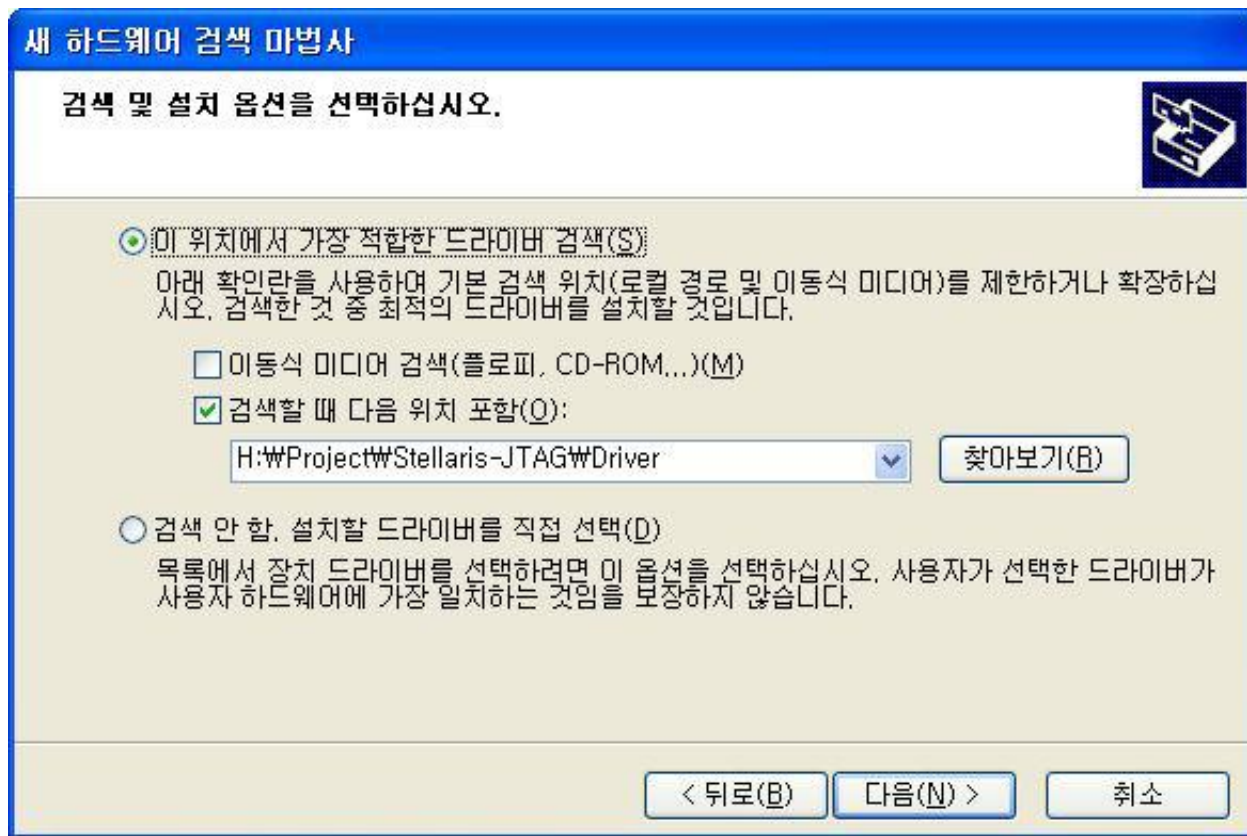
- [목록 또는 특정 위치에서 설치(고급)(S)] 선택한다.



4. 개발환경 구축

■ 드라이버 설치

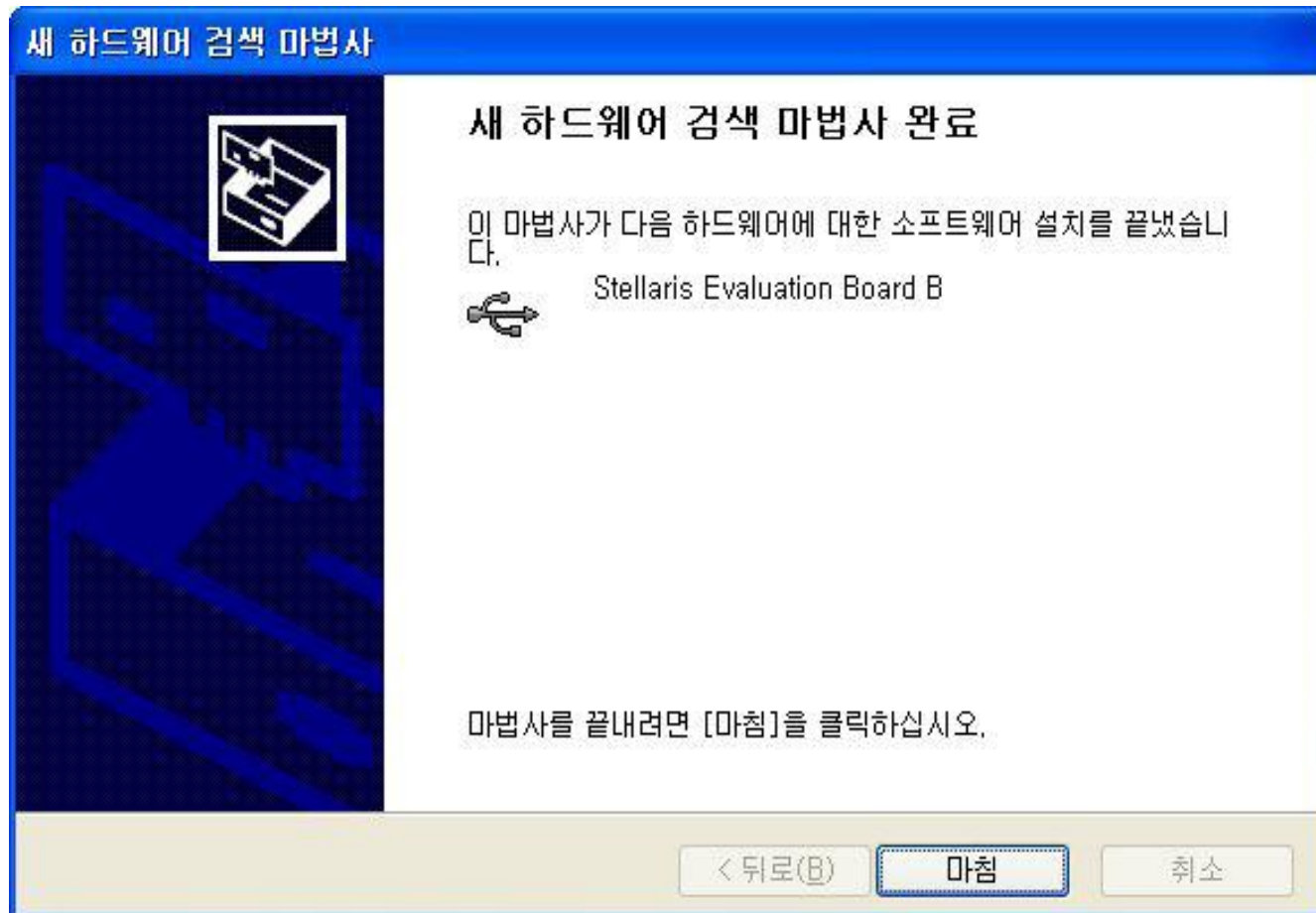
- ▶ 아래 그림과 같이 옵션값을 선택하고 [찾아보기] 버튼을 클릭하여 압축이 풀려진 디바이스 드라이버 폴더를 선택한다.



4. 개발환경 구축

■ 드라이버 설치

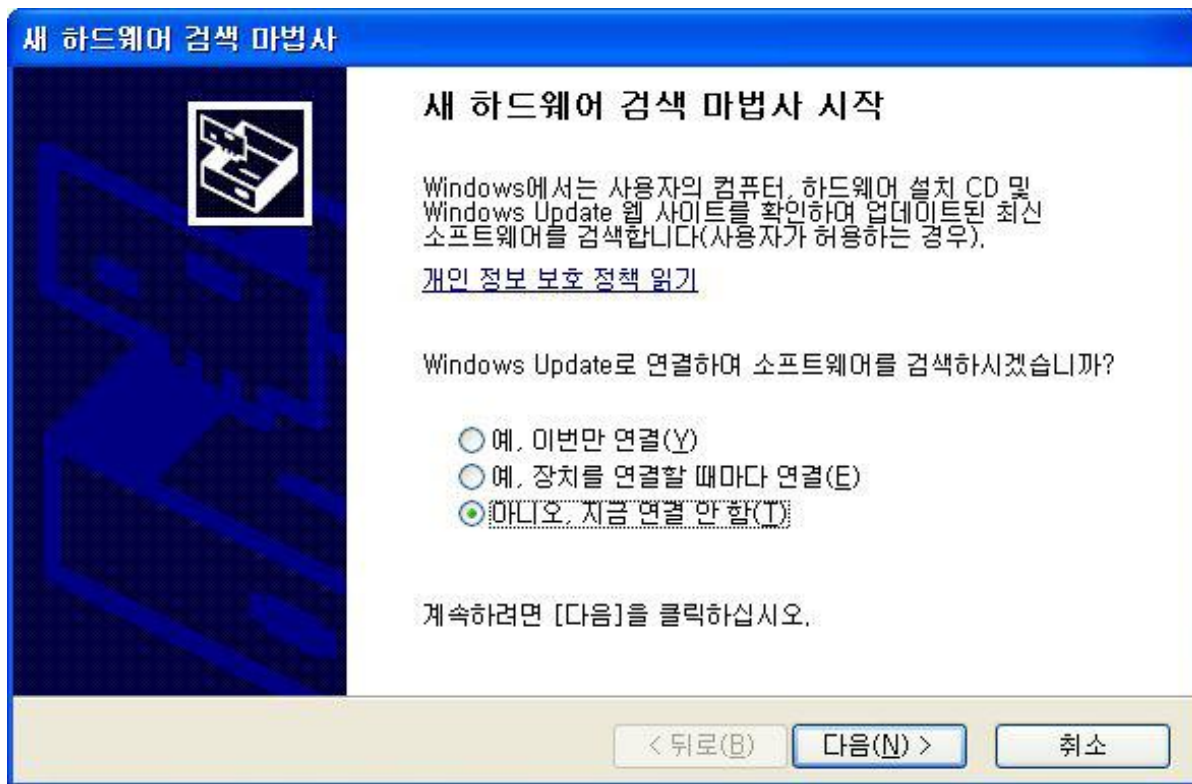
- ▶ 디바이스 드라이버가 설치가 완료 되었으면 장치 관리자에서 Stellaris Evaluation Board B가 설치 되어 있음을 확인 할 수 있다.



4. 개발환경 구축

■ 드라이버 설치

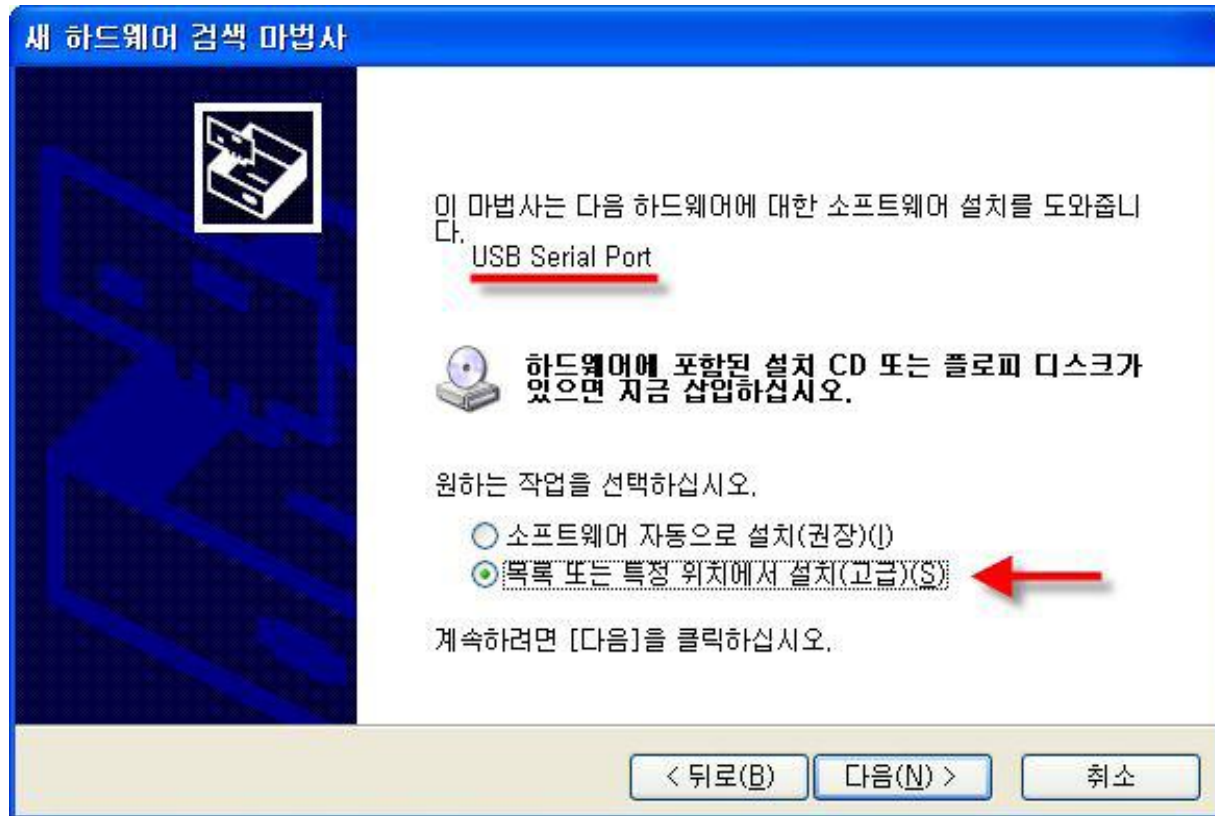
- ④ 계속해서 한 번 더 디바이스 드라이버가 설치해야 한다. 이 디바이스 드라이버는 가상의 USB 생성하기 위한 드라이버를 요구하는 것으로 이 역시 압축을 푼 같은 폴더에 필요한 파일이 들어있다.
- ► 다음 단계는 USB Serial를 다음에 단계를 거쳐 설치한다.
- [예, 장치를 연결할 때마다 연결]을 선택한다.



4. 개발환경 구축

■ 드라이버 설치

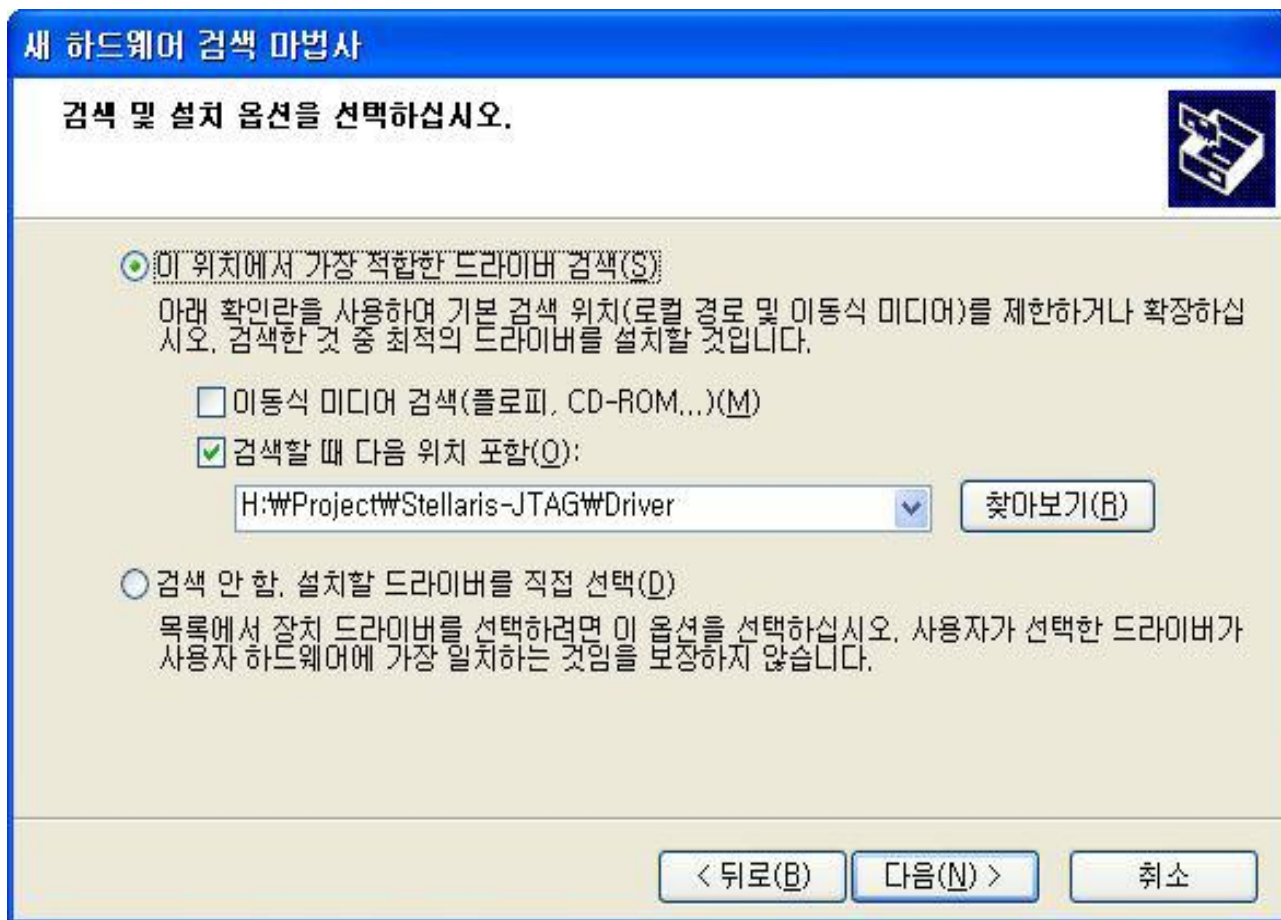
- ▶ [목록 또는 특정 위치에서 설치(고급)(S)] 선택한다.



4. 개발환경 구축

■ 드라이버 설치

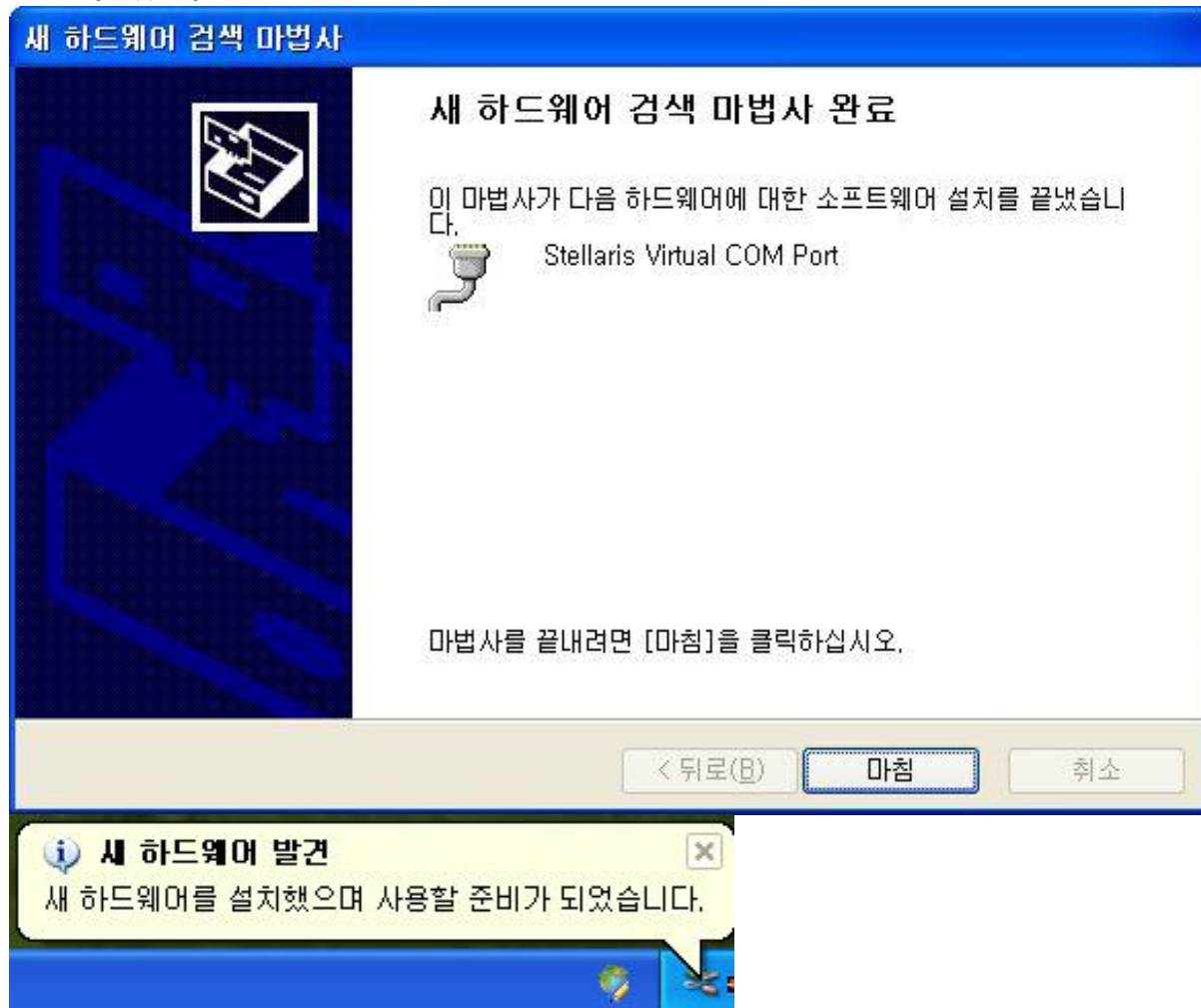
- ▶ 아래 그림과 같이 옵션값을 선택하고 [찾아보기] 버튼을 클릭하여 압축이 풀려진 디바이스 드라이버 폴더를 선택한다.



4. 개발환경 구축

■ 드라이버 설치

- ▶ 디바이스 드라이버가 설치가 완료 되었으면 장치 관리자에서 USB Serial가 설치되어 있음을 확인 할 수 있다.



4. 개발환경 구축

■ 드라이버 설치

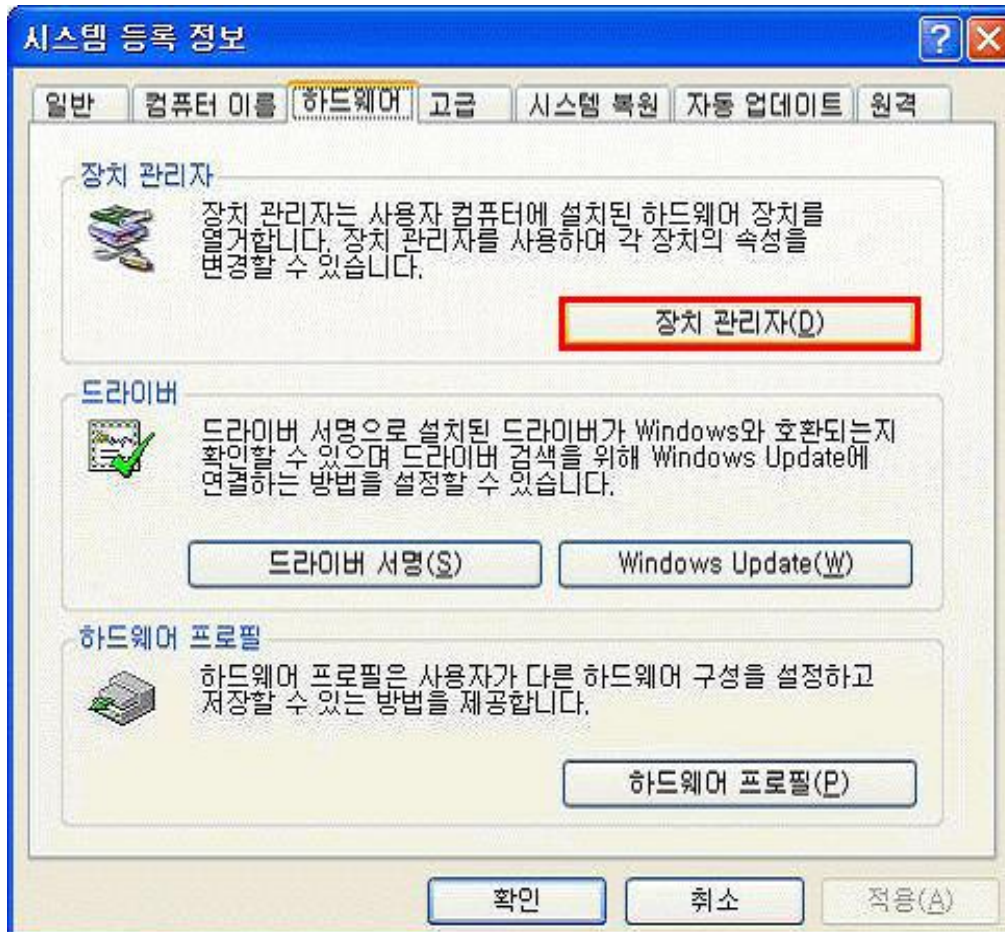
- ⑤ 해당되는 디바이스 드라이버가 잘 설치 되었는지 설치 확인을 위해 제어판->시스템에서 하드웨어 탭을 클릭한다



4. 개발환경 구축

■ 드라이버 설치

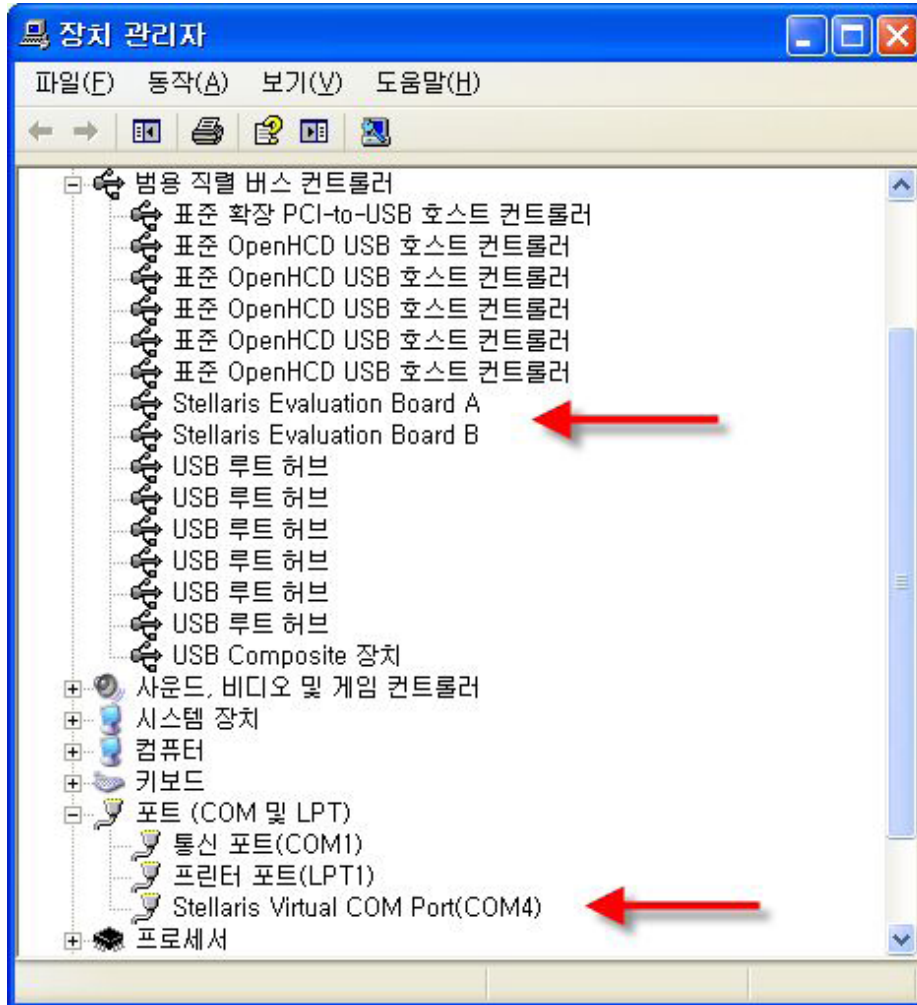
- ▶ 장치 관리자를 클릭한다.



4. 개발환경 구축

■ 드라이버 설치

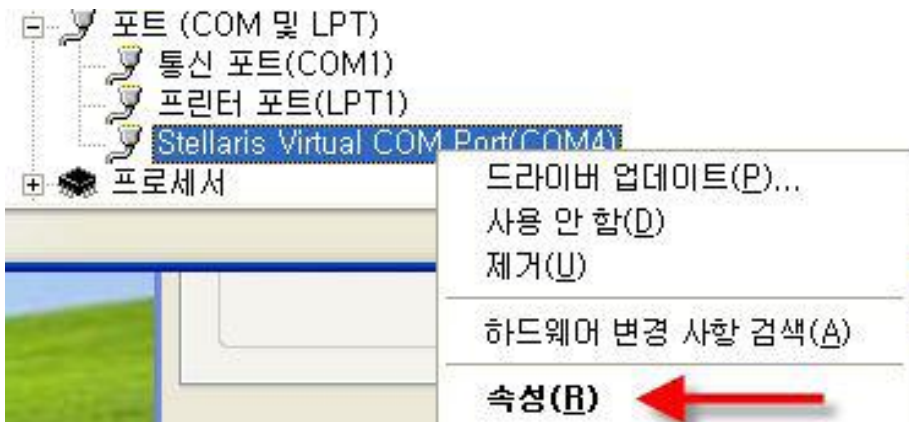
- ▶ 아래 그림과 같이 3개의 장치가 설치되었는지 확인 해 본다.



4. 개발환경 구축

■ COM Port 변경 하기

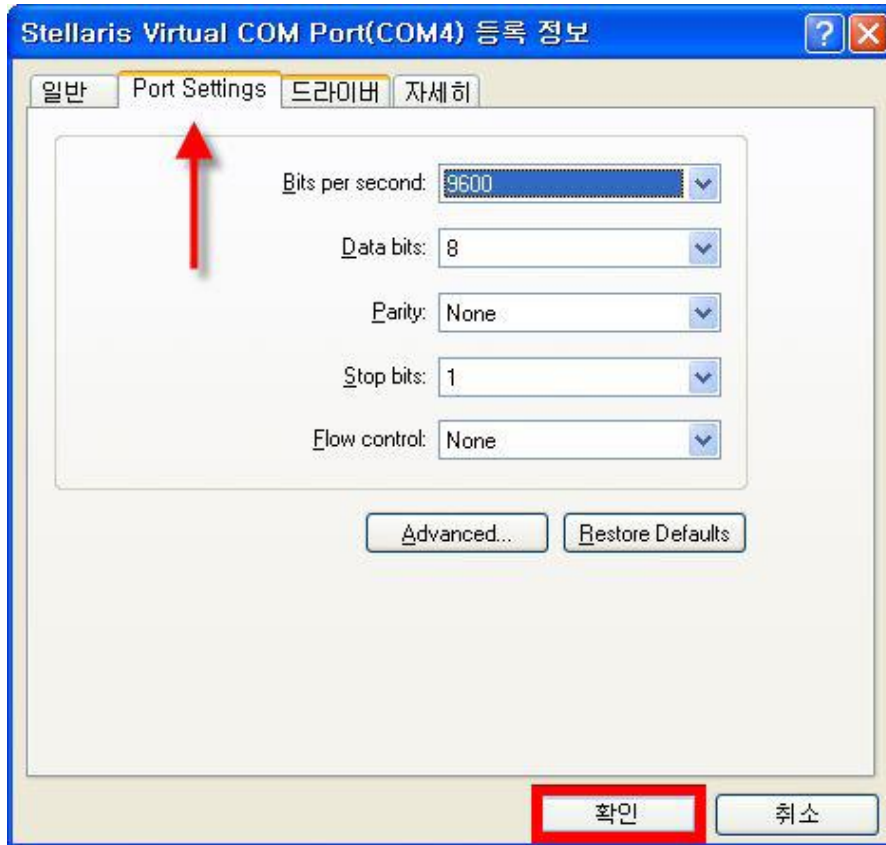
- 가상 USB Serial Port를 자기가 사용하고자 하는 COM 포트 번호로 변경시켜 사용하는 것이 가능하다 .
- ① 장치 관리자의 USB Serial Port 항목에서 마우스 오른쪽 버튼을 클릭하여 보조 메뉴 중에서 [속성]을 선택합니다.



4. 개발환경 구축

■ COM Port 변경 하기

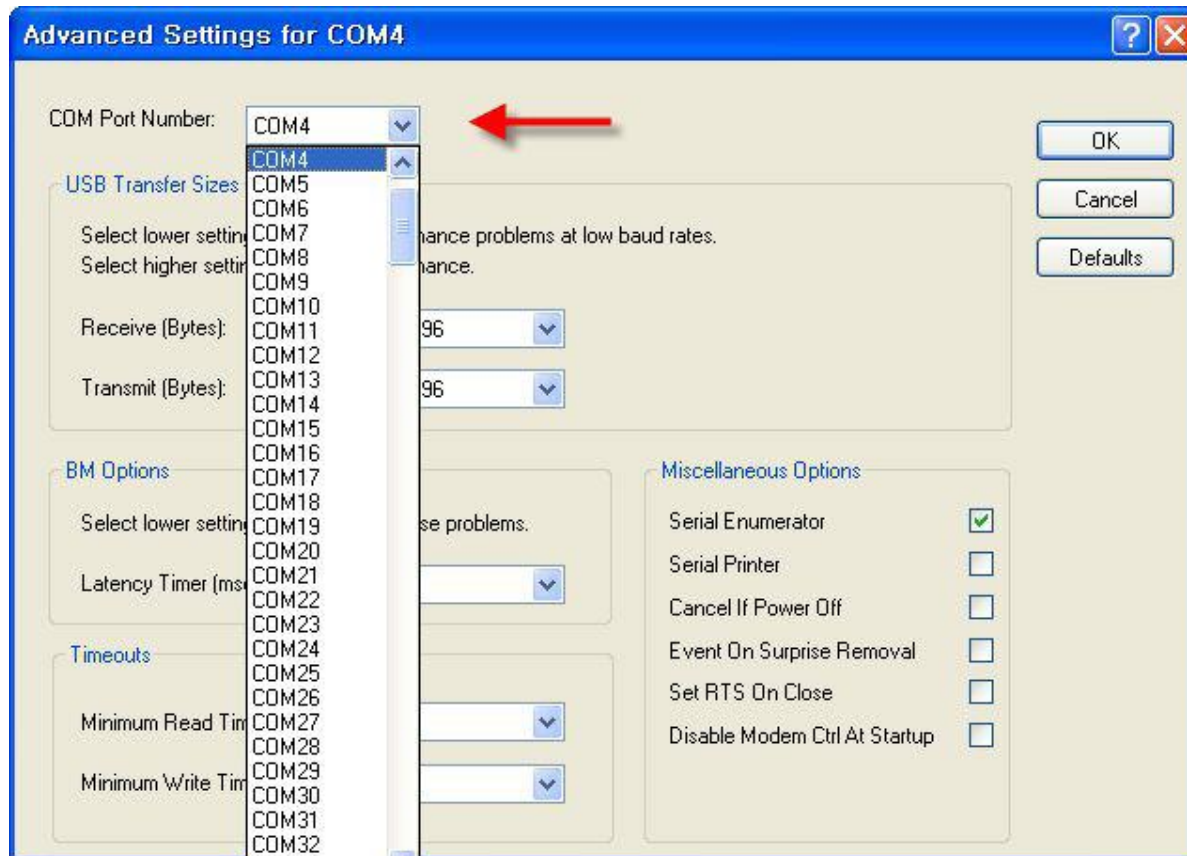
- ② 아래 그림과 같은 Stellaris Virtual Com Port 다이얼로그 창이 뜹니다. “Port Settings” 탭을 클릭한 후 [Advanced...] 버튼을 클릭합니다.



4. 개발환경 구축

■ COM Port 변경 하기

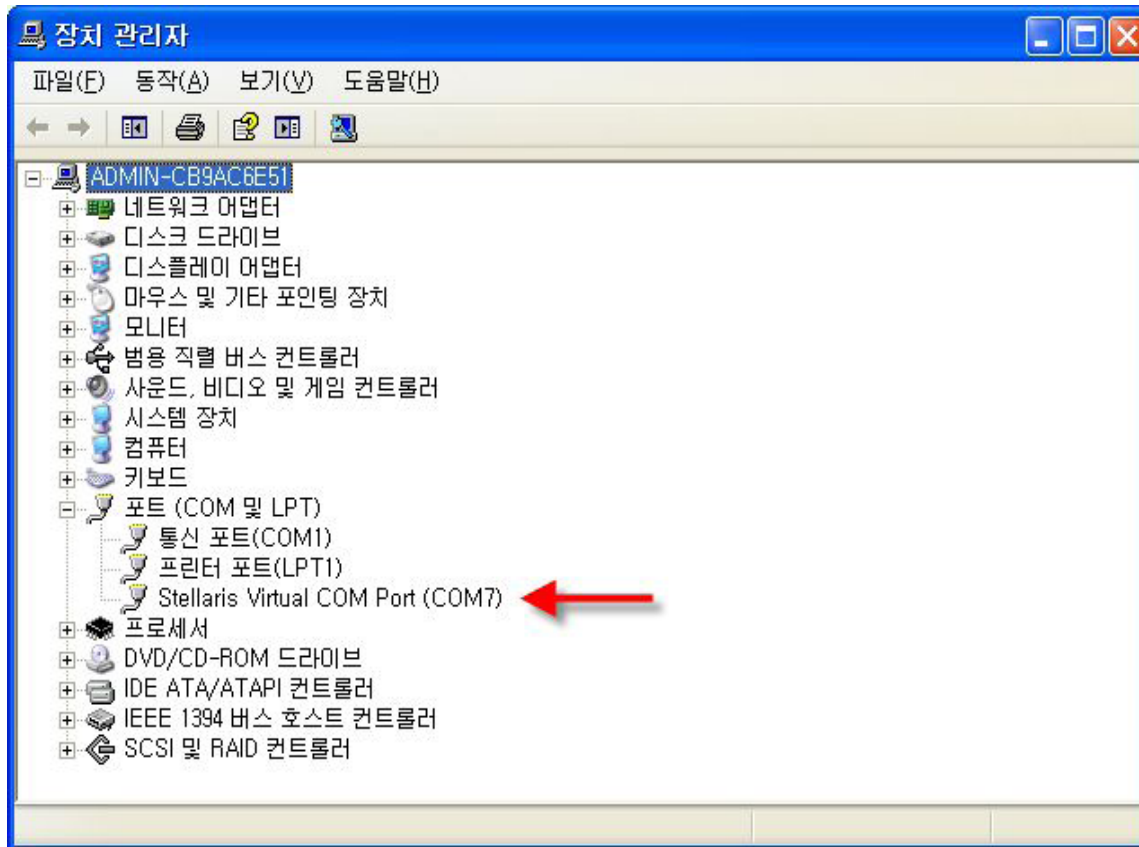
- ③ 설치한 가상 포트에 대해 좀 더 자세한 설정을 할 수 있는 고급 설정 다이얼로그 창이 뜹니다. 이 중 상단에 있는 COM Port Number 콤보 박스를 열면 변경할 수 있는 COM 포트 번호가 출력됩니다. 원하는 번호를 선택하시면 됩니다. 설명서에서는 COM 7 로 변경해 보겠습니다.



4. 개발환경 구축

■ COM Port 변경 하기

- ④ 변경한 후 장치 관리자를 다시 열어 확인해 보면, COM4 가 COM7 로 바뀐 것을 확인할 수 있습니다. 이 기능을 이용하면 사용자가 원하는 COM 포트로 변경 후 사용하실 수 있습니다.

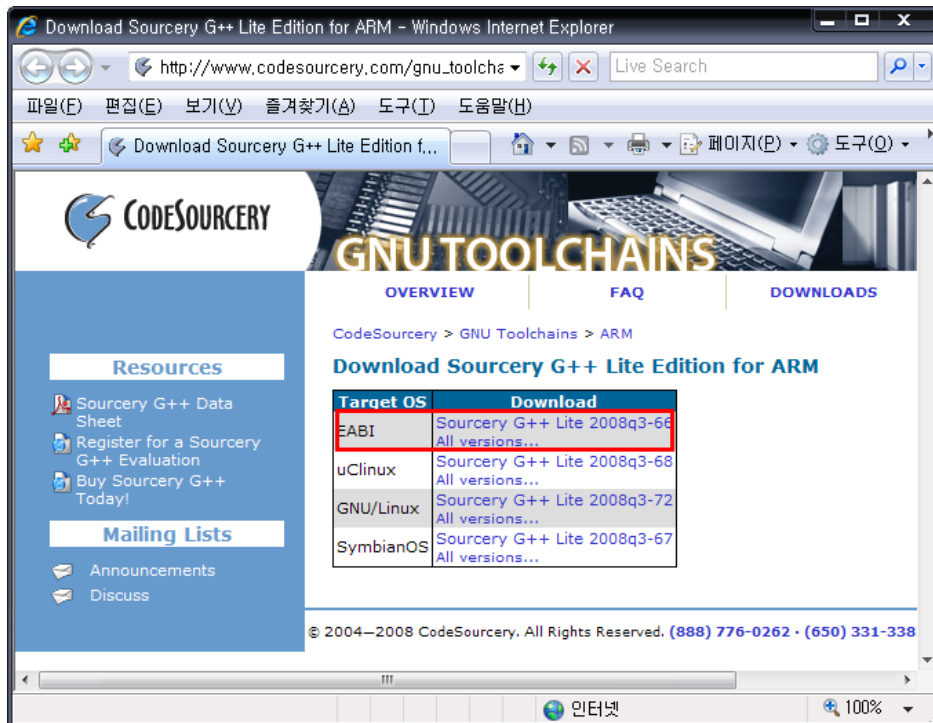


4. 개발환경 구축

■ 개발환경 설치

■ CodeSourcery G++ lite 설치

- 일단 GCC 크로스 컴파일러를 설치하여보자.
- GCC는 여러 플랫폼에 맞는 다양한 버전이 있으며, Code Sourcery 에서 배포하는 CodeSourcery G++ lite 를 사용하여 보자.
- 아래 주소에서 GCC 설치 파일을 다운로드 해서 설치할 수 있다.
- http://www.codesourcery.com/gnu_toolchains/arm/download.html

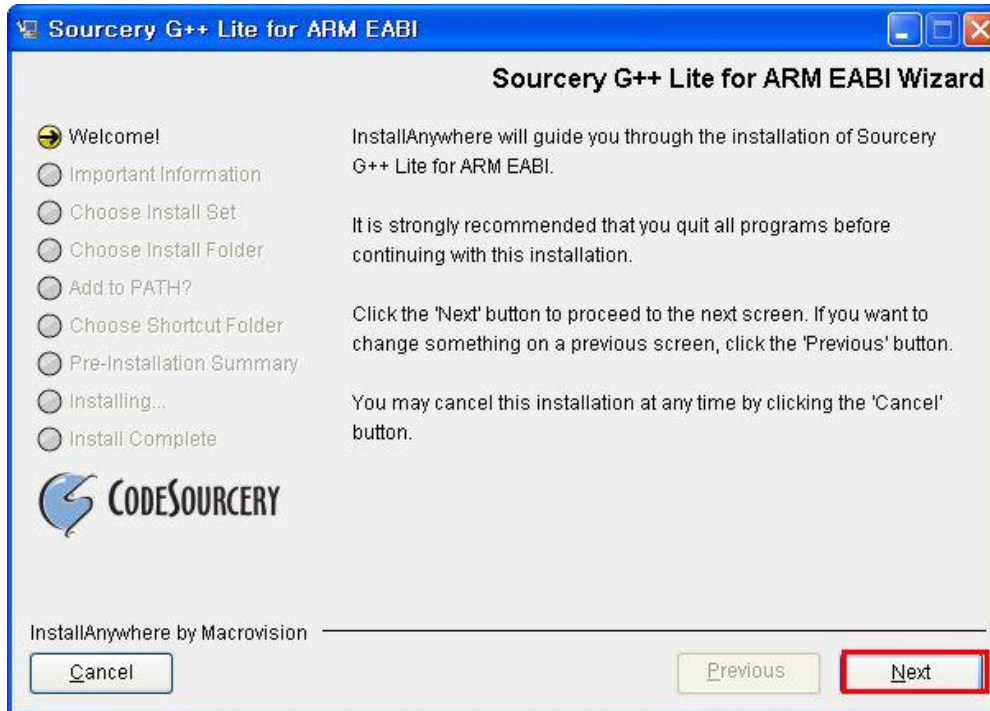


4. 개발환경 구축

■ 개발환경 설치

■ CodeSourcery G++ lite 설치

- 설치시에는 아래와 같은 환경설정을 적용하여 설치하자.



4. 개발환경 구축

■ 개발환경 설치

■ CodeSourcery G++ lite 설치

- 설치시에는 아래와 같은 환경설정을 적용하여 설치하자.



4. 개발환경 구축

■ 개발환경 설치

■ CodeSourcery G++ lite 설치

- 설치시에는 아래와 같은 환경설정을 적용하여 설치하자.

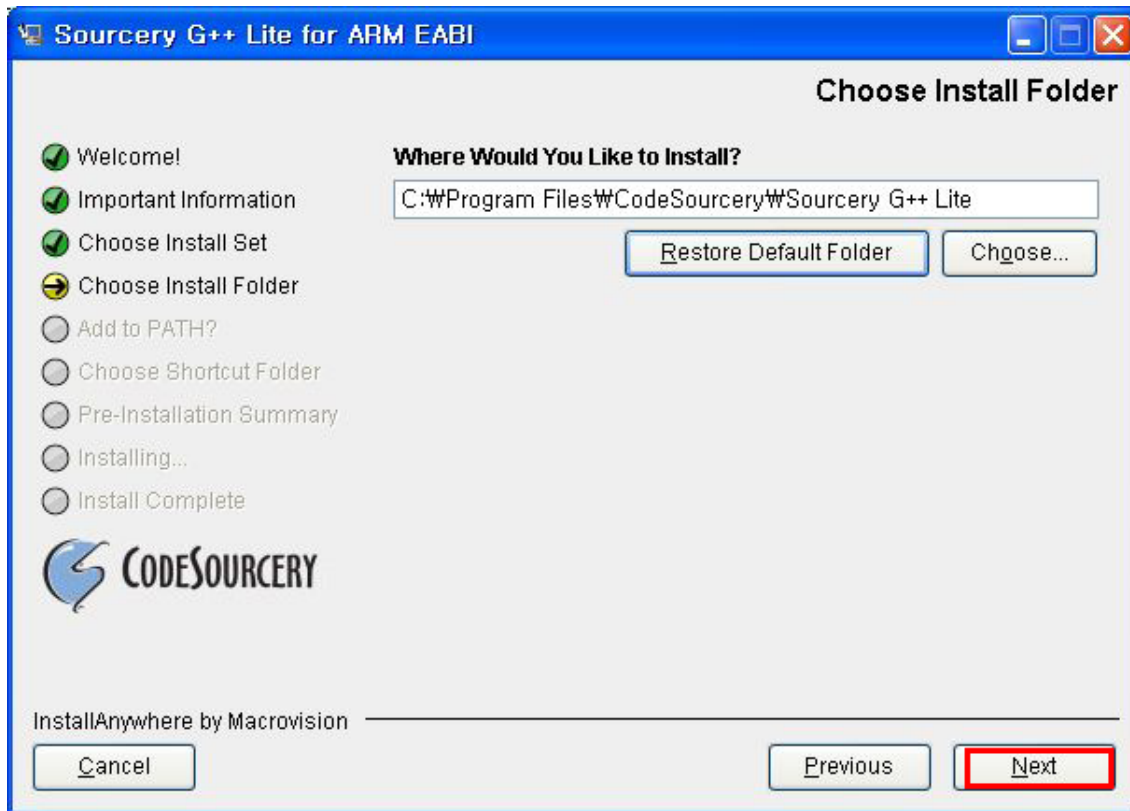


4. 개발환경 구축

■ 개발환경 설치

■ CodeSourcery G++ lite 설치

- 설치시에는 아래와 같은 환경설정을 적용하여 설치하자.

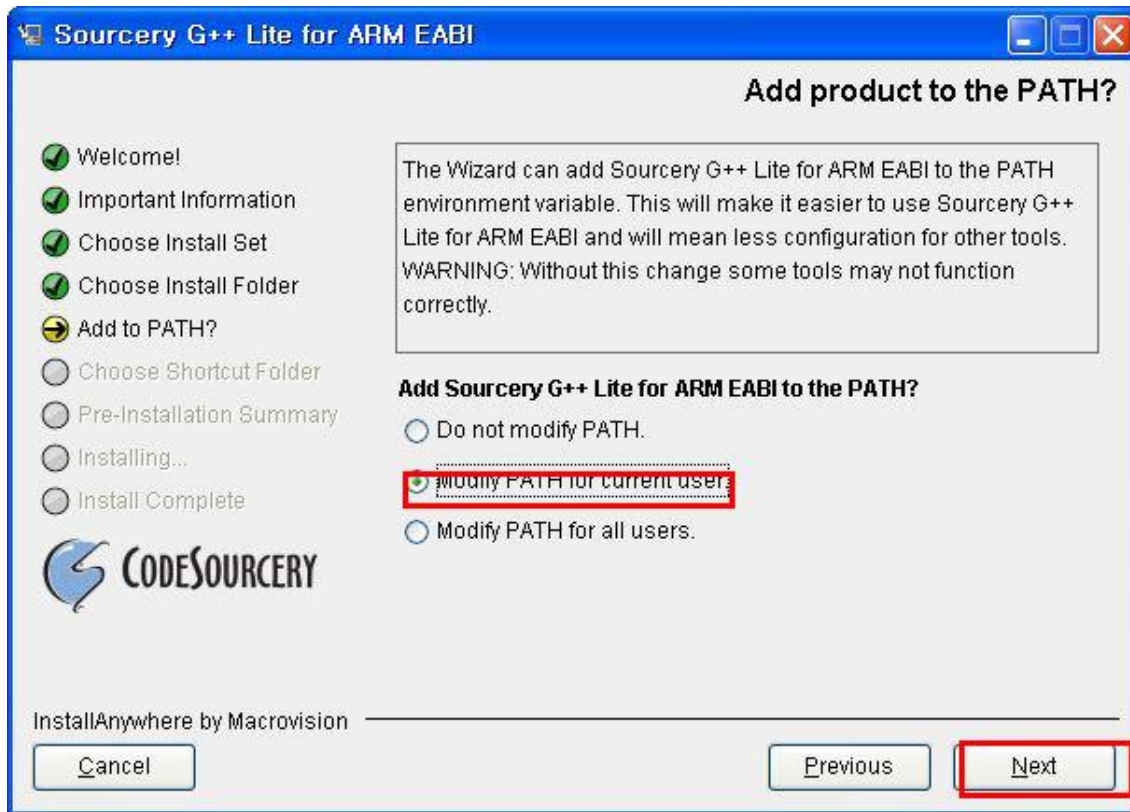


4. 개발환경 구축

■ 개발환경 설치

■ CodeSourcery G++ lite 설치

- 설치시에는 아래와 같은 환경설정을 적용하여 설치하자.

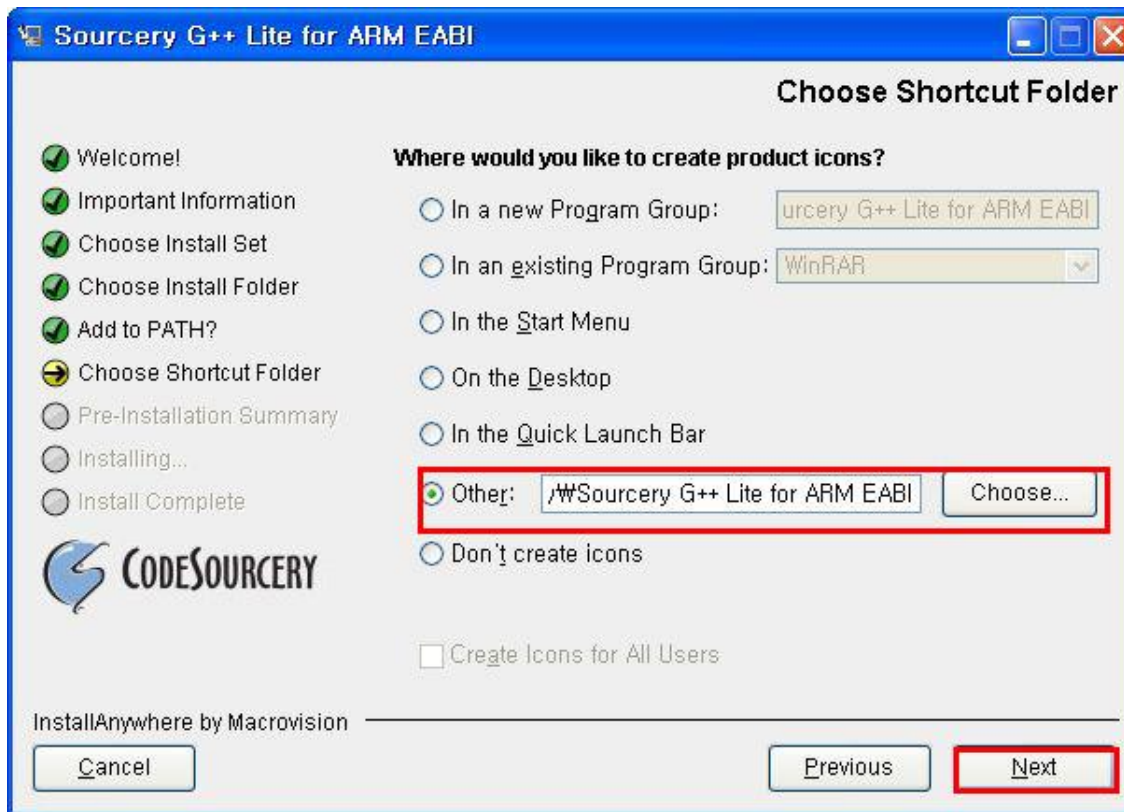


4. 개발환경 구축

■ 개발환경 설치

■ CodeSourcery G++ lite 설치

- 설치시에는 아래와 같은 환경설정을 적용하여 설치하자.



4. 개발환경 구축

■ 개발환경 설치

■ CodeSourcery G++ lite 설치

- 위 과정을 마치면 자동으로 해당폴더에 CodeSourcery G++ lite 설치 되고 시스템 환경변수에 설치가 된 PATH 경로가 추가된다.
- 또한 추가된 PATH의 환경변수 값을 가지고 컴파일러와 링커가 접근하여 사용하게 된다.

4. 개발환경 구축

■ 개발환경 설치

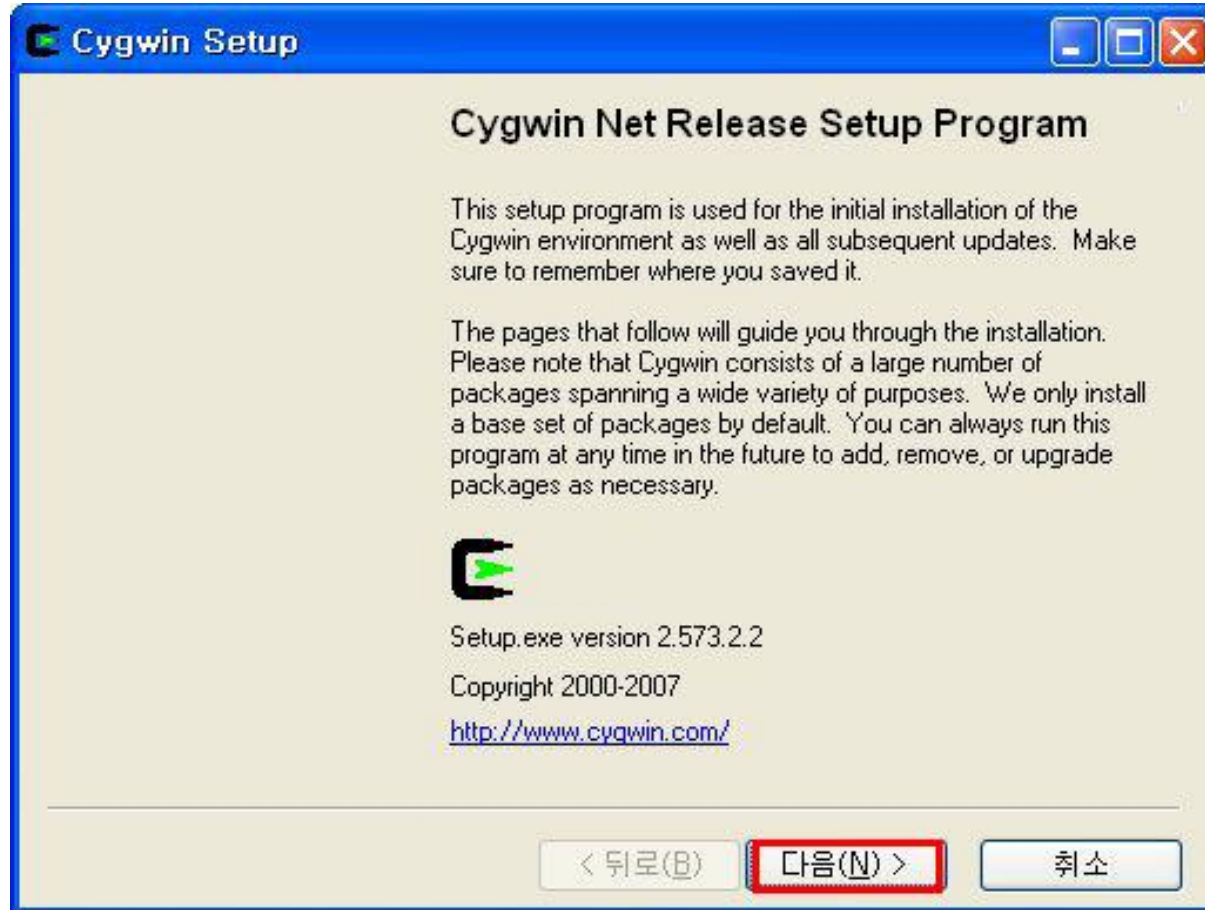
■ Cygwin 설치

- Cygwin의 인스톨 파일은 용량이 아주 적다 이것은 설치에 필요한 파일을 가지고 있는 것이 아니라 사용자가 선택한 옵션에 따라 필요한 파일들만 인터넷에서 실시간으로 가져와서 설치하는 방법을 사용하고 있기 때문에 인스톨러 자체는 수백 킬로바이트 정도의 크기입니다.
- 아래 주소에서 실행 파일을 다운로드 받아 설치 하여 보자.
- <http://www.cygwin.com/setup.exe>
- 설치시 아래의 그림을 따라 설정하여 설치하여도 되고 Full Package 로 설치하여도 된다.

4. 개발환경 구축

■ 개발환경 설치

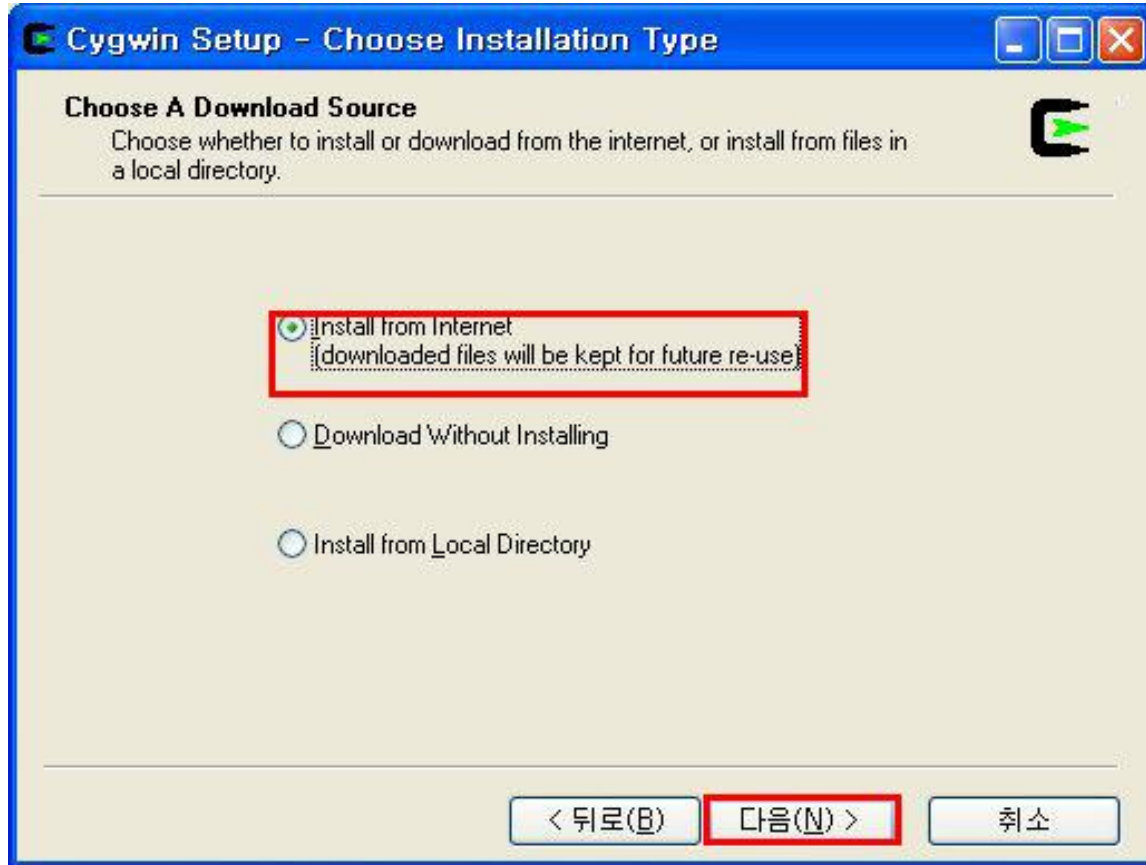
▣ Cygwin 설치



4. 개발환경 구축

■ 개발환경 설치

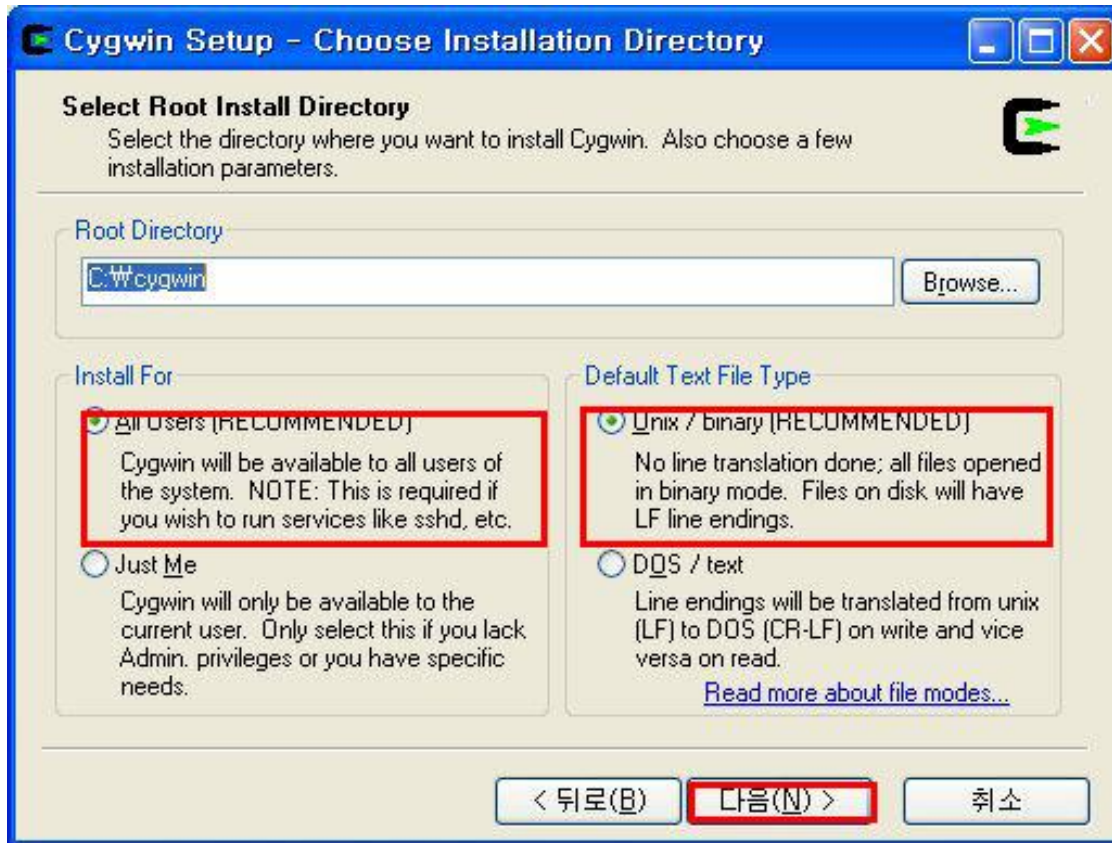
▣ Cygwin 설치



4. 개발환경 구축

■ 개발환경 설치

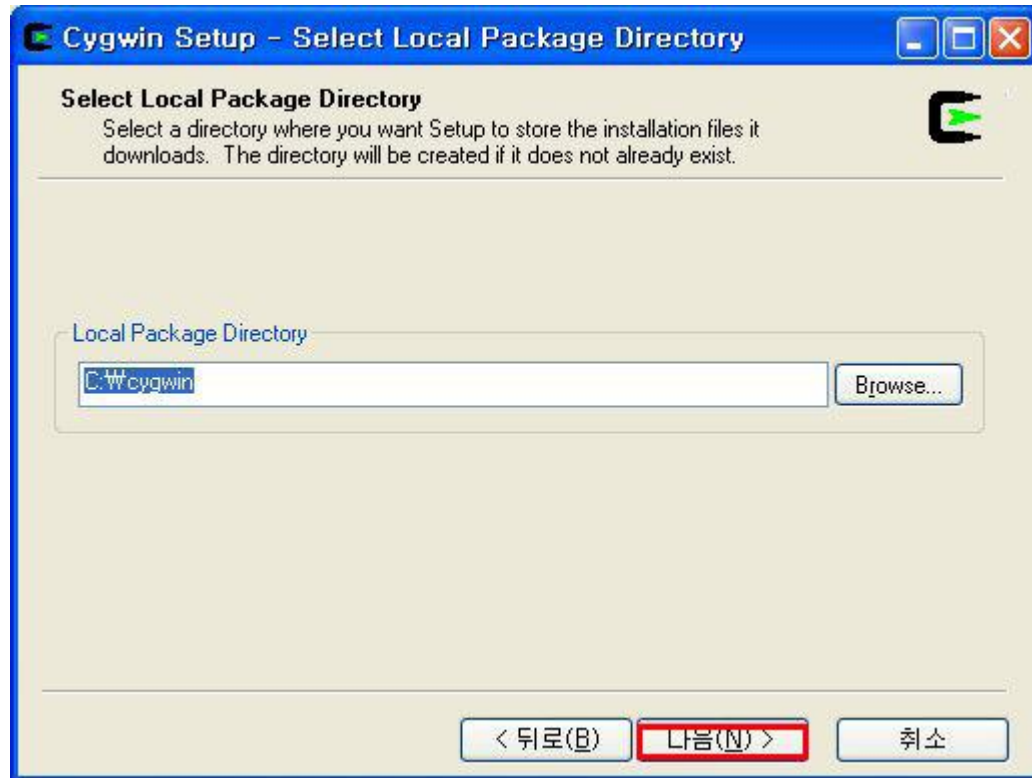
■ Cygwin 설치



4. 개발환경 구축

■ 개발환경 설치

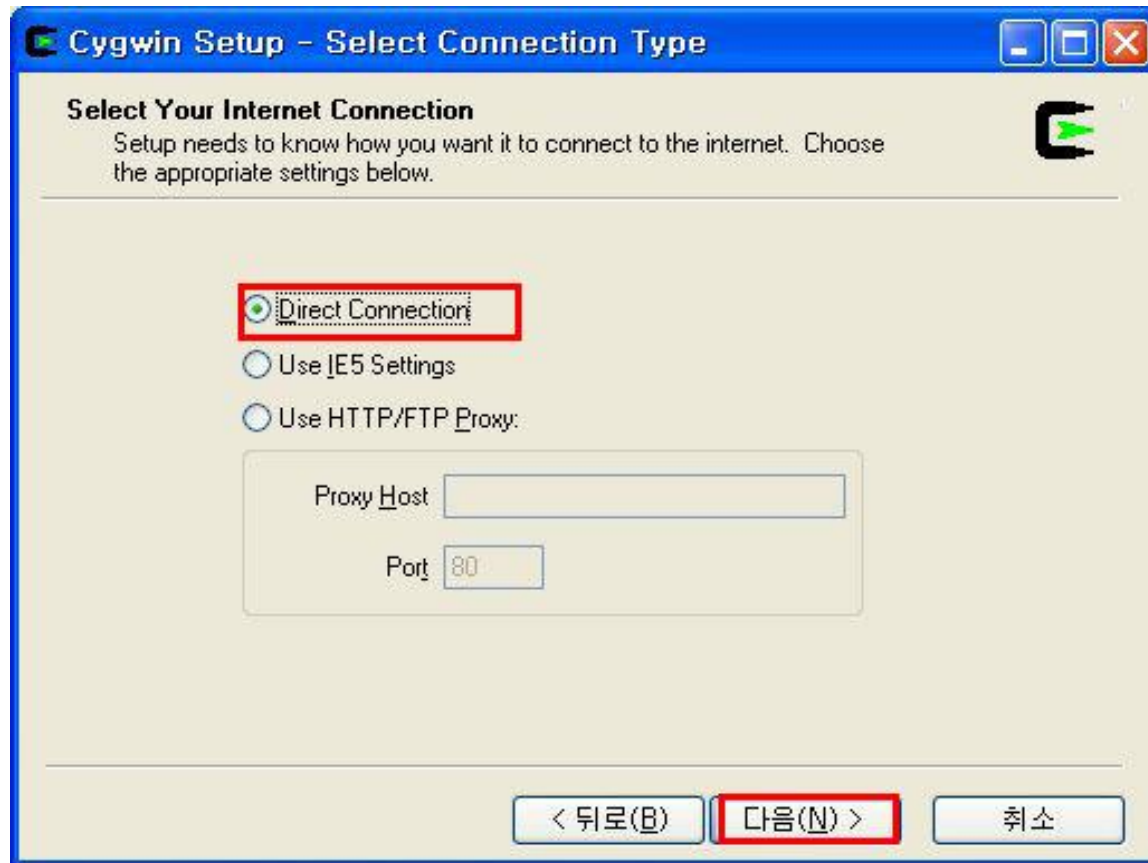
▪ Cygwin 설치



4. 개발환경 구축

■ 개발환경 설치

▣ Cygwin 설치

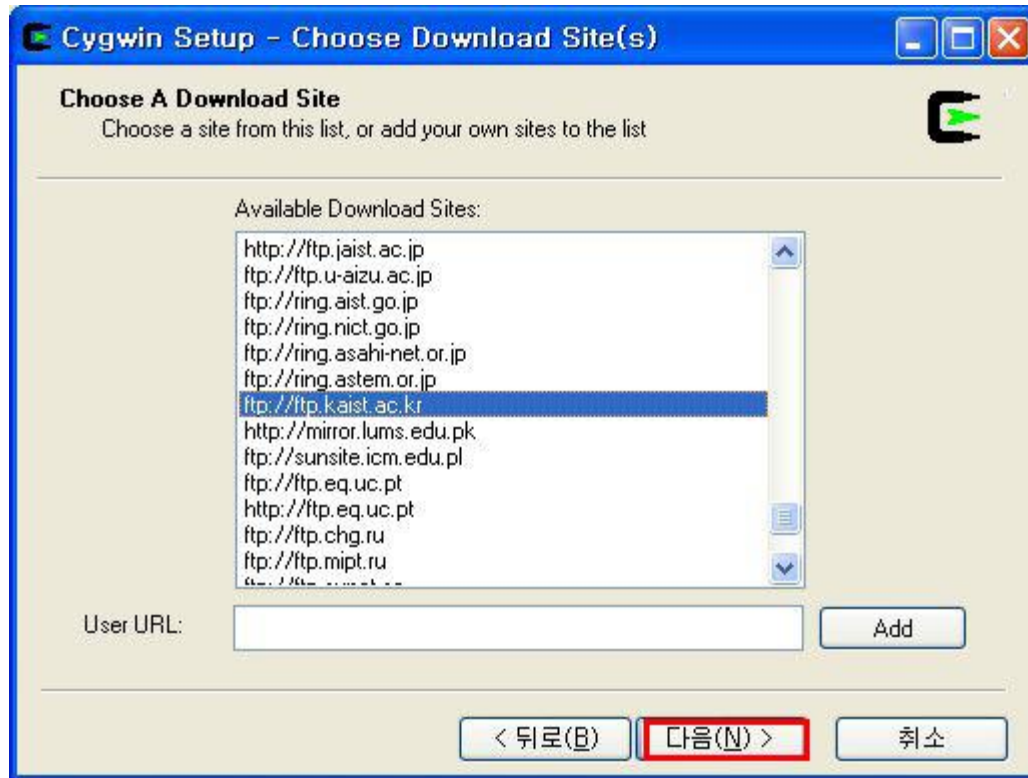


4. 개발환경 구축

■ 개발환경 설치

■ Cygwin 설치

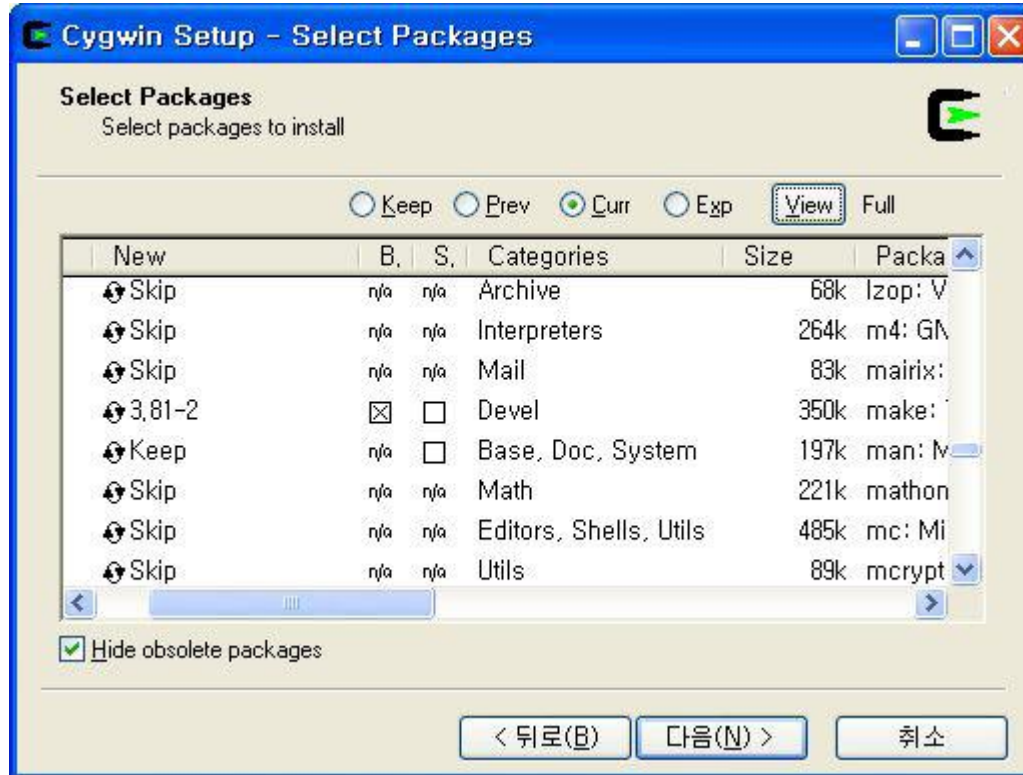
- 아래의 그림의 단계에서는 국내 서버를 선택하는 것이 조금이라도 빨리 설치가 되는 장점이 있으므로 국내 서버를 선택하자.



4. 개발환경 구축

■ 개발환경 설치

■ Cygwin 설치



4. 개발환경 구축

■ 개발환경 설치

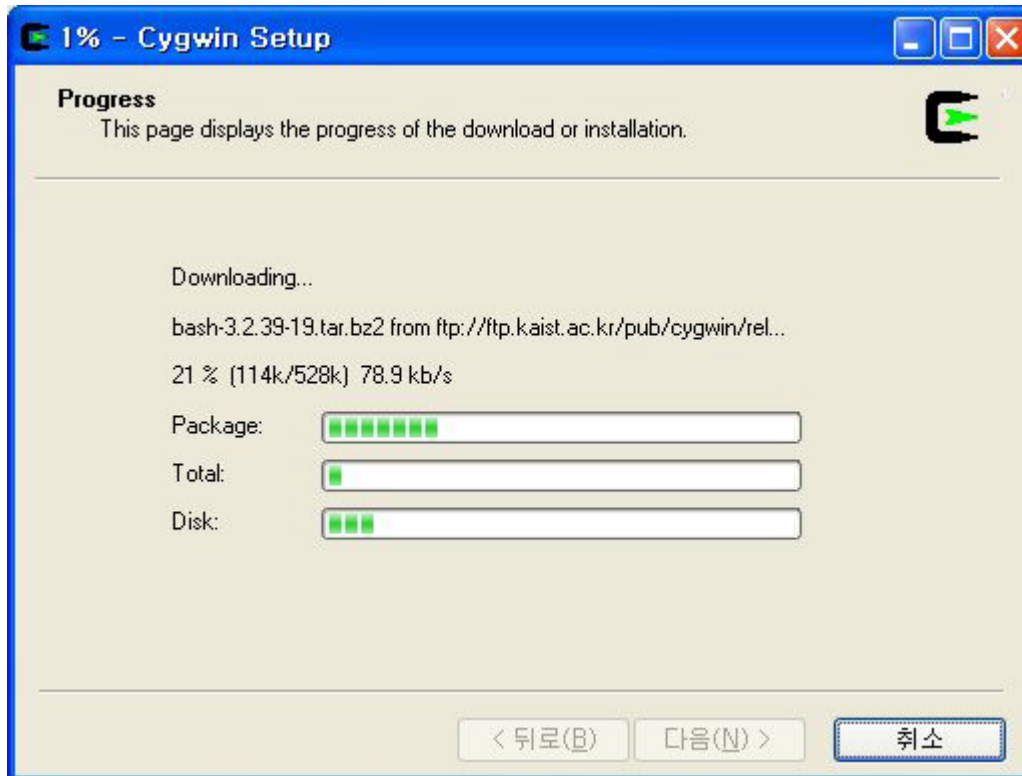
■ Cygwin 설치

- Select Packages 단계에서 화면 우측 상단의 "View" 버튼을 한번 클릭하면 "View" 버튼 옆의 "Category" 문구가 "Full"로 변경됩니다.
- 이때 화면 중앙 리스트의 "Package" 항목에서 "make: The GNU version of the 'make' utility"를 찾아 "New" 항목의 "Skip"을 마우스로 클릭하면 "Skip" 이 "3.81-2"와 같은 버전 이름으로 바뀌게 됩니다.
- 이 버전은 make 유틸리티의 버전이며, 바뀔 수 있습니다.
- 마찬가지로 방법으로 "Package" 중 "perl: Larry Wall's Practical extracting and Report Language" 를 찾아 "New" 항목의 "Skip"을 마우스로 한번 클릭하면 "5.8.8-4"와 같은 문자로 바뀌게 됩니다.
- 이제 다음 단계로 진행하면 됩니다.

4. 개발환경 구축

■ 개발환경 설치

■ Cygwin 설치



4. 개발환경 구축

■ 개발환경 설치

■ Cygwin 설치

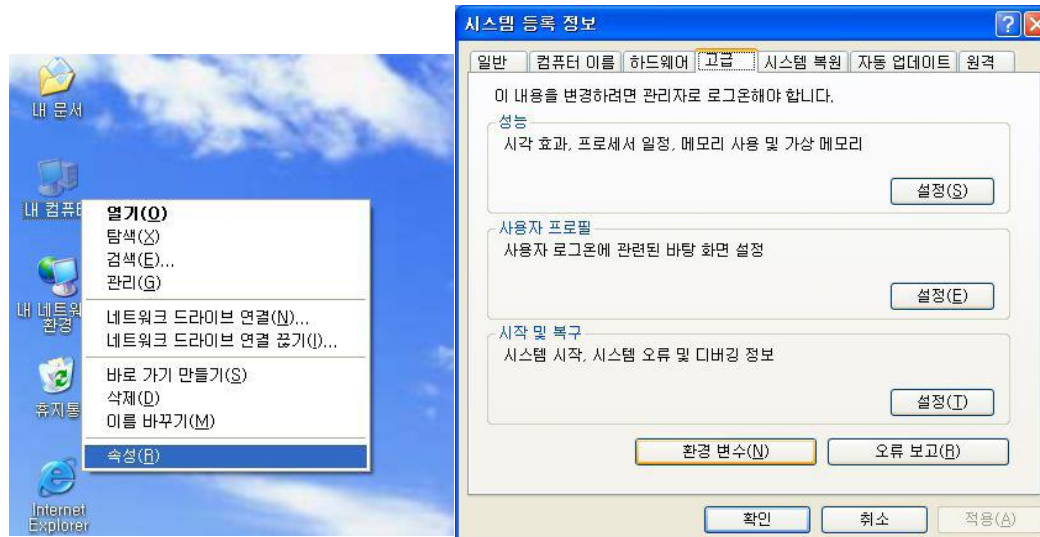


4. 개발환경 구축

■ 개발환경 설치

■ Cygwin 설치

- Cygwin 설치가 끝나면 환경변수에 cygwin 설치 경로를 등록해야 합니다.
- 바탕화면의 “내 컴퓨터” 아이콘에 마우스 오른쪽 버튼을 클릭 하여 “속성” 윈도우를 열고 고급 탭에서 “환경 변수” 버튼을 클릭합니다.

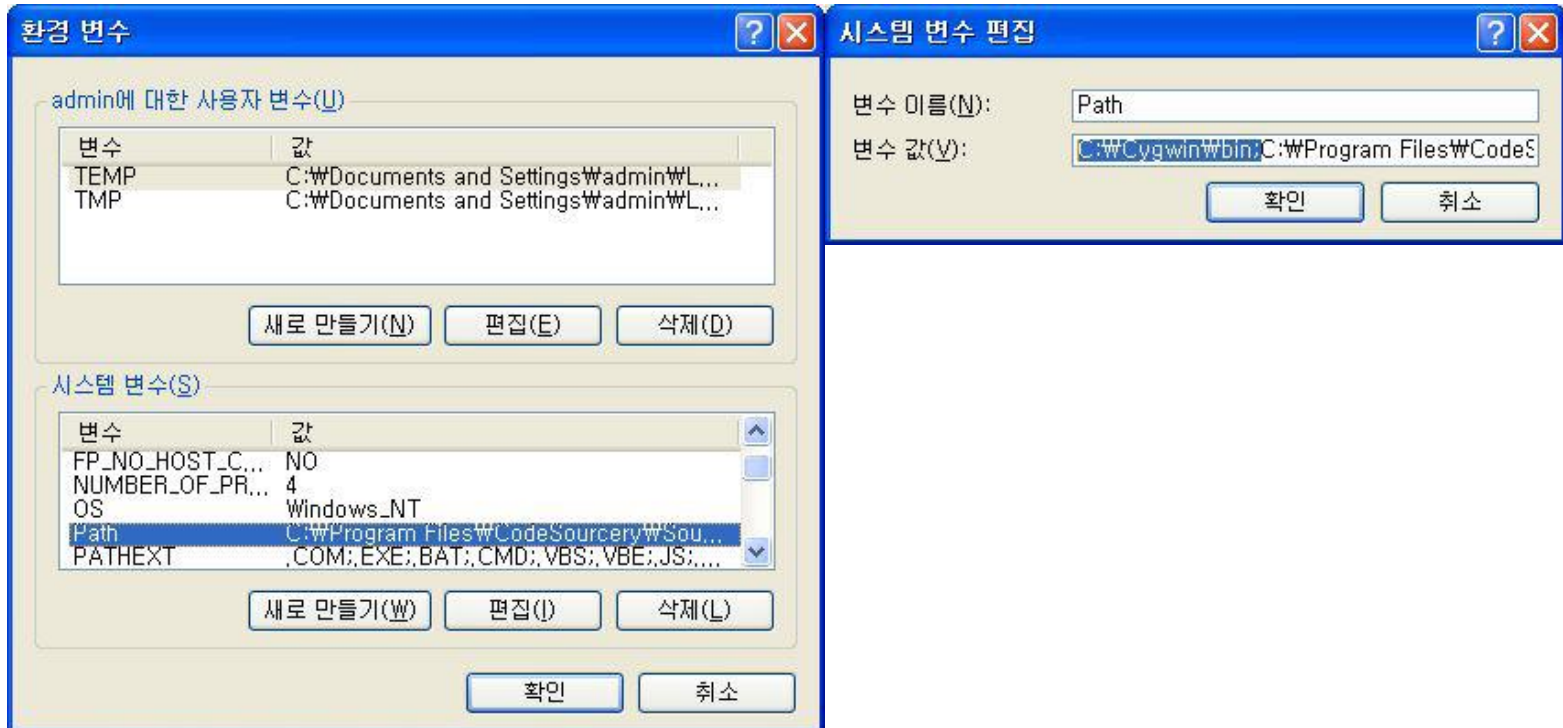


4. 개발환경 구축

■ 개발환경 설치

■ Cygwin 설치

- Cygwin 설치가 끝나면 환경변수에 cygwin 설치 경로를 등록해야 합니다.
- 바탕화면의 "내 컴퓨터" 아이콘에 마우스 오른쪽 버튼을 클릭 하여 "속성" 윈도우를 열고 고급 탭에서 "환경 변수" 버튼을 클릭합니다.



4. 개발환경 구축

■ 개발환경 설치

■ Cygwin 설치

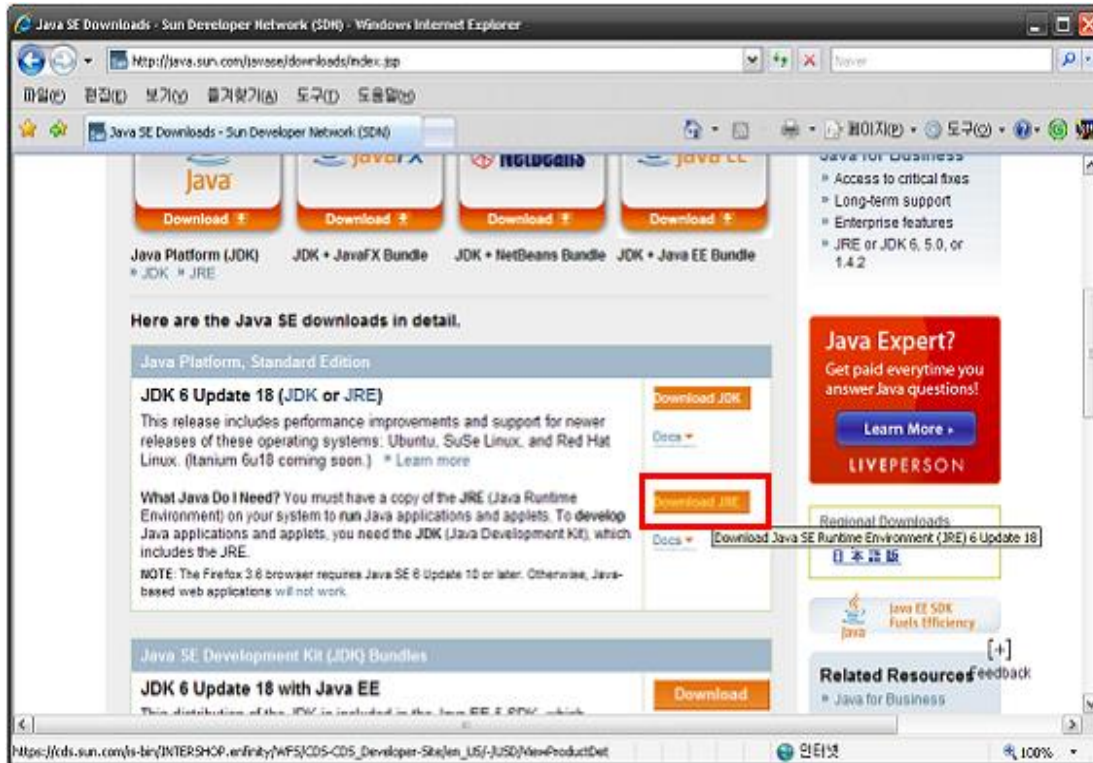
- "환경 변수" 설정 윈도우에서 아래쪽의 "시스템 변수" 내에 있는 "Path"를 선택한 후 "편집" 버튼을 클릭하고, "변수 값" 항목의 가장 앞에 cygwin 이 설치된 경로를 추가합니다.
- 이때 cygwin 이 C:\Cygwin 에 설치되었다면 C:\Cygwin\bin 과 같이 " bin"을 덧붙여서 추가해야 합니다.

4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ 준비물

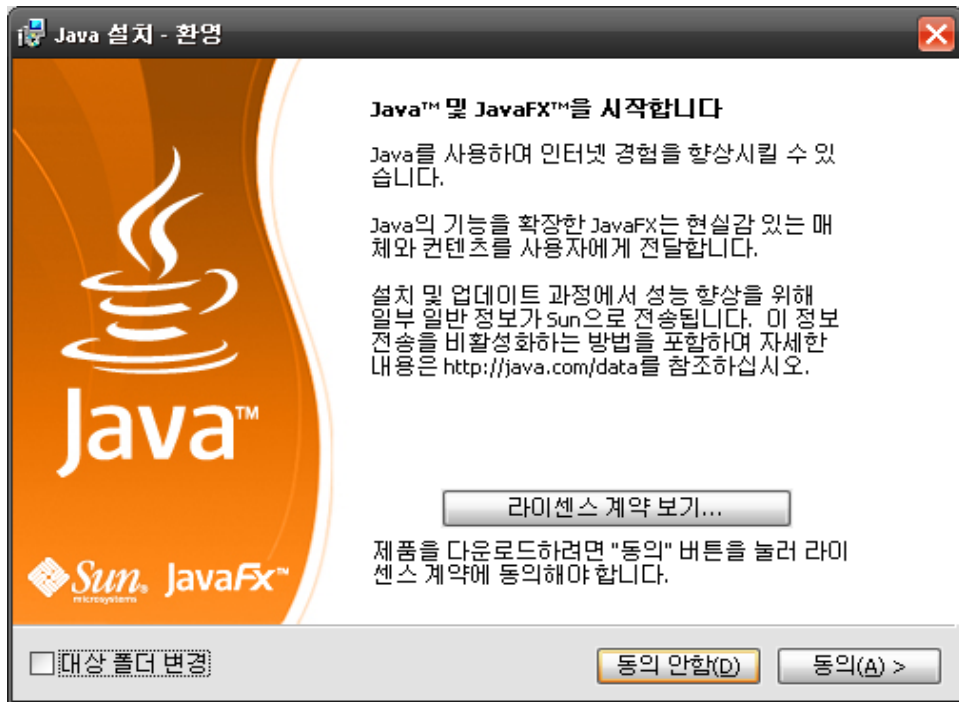
- JRE (<http://java.sun.com/javase/downloads/index.jsp>)
 - Eclipse가 Java 기반 프로그램이므로 구동을 위해서는 반드시 JRE 설치가 선행되어야 함.
- Eclipse C/C++ 3.x 이상
- LM3S8962(Cortex-M3)관련 Driver Library



4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

▪ JRE 설치



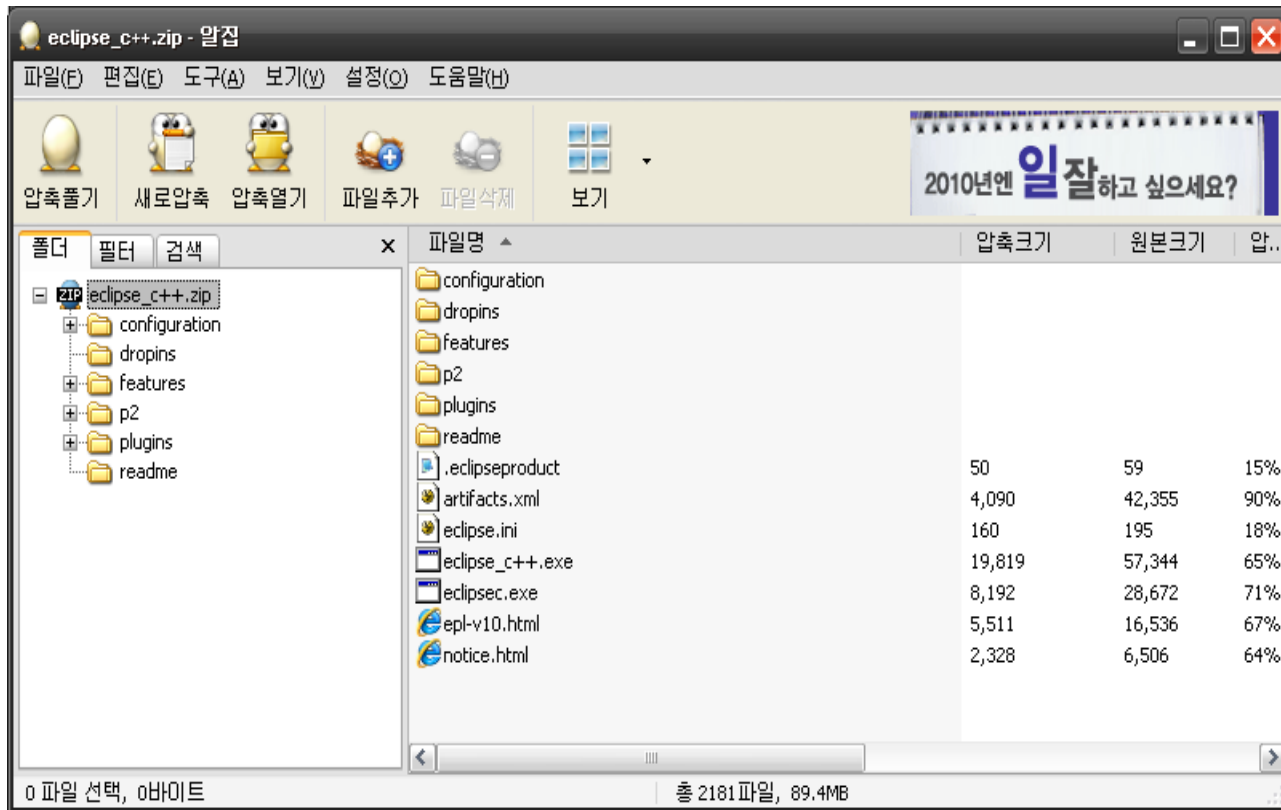
- 동의를 선택하여 설치를 진행하며, 별도의 변경사항 없이 "다음"으로 계속 진행한다.

4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse C/C++ 설치

- eclipse_c++.zip 파일을 적당한 디렉토리에 압축을 해제 한다.

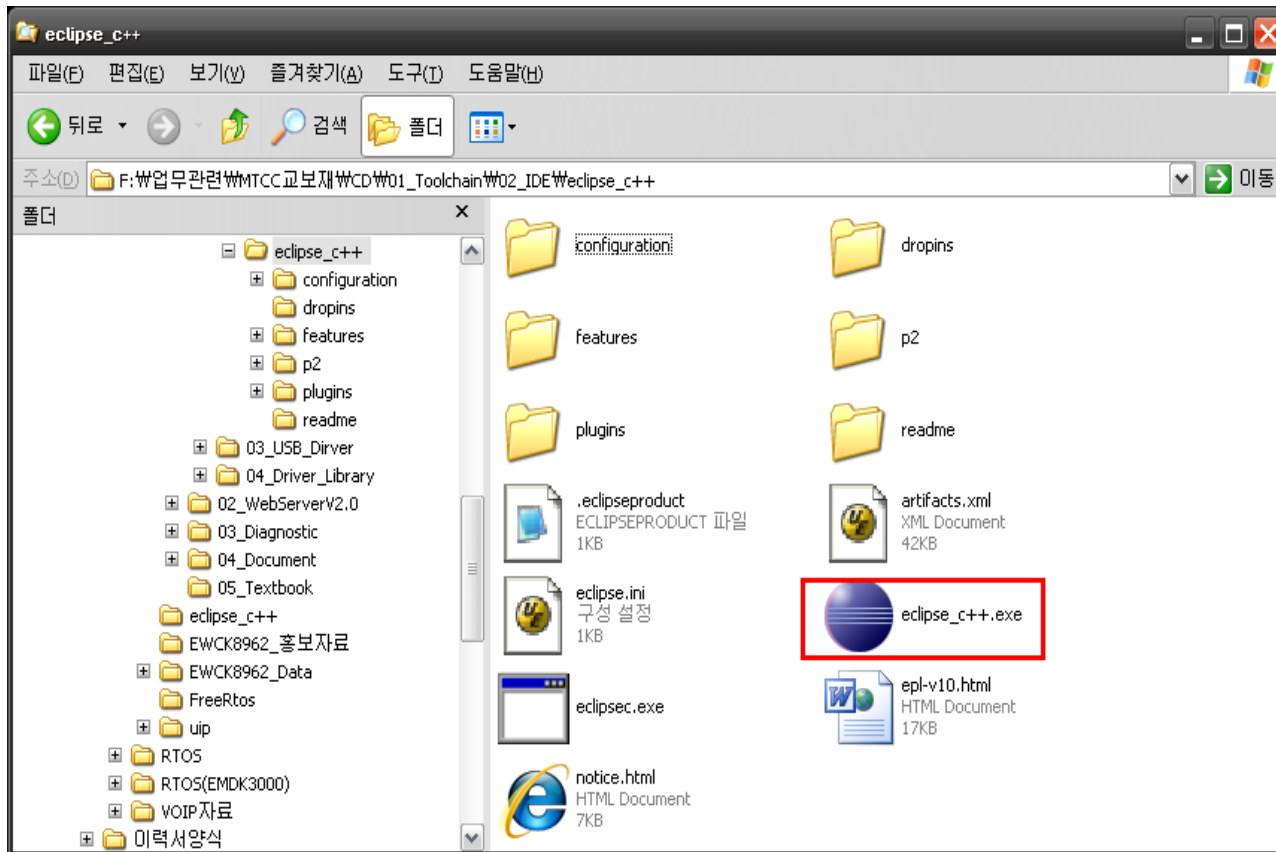


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse C/C++ 설치

- 프로그램 단축 아이콘 만들기
 - eclipse c++ 압축을 해제한 디렉토리로 이동한다.
 - eclipse_c++.exe 파일을 바탕화면이나 단축창으로 드래그&드랍하여 바로가기 아이콘을 생성한다.

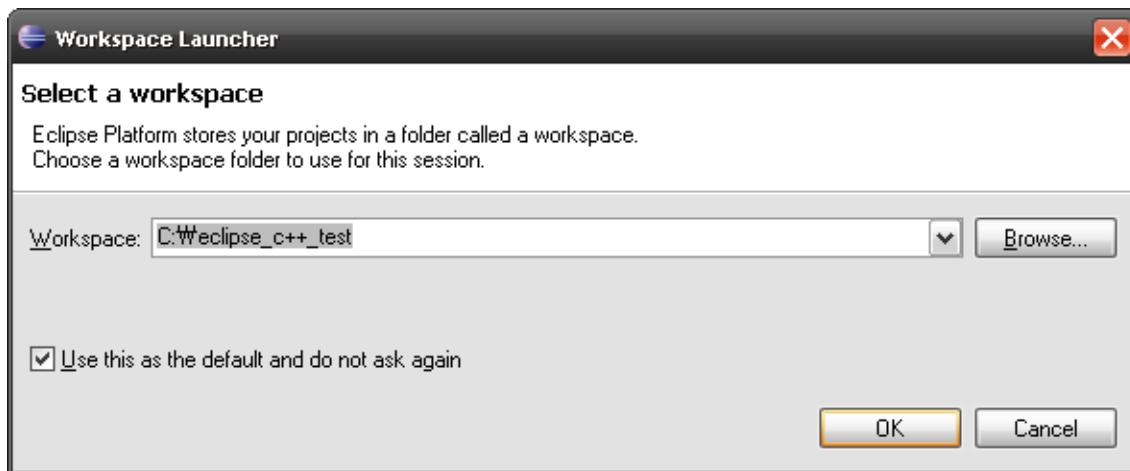


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse 실행

- 위에서 만들어 놓은 바로가기를 클릭하여 실행하거나 압축을 해제한 디렉토리에서 실행 파일을 클릭하여 프로그램을 실행한다.
 - Eclipse 작업 공간 설정 화면

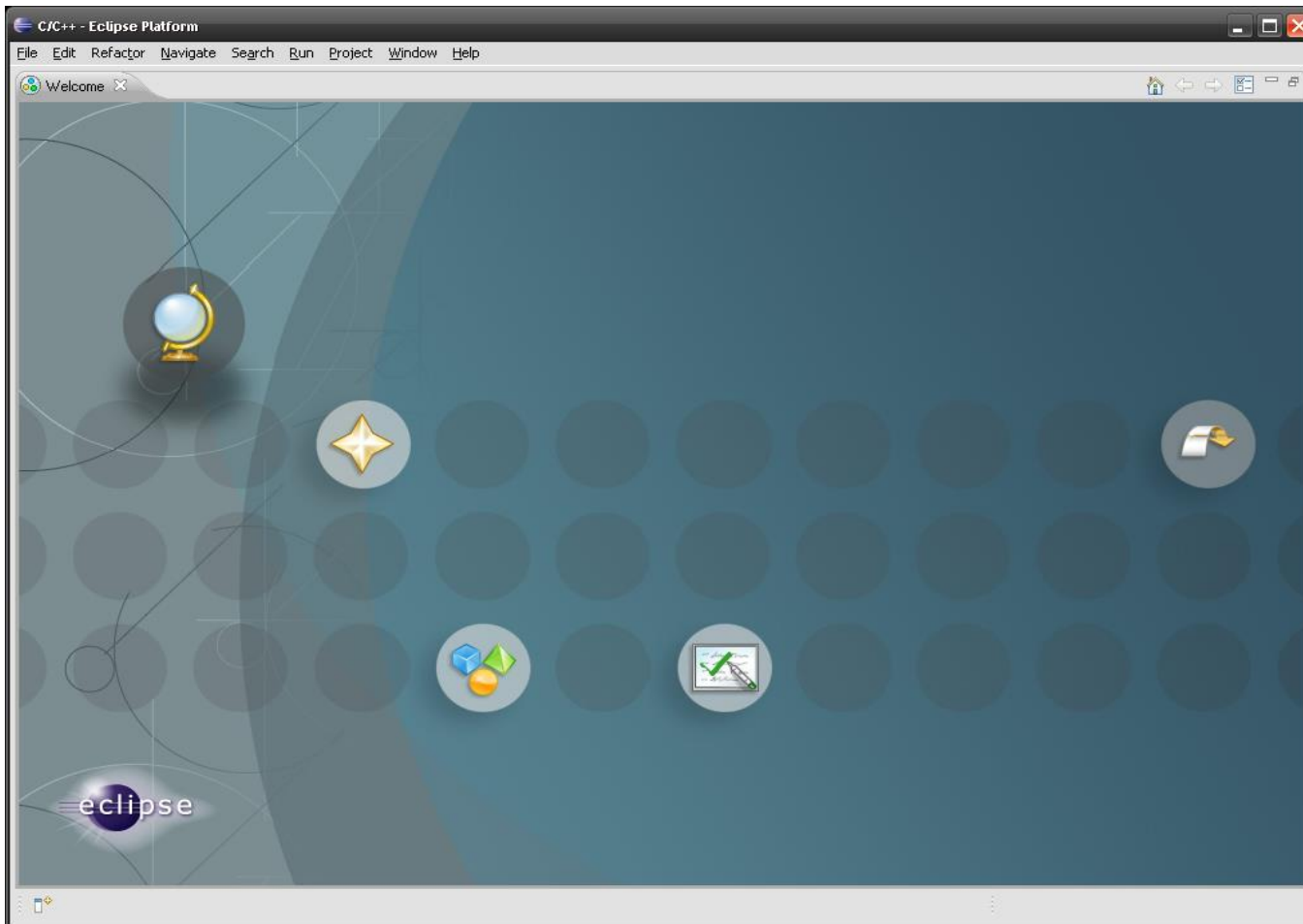


- Eclipse를 실행하면 가장 먼저 나타나는 화면으로, 프로젝트를 생성할 작업공간을 선택하는 화면으로 적당한 곳에 작업공간을 설정해두도록 한다.

4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

- Eclipse 실행
 - Eclipse 시작 화면

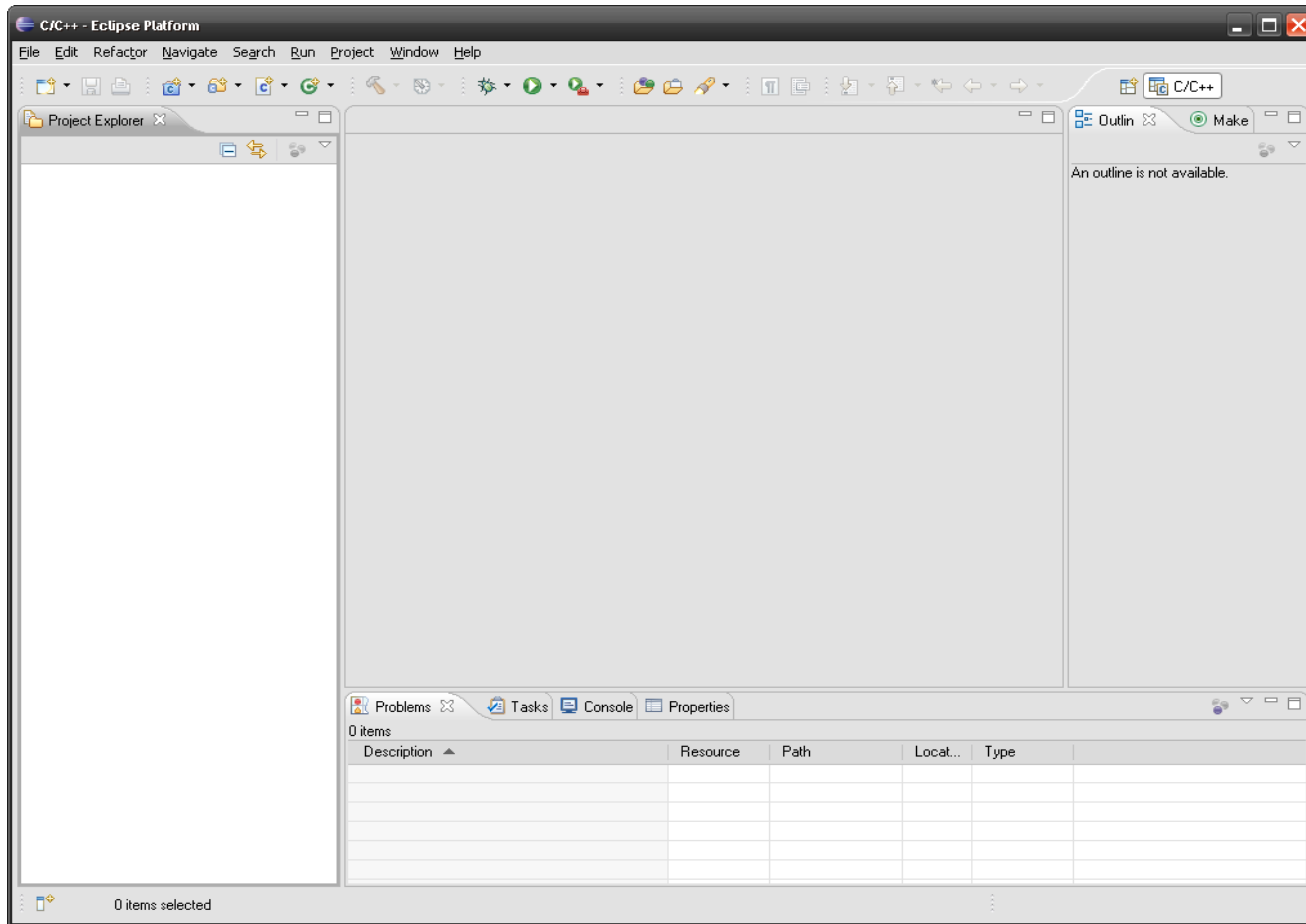


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse 실행

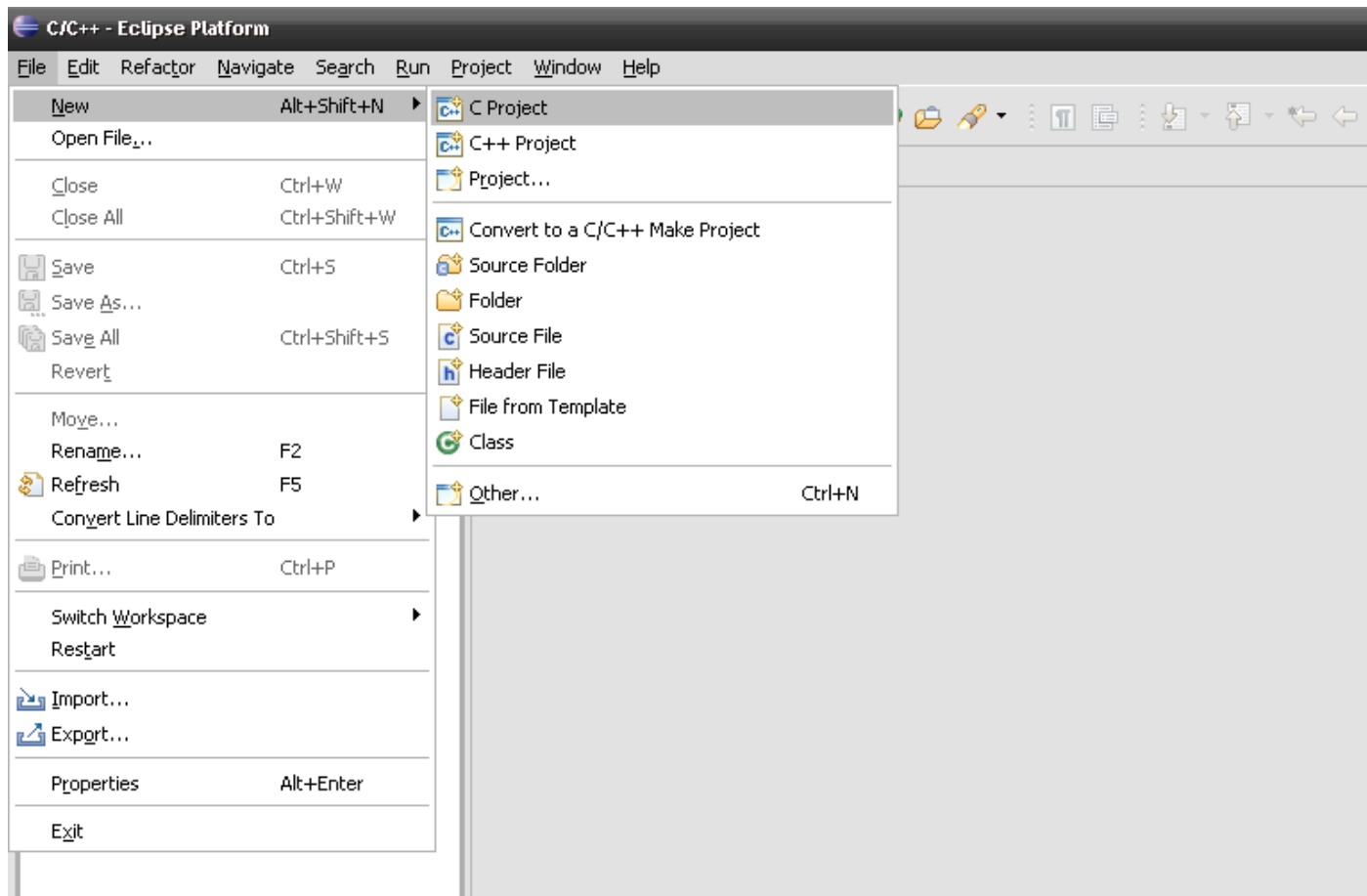
- Welcome 탭의 'X' 버튼을 눌러 창을 닫게되면 아래와 같은 화면으로 바뀌게 된다. 아래 그림이 실질적인 작업 화면이 된다.



4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

- Eclipse에서 프로젝트 생성하기
 - 메뉴에서 File -> New -> C Project를 선택한다.

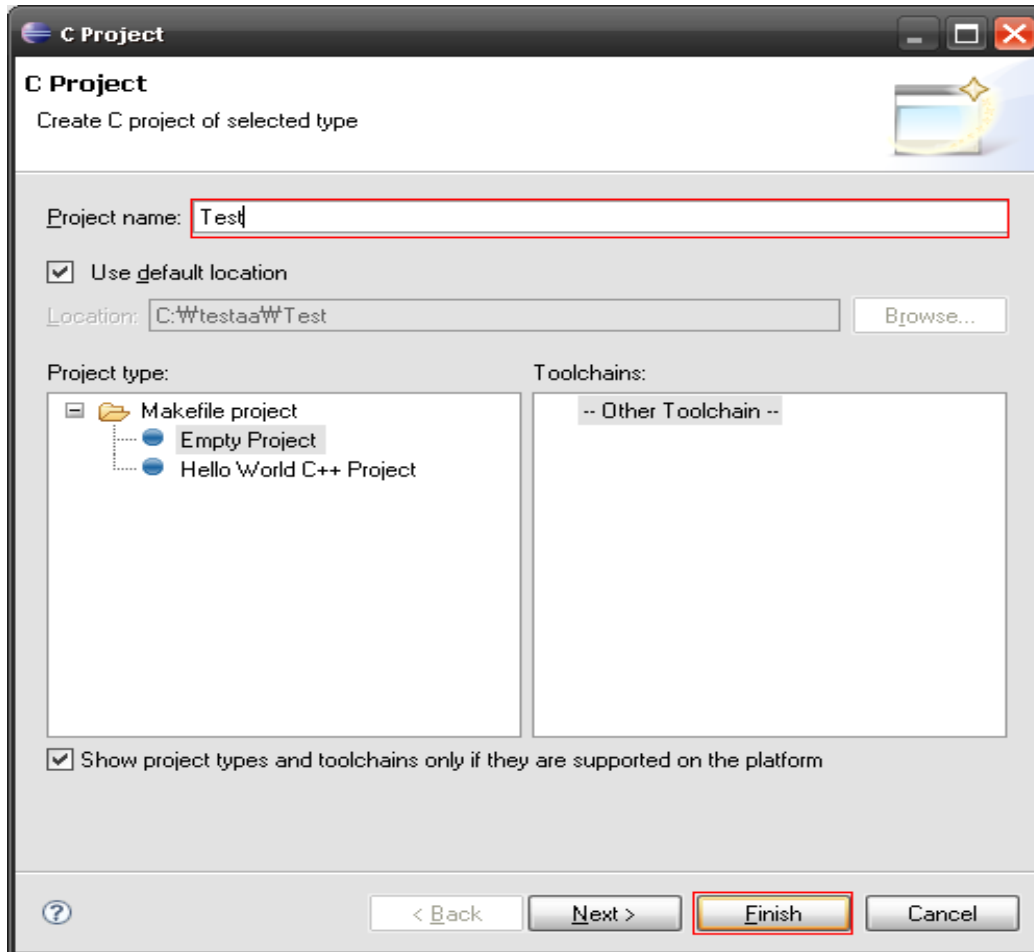


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse에서 프로젝트 생성하기

- 프로젝트 이름을 입력하고 Finish 버튼을 누른다.

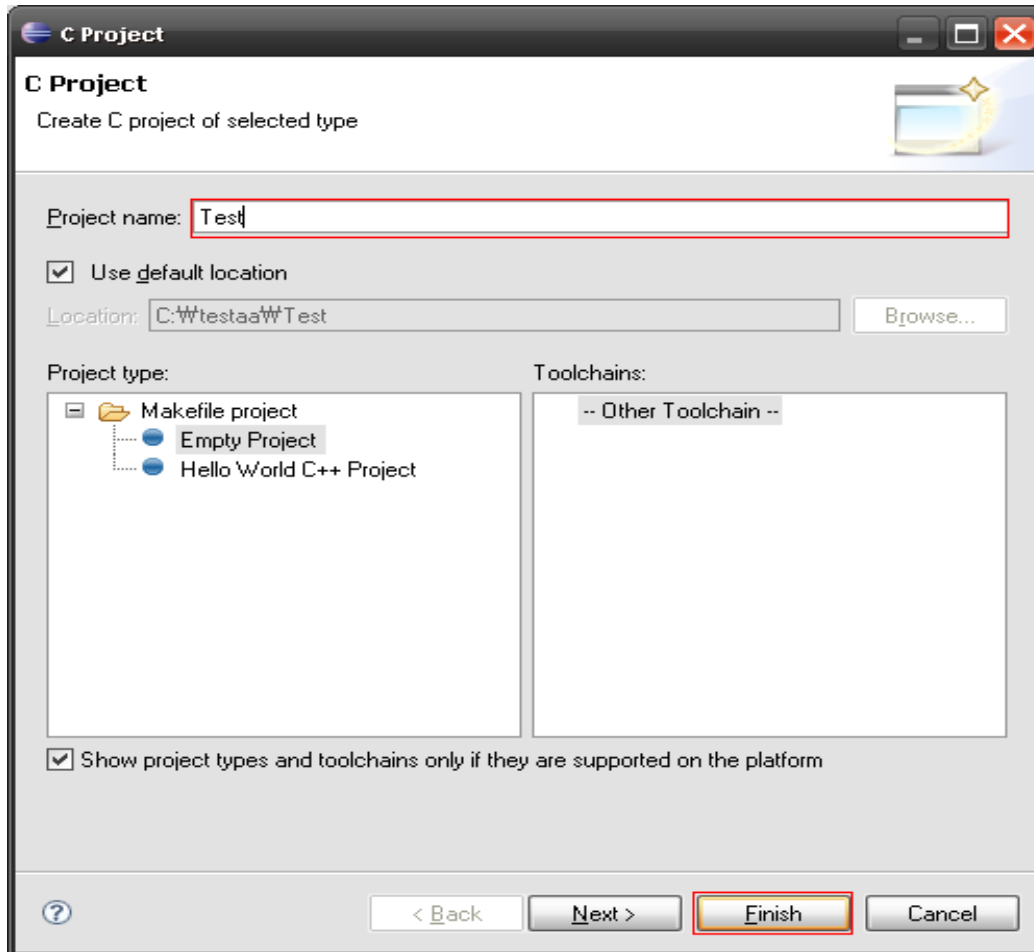


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse에서 프로젝트 생성하기

- 프로젝트 이름을 입력하고 Finish 버튼을 누른다.

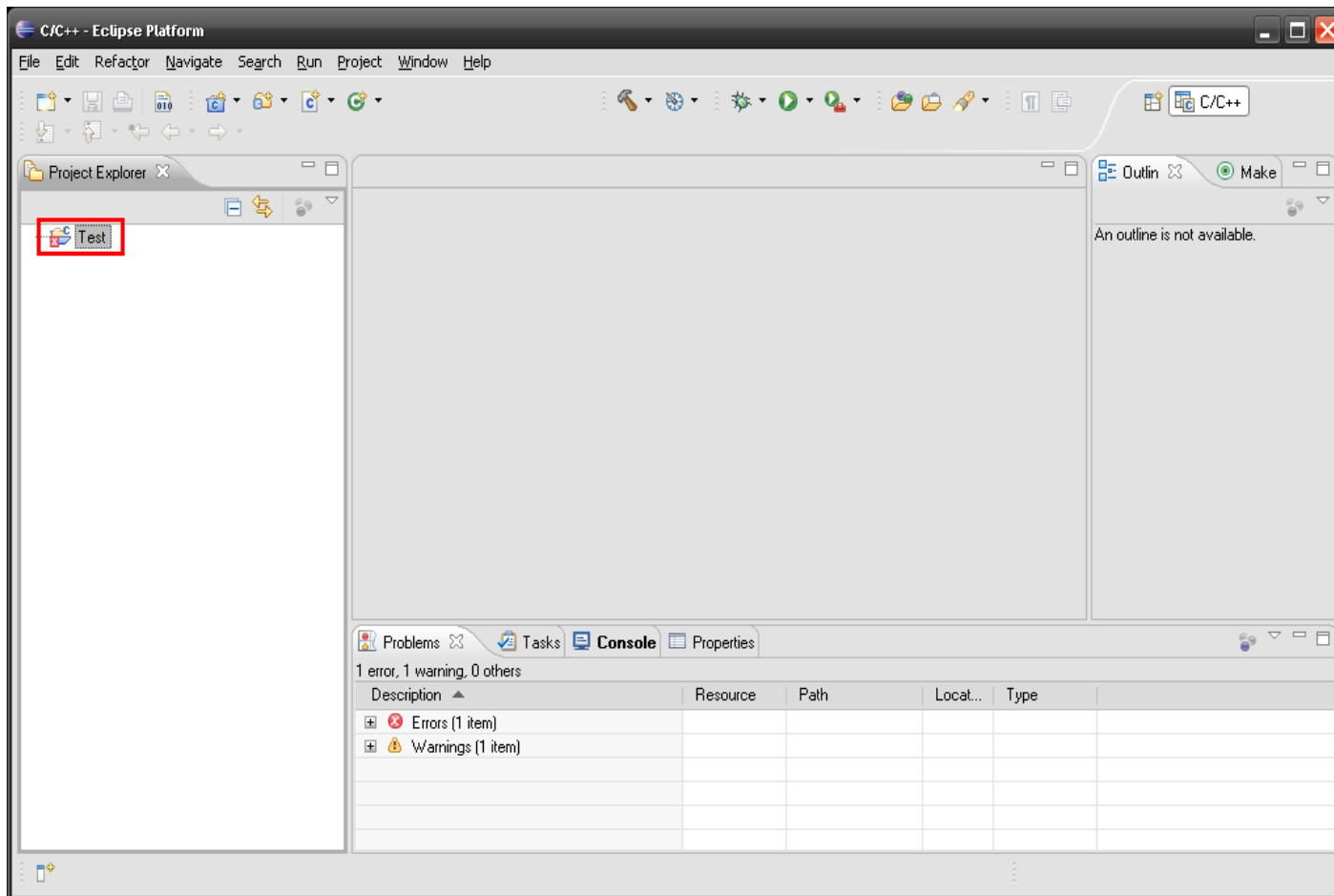


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse에서 프로젝트 생성하기

- Test라는 이름의 새로운 프로젝트가 생성되어진 것을 확인할 수 있다.

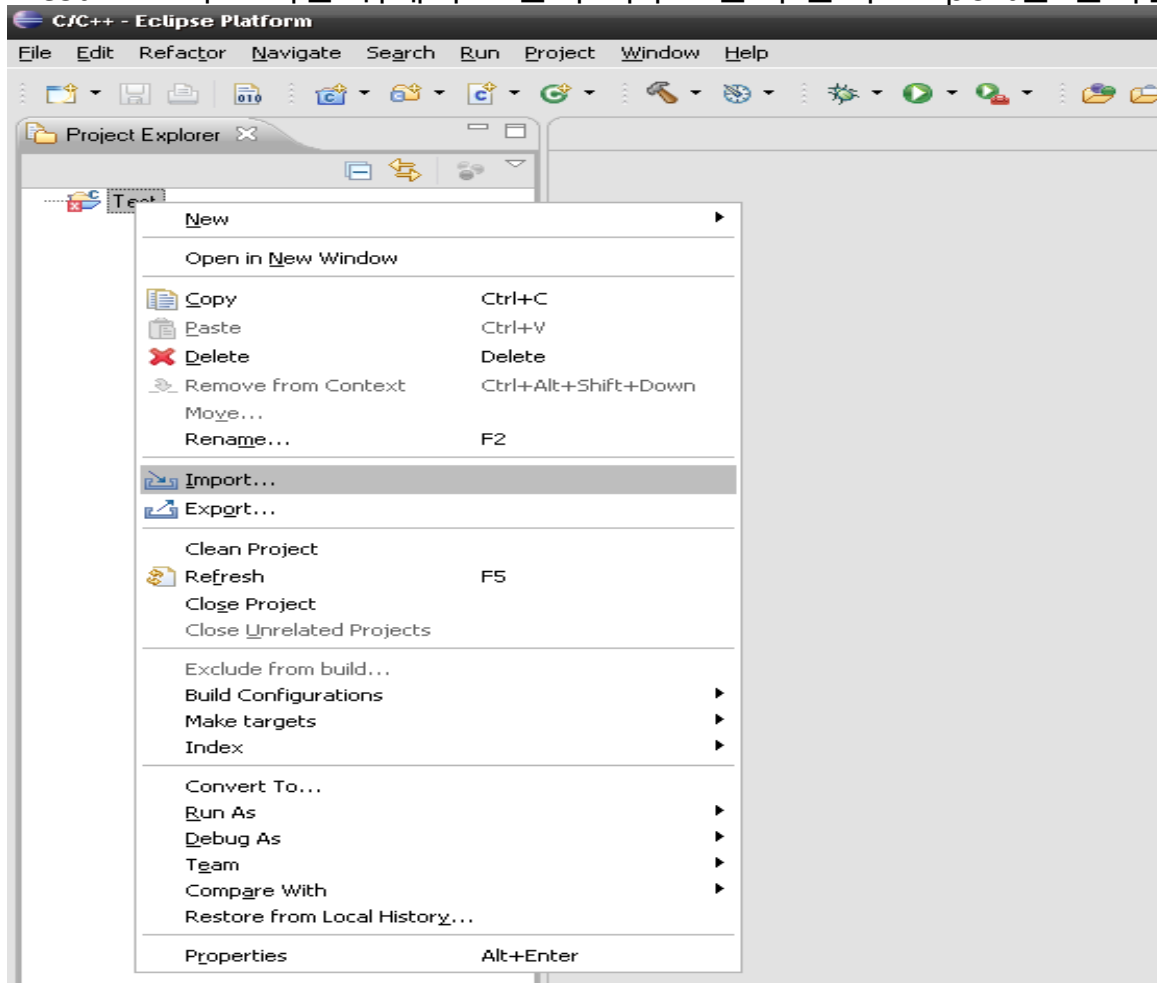


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse에서 프로젝트 생성하기

- Test 프로젝트 이름 위에서 오른쪽 마우스를 누른 후 Import를 선택한다.

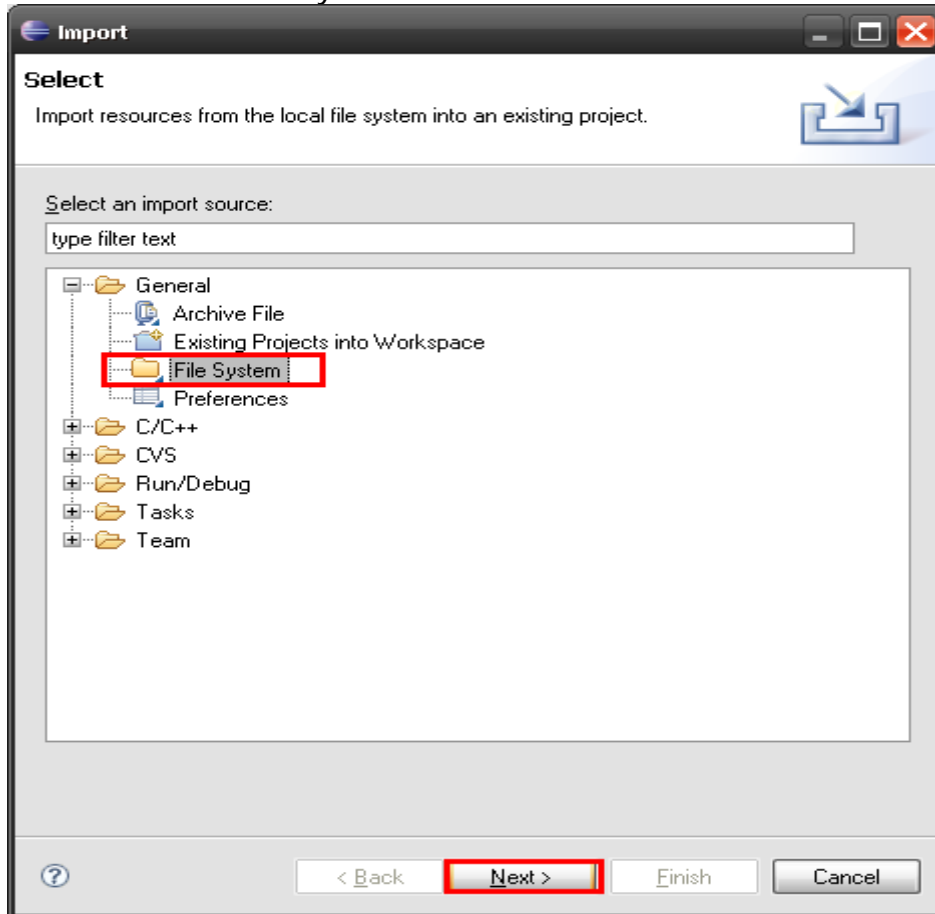


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse에서 프로젝트 생성하기

- General에서 FileSystem을 선택하고 Next 버튼을 누른다.

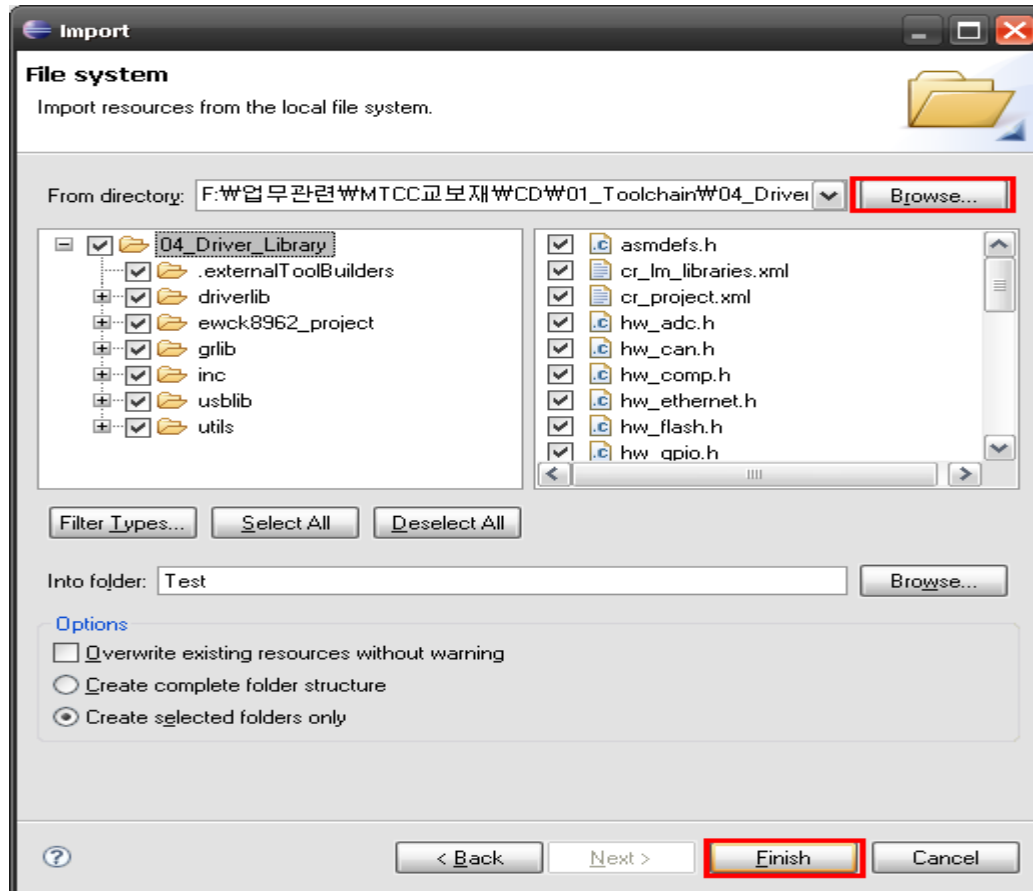


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse에서 프로젝트 생성하기

- Browse 버튼을 눌러 CD의 01_Toolchain -> 04_Driver_Library 디렉토리를 선택하도록 한다.
- 선택을 하고 확인을 누르게 되면 위 그림과 같이 디렉토리와 파일들이 화면에 보여지게 된다.

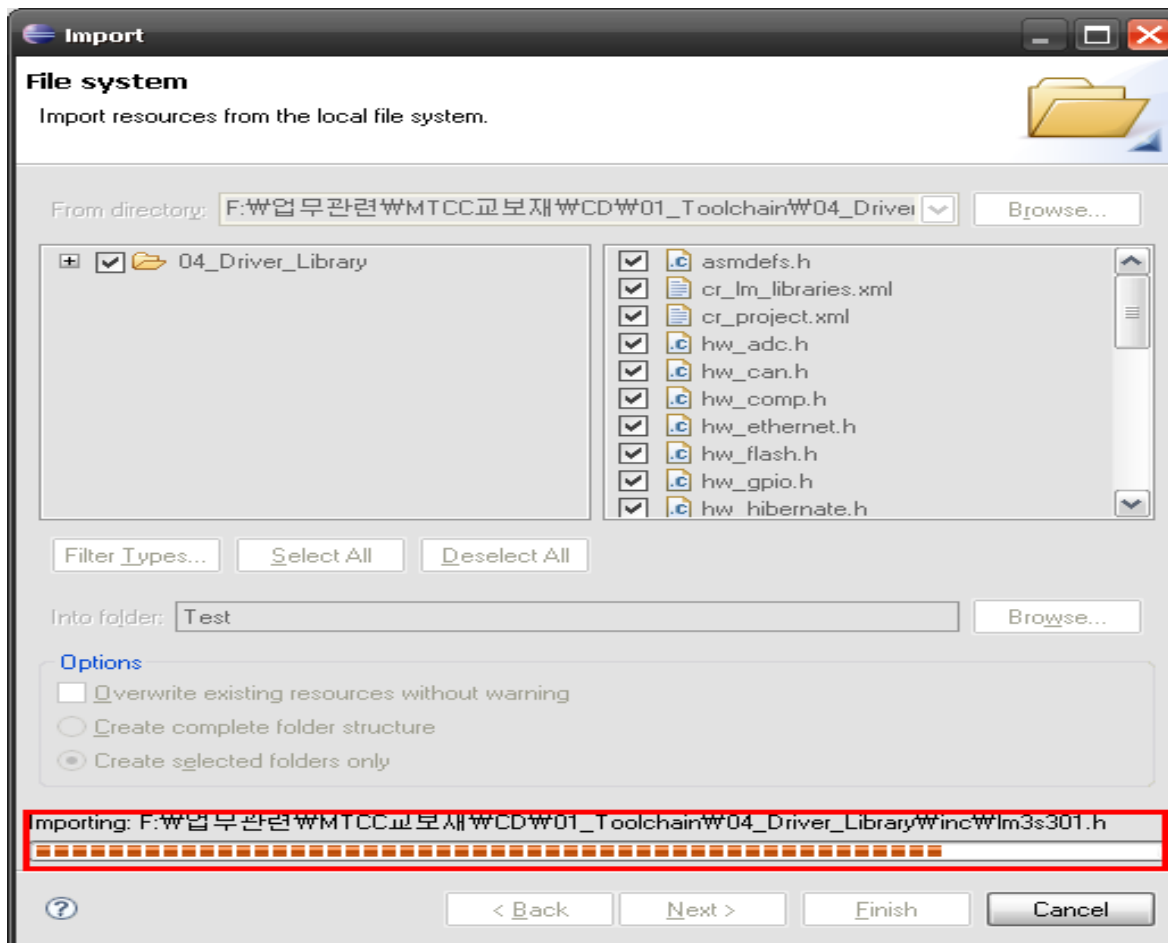


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse에서 프로젝트 생성하기

- Finish를 눌러 드라이버 라이브러리 추가를 마치도록 한다.

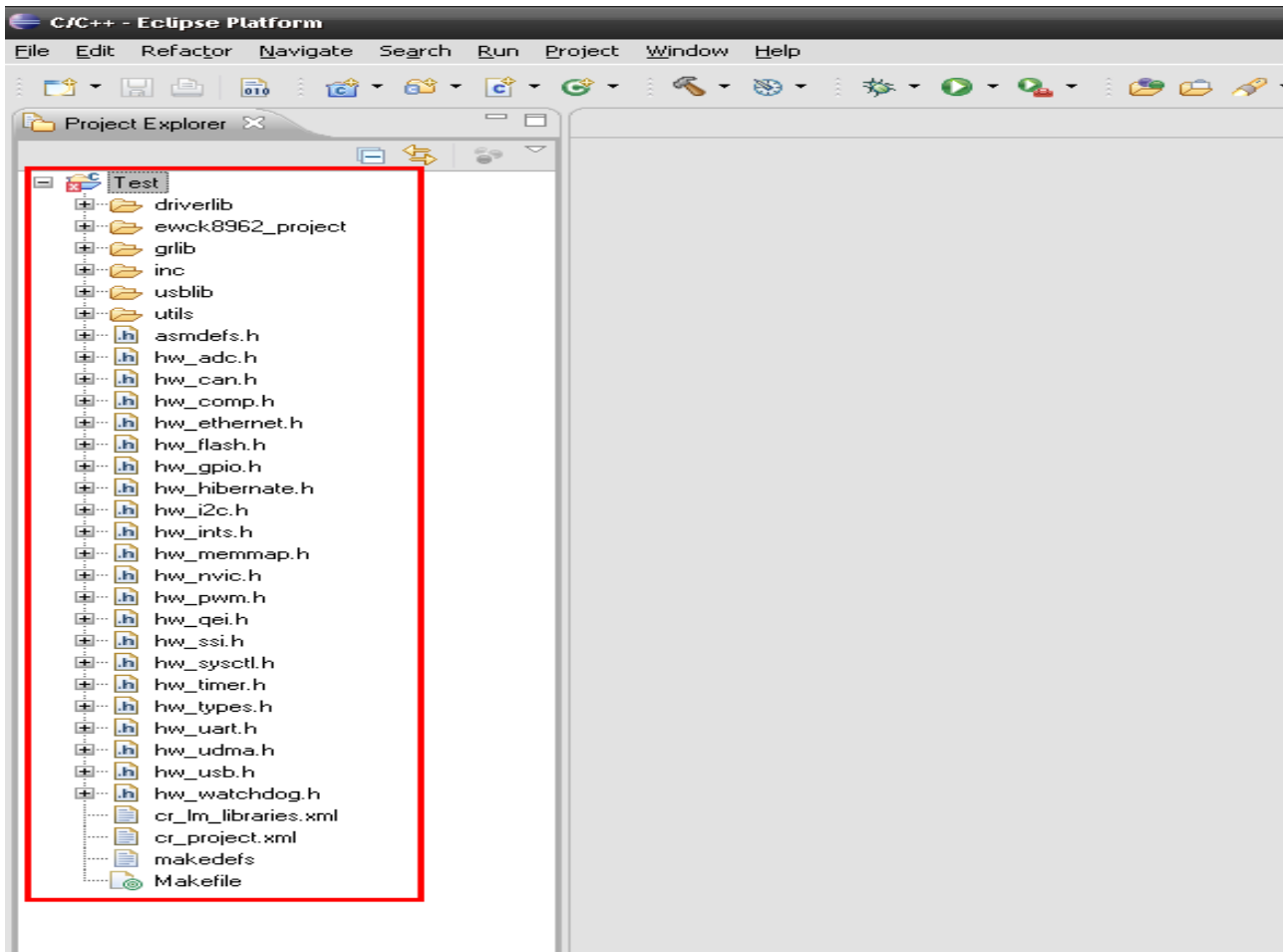


4. 개발환경 구축

■ Target Board 펌웨어 프로그래밍

■ Eclipse에서 프로젝트 생성하기

- 추가가 완료되고 난 후의 프로젝트 모습. 파일, 디렉토리가 추가되어 있는 것을 확인할 수 있다.

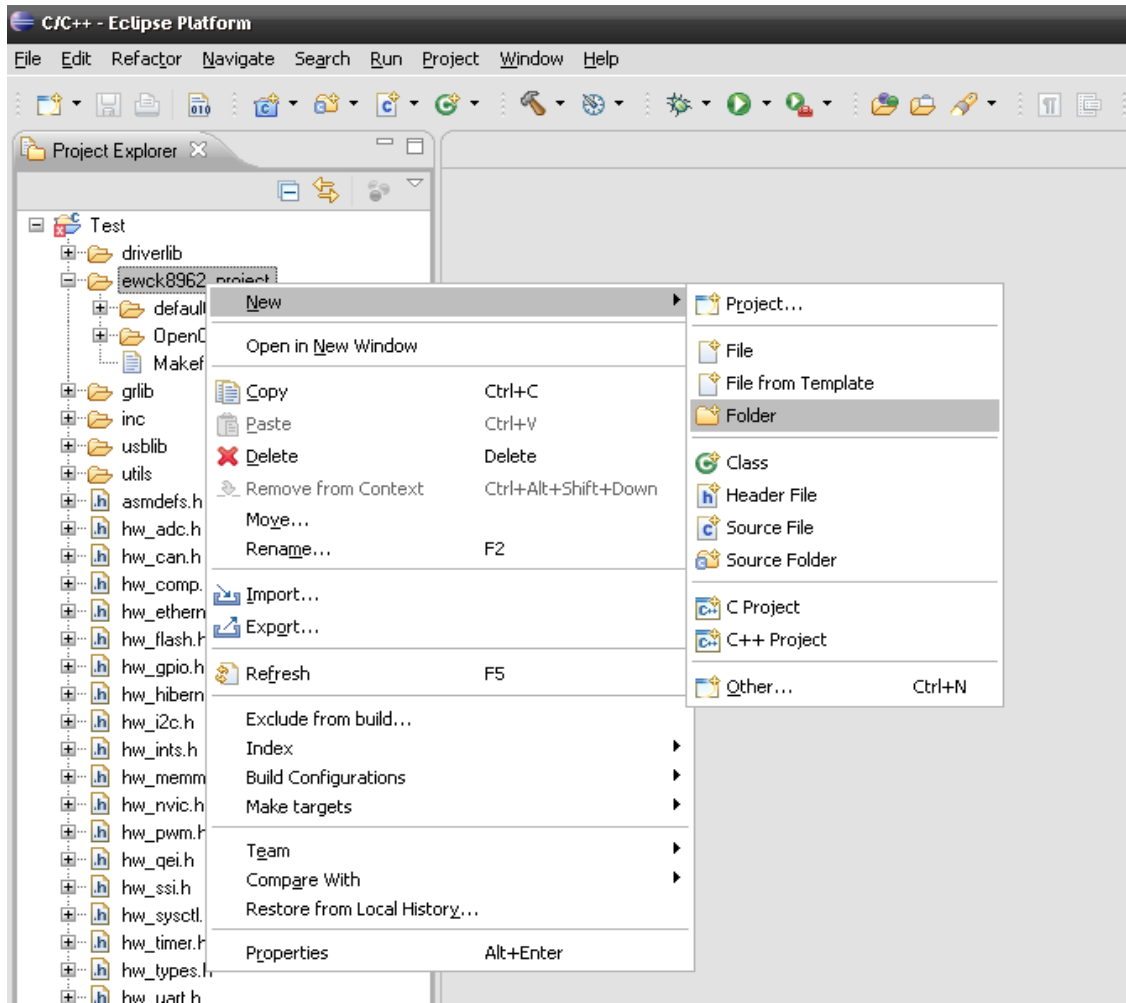


5. LED 제어 프로그램 추가하기

- 이전까지 만들어 놓은 프로젝트에서 ewck8962_project 트리를 열게 되면 default_files라는 폴더가 존재한다.
- 이 폴더는 각 개별 프로젝트를 생성하기 위한 가장 기본이 되는 설정 파일들이 포함되어 있는 폴더로 새로운 I/O 제어 프로그램을 만들때마다 해당 내용을 복사하여 사용하도록 한다.

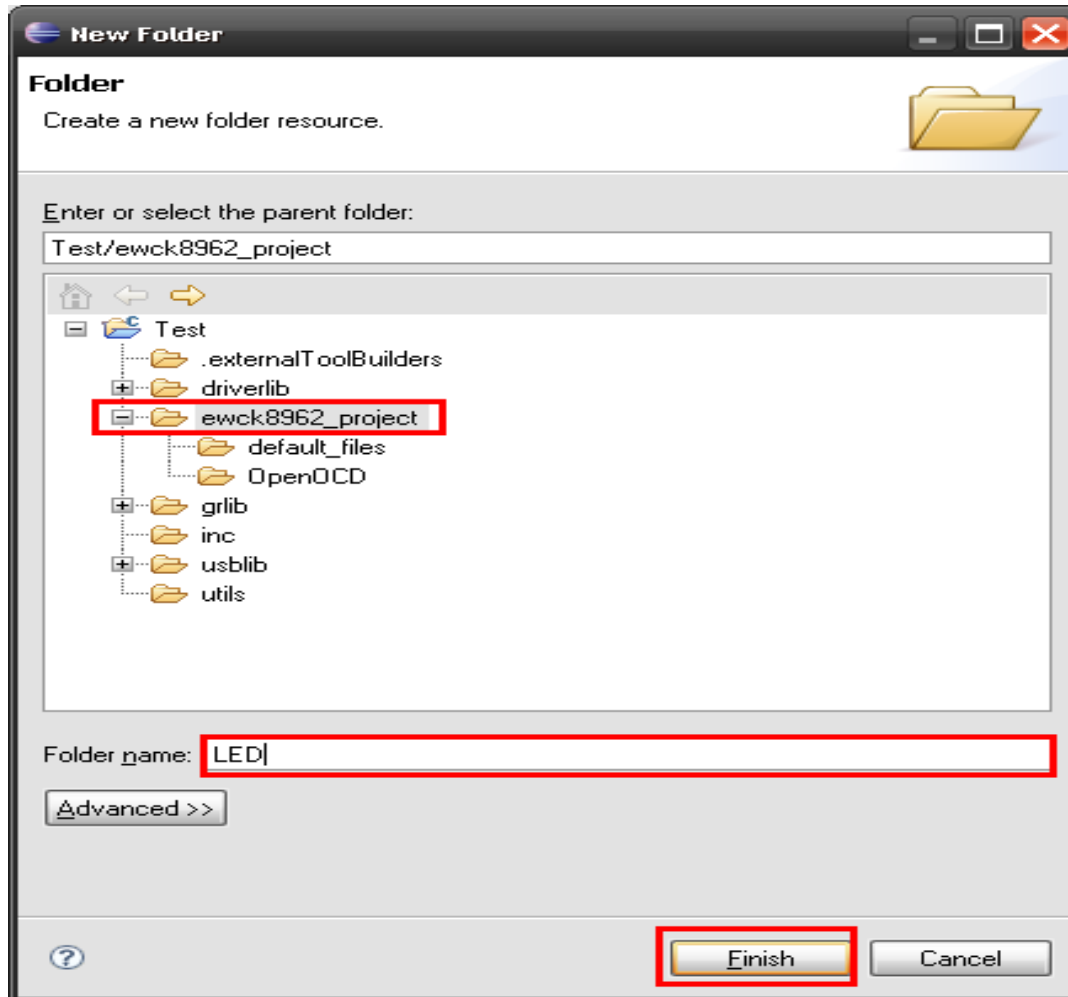
5. LED 제어 프로그램 추가하기

■ LED 제어 프로그램 폴더 만들기



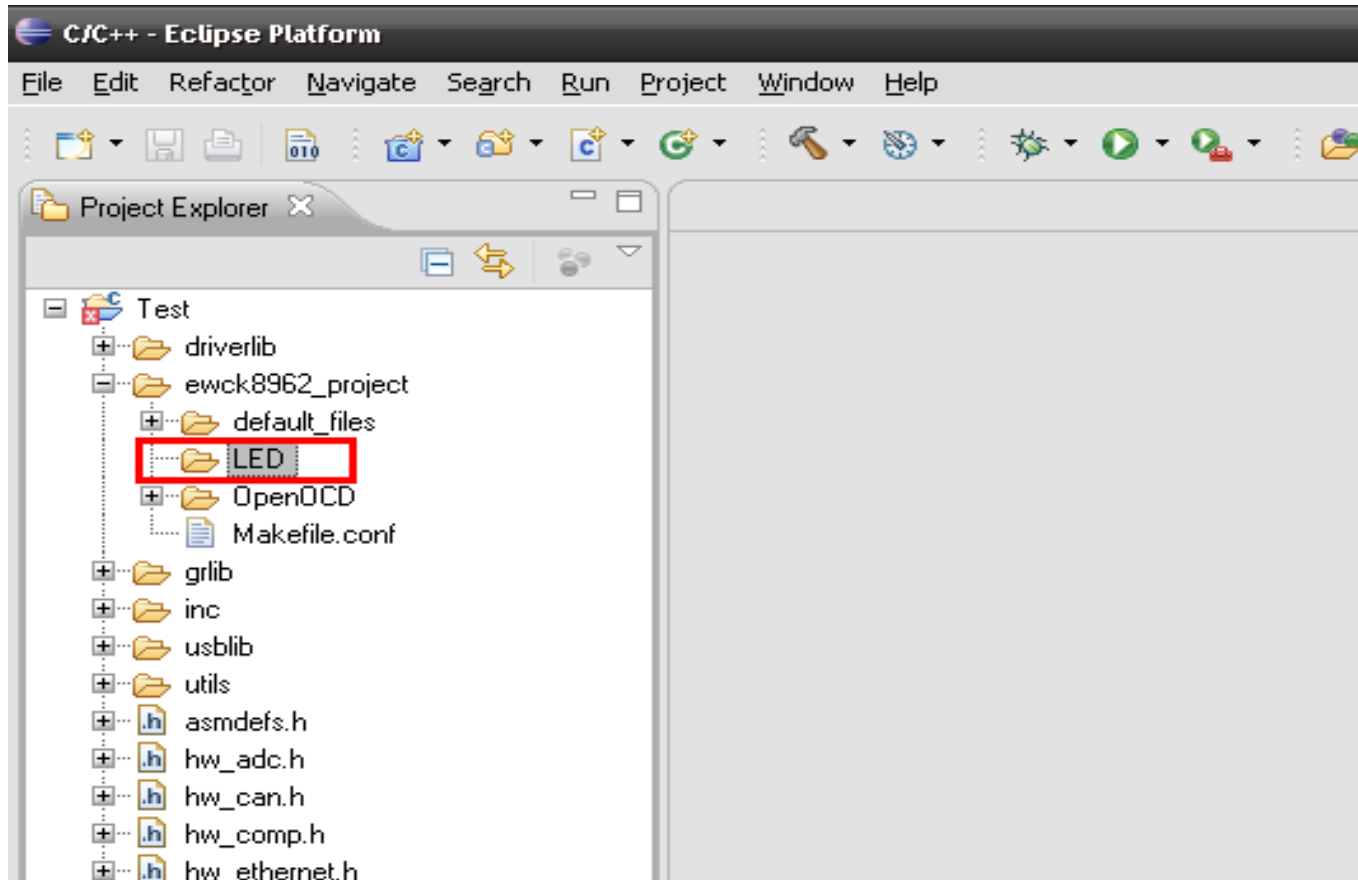
5. LED 제어 프로그램 추가하기

- 우선 ewck8962_project 폴더 위에서 오른쪽 마우스를 눌러 New -> Folder을 선택하여 LED 폴더를 만들도록 한다.



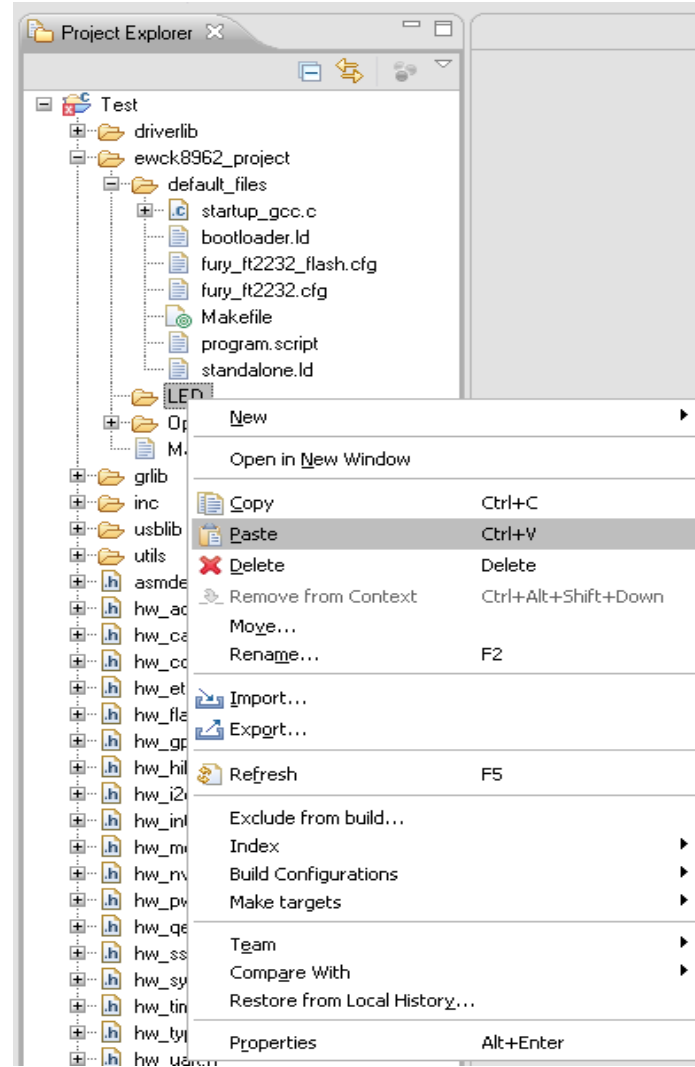
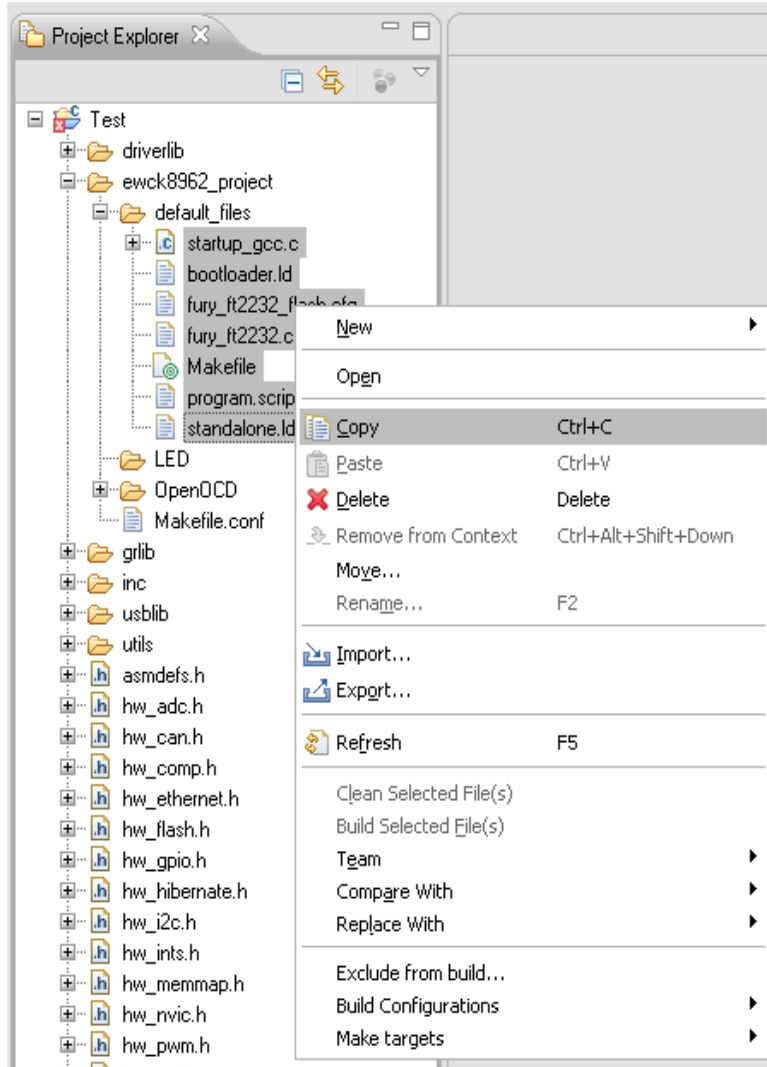
5. LED 제어 프로그램 추가하기

- 폴더 추가를 마치게되면 프로젝트 화면에 LED 폴더가 추가되어진 것을 확인할 수 있다.



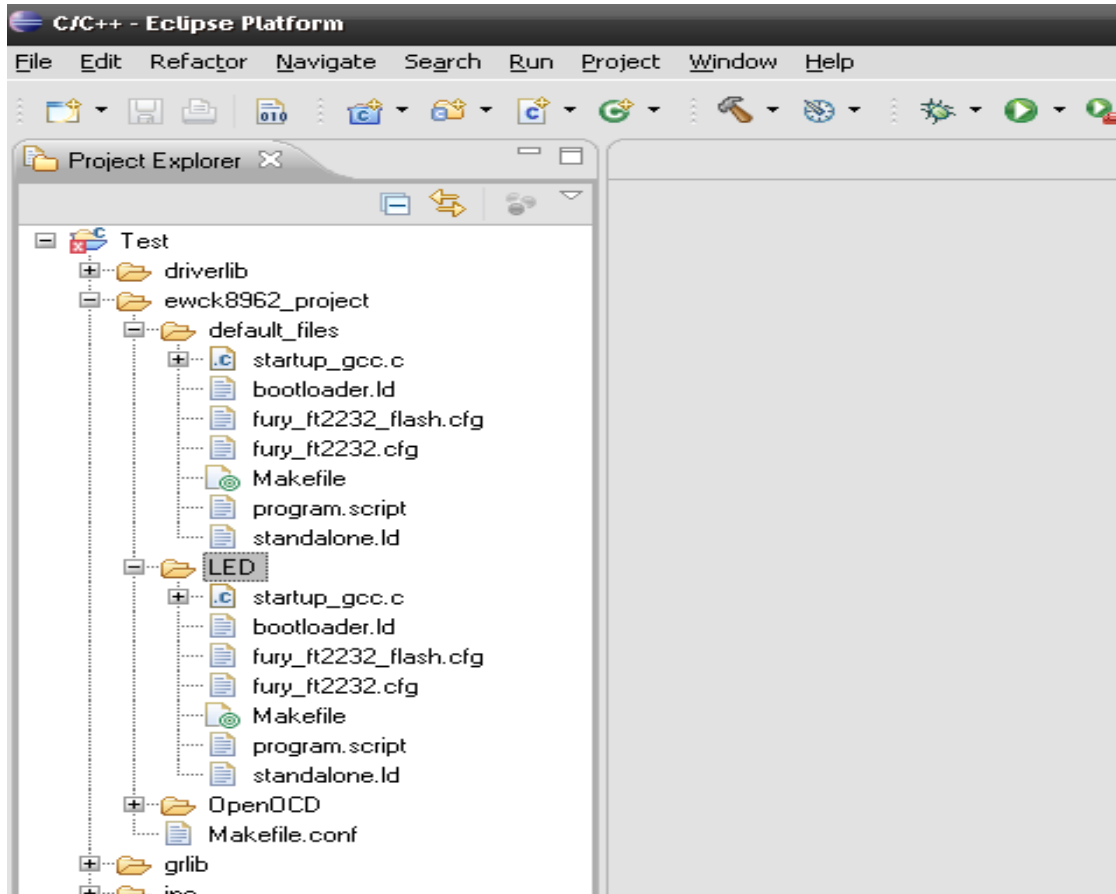
5. LED 제어 프로그램 추가하기

- 폴더가 만들어 졌다면 기본설정 파일을 LED 폴더에 복사하도록 한다.



5. LED 제어 프로그램 추가하기

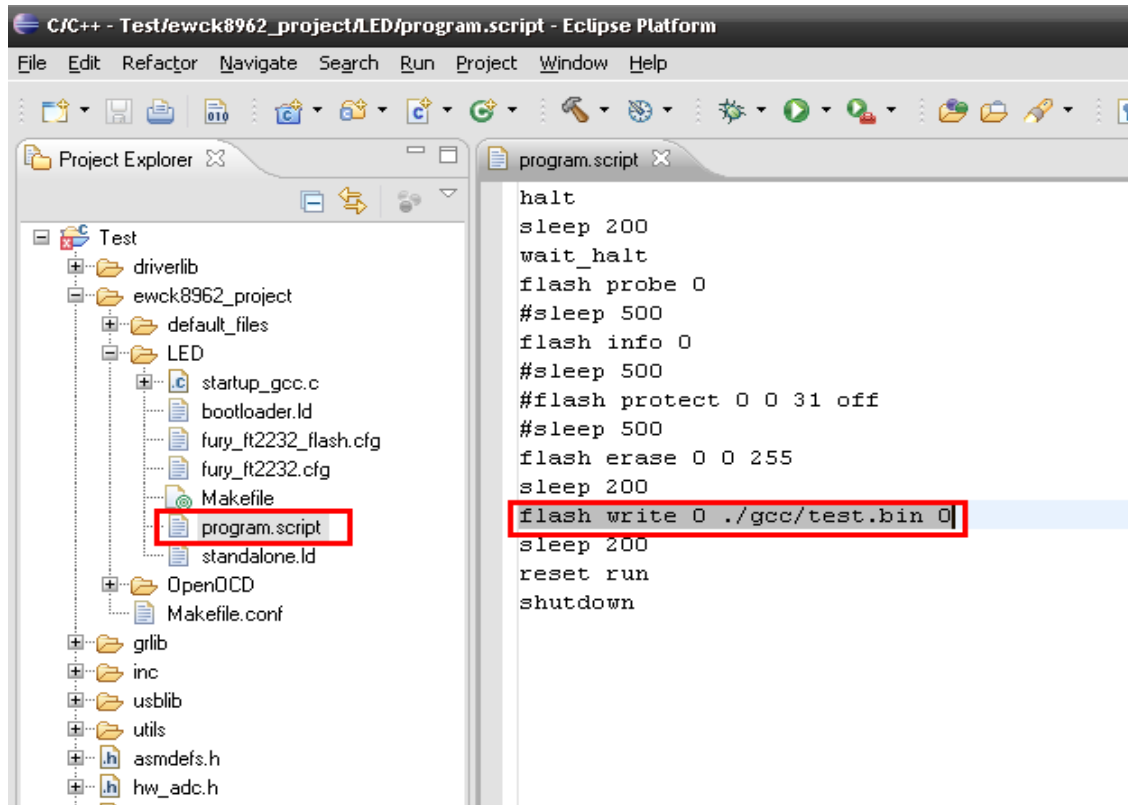
- 폴더가 만들어 졌다면 기본설정 파일을 LED 폴더에 복사하도록 한다.



5. LED 제어 프로그램 추가하기

■ 파일 복사를 마치게되면 LED 폴더에도 default_files에 있는 내용이 그대로 추가 되게 된다.

- 프로그램을 컴파일게되면 생성되는 파일 이름 수정.
- program.script 파일 수정



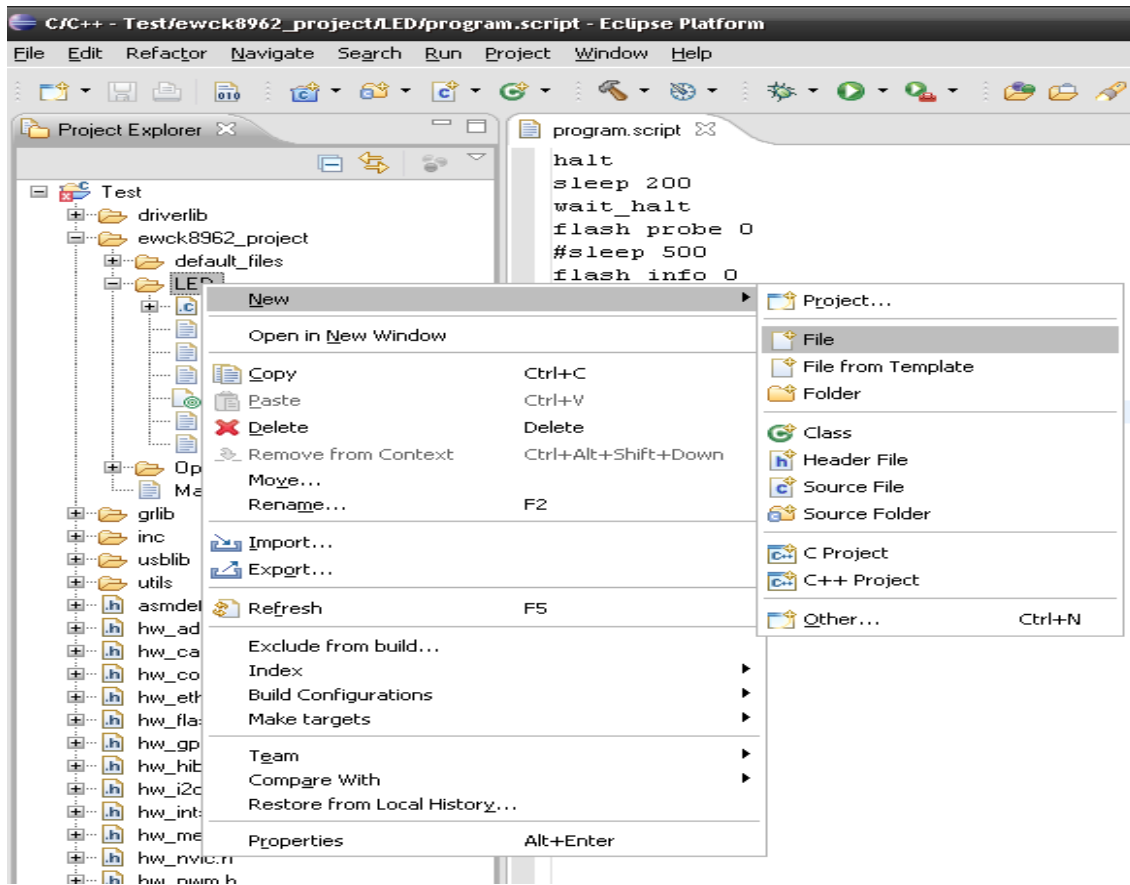
program.script 파일을 더블클릭하게 되면 우측 화면에 내용이 표시된다.

test.bin이라고 되어 있는 부분을 현재 폴더의 이름으로 변경해준다. 컴파일후 생성되는 바이너리 파일의 이름이 해당 폴더 이름으로 생성이 되어지기 때문에 변경작업은 반드시 해야만한다.

현재 폴더의 이름이 LED이므로 test를 LED로 변경하도록 한다.

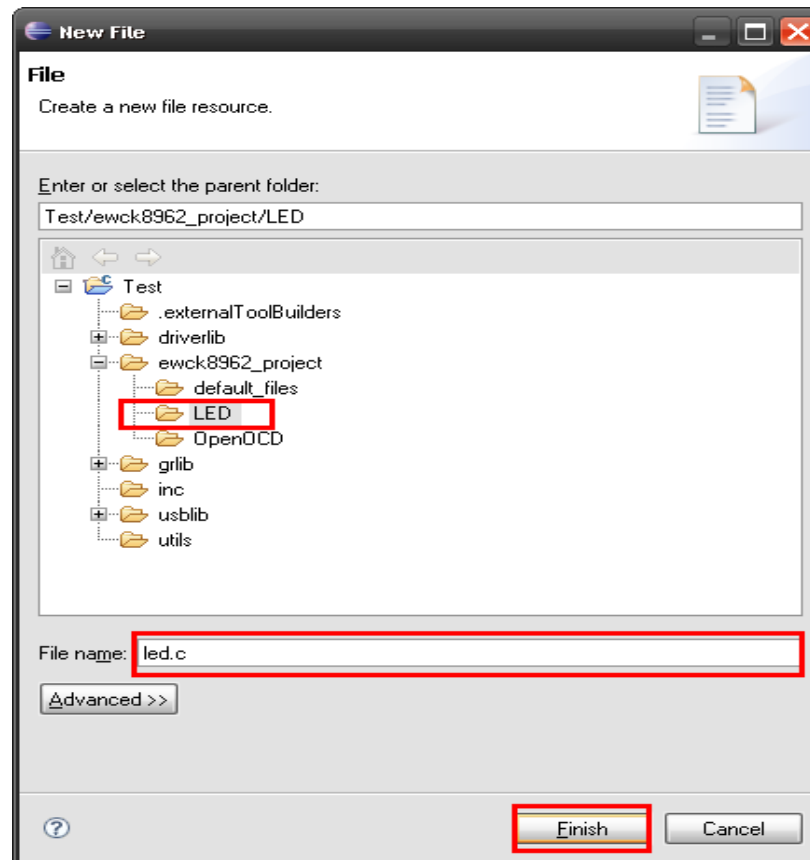
5. LED 제어 프로그램 추가하기

- 파일 복사를 마치게되면 LED 폴더에도 default_files에 있는 내용이 그대로 추가 되게 된다.
- 프로젝트 설정 마지막 단계로 소스 파일을 생성하도록 한다.
- (led.c, led.h, main.c 등....) 필요한 파일을 만들도록 한다.



5. LED 제어 프로그램 추가하기

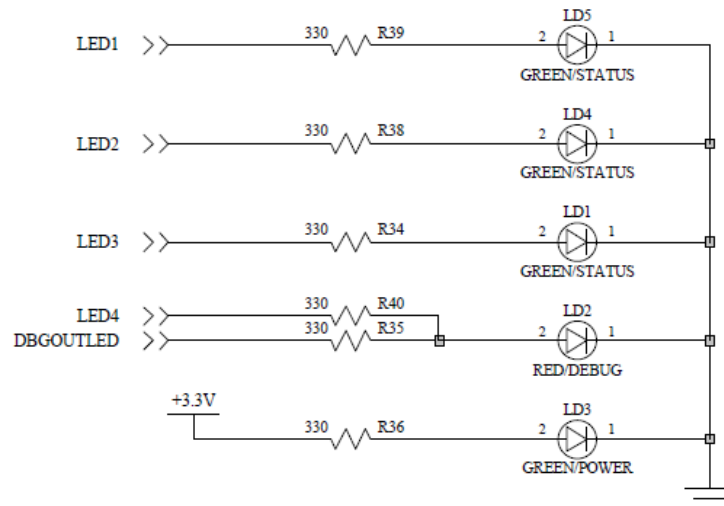
- 파일 복사를 마치게되면 LED 폴더에도 default_files에 있는 내용이 그대로 추가되게 된다.
 - 프로젝트 설정 마지막 단계로 소스 파일을 생성하도록 한다.
 - (led.c, led.h, main.c 등...) 필요한 파일을 만들도록 한다.



5. LED 제어 프로그램 추가하기

■ LED 회로도

LED



데이터 시트를 살펴보면 LED1은 GPIOC의 4번핀, LED2은 GPIOC의 6번핀, LED3은 GPIOE의 2번핀, LED4은 GPIOE의 3번핀에 맞추어져 있다.

PC4_PHA0	R109	0/NC	LED1
PC6_PHB0	R110	0/NC	LED2
PE2_PHB1	R111	0/NC	LED3
PE3_PHA1	R112	0/NC	LED4
PD0_CAN0RX	R116	0/NC	PF0_PLD
PD1_CAN0TX	R155	0/NC	PB0_PLD
PD5	R156	0/NC	PB1_PLD
PD6_FAULT	R157	0/NC	PD0_PLD
PG0	R158	0/NC	PD1_PLD

5. LED 제어 프로그램 추가하기

■ LED 실습

- 실습에 사용될 API 함수에 대해서 알아보자.
- 다음에 3개의 API 함수만 알면 쉽게 LED를 제어할 수 있다.
- 제어 함수
 - void SysCtlPeripheralEnable(unsigned long ulPeripheral);
 - 주변장치에 클럭을 제공해 줄 수 있게 만들어 주는 함수이다.
 - ulPeripheral: 클럭을 제공할 주변장치
 - sysctl.h 참고
 - void GPIOPinTypeGPIOOutput(unsigned long ulPort, unsigned char ucPins);
 - GPIO의 해당 포트에 핀을 출력상태로 결정하게 만들어 주는 함수이다.
 - ulPort: GPIO Port
 - ucPins: GPIO Pin
 - gpio.h 참고
 - void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins, unsigned char ucVal);
 - GPIO의 해당 포트에 핀을 값을 제공해 주는 함수이다.
 - ulPort: GPIO Port
 - ucPins: GPIO Pin
 - ucVal: GPIO 값
 - gpio.h 참고

5. LED 제어 프로그램 추가하기

■ led.h

```
#include "lm3s8962.h"
#include "hw_types.h"
#include "hw_memmap.h"
#include "sysctl.h"
#include "gpio.h"

#ifndef __LED_H__
#define __LED_H__

#define ON (1)
#define OFF (0)

#define SYS_CTL_LED12 SYSCTL_PERIPH_GPIOC
#define GPIO_BASE_LED12 GPIO_PORTC_BASE
#define LED1 GPIO_PIN_4
#define LED2 GPIO_PIN_6
```

5. LED 제어 프로그램 추가하기

■ led.h

```
#define SYS_CTL_LED34          SYSCTL_PERIPH_GPIOE
#define GPIO_BASE_LED34       GPIO_PORTE_BASE
#define LED3                   GPIO_PIN_2
#define LED4                   GPIO_PIN_3

#define LED1_BIT_NUM           (4)
#define LED2_BIT_NUM           (6)
#define LED3_BIT_NUM           (2)
#define LED4_BIT_NUM           (3)

void Led_Port_Init(void);
void WritePin(unsigned long PortBase, unsigned long Pin, int On_Off, int PinBitNum);
void Led_On(void);
void Led_Off(void);
void delay(int time);

#endif
```


5. LED 제어 프로그램 추가하기

■ led.c

```
#include "led.h"

void Led_Port_Init(void)
{
    SysCtlPeripheralEnable(SYS_CTL_LED12);
    GPIOPinTypeGPIOOutput(GPIO_BASE_LED12, LED1 | LED2);

    SysCtlPeripheralEnable(SYS_CTL_LED34);
    GPIOPinTypeGPIOOutput(GPIO_BASE_LED34, LED3 | LED4);
}

void WritePin(unsigned long PortBase, unsigned long Pin, int On_Off, int PinBitNum)
{
    GPIOPinWrite(PortBase, Pin, (On_Off << PinBitNum));
}

void Led_On(void)
{
    WritePin(GPIO_BASE_LED12, LED1, ON, LED1_BIT_NUM);
    WritePin(GPIO_BASE_LED12, LED2, ON, LED2_BIT_NUM);
    WritePin(GPIO_BASE_LED34, LED3, ON, LED3_BIT_NUM);
    WritePin(GPIO_BASE_LED34, LED4, ON, LED4_BIT_NUM);
}
```

5. LED 제어 프로그램 추가하기

■ led.c

- Led_Port_Init: Led 포트 초기화 함수
- WritePin: Led Port에 직접적으로 값을 입력하기 위한 함수
- Led_On: 4개의 LED가 동시에 on 시키는 함수
- Led_Off: 4개의 LED를 동시에 off 시키는 함수
- delay: 동작을 확인하기 위한 CPU 점유 딜레이

```
void Led_Off(void)
{
    WritePin(GPIO_BASE_LED12, LED1, OFF, LED1_BIT_NUM);
    WritePin(GPIO_BASE_LED12, LED2, OFF, LED2_BIT_NUM);
    WritePin(GPIO_BASE_LED34, LED3, OFF, LED3_BIT_NUM);
    WritePin(GPIO_BASE_LED34, LED4, OFF, LED4_BIT_NUM);
}

void delay(int time)
{
    int i;
    for(i = 0; i < time; i++);
}
```

5. LED 제어 프로그램 추가하기

■ main.c

```
#include "led.h"

int main(void)
{
    Led_Port_Init();

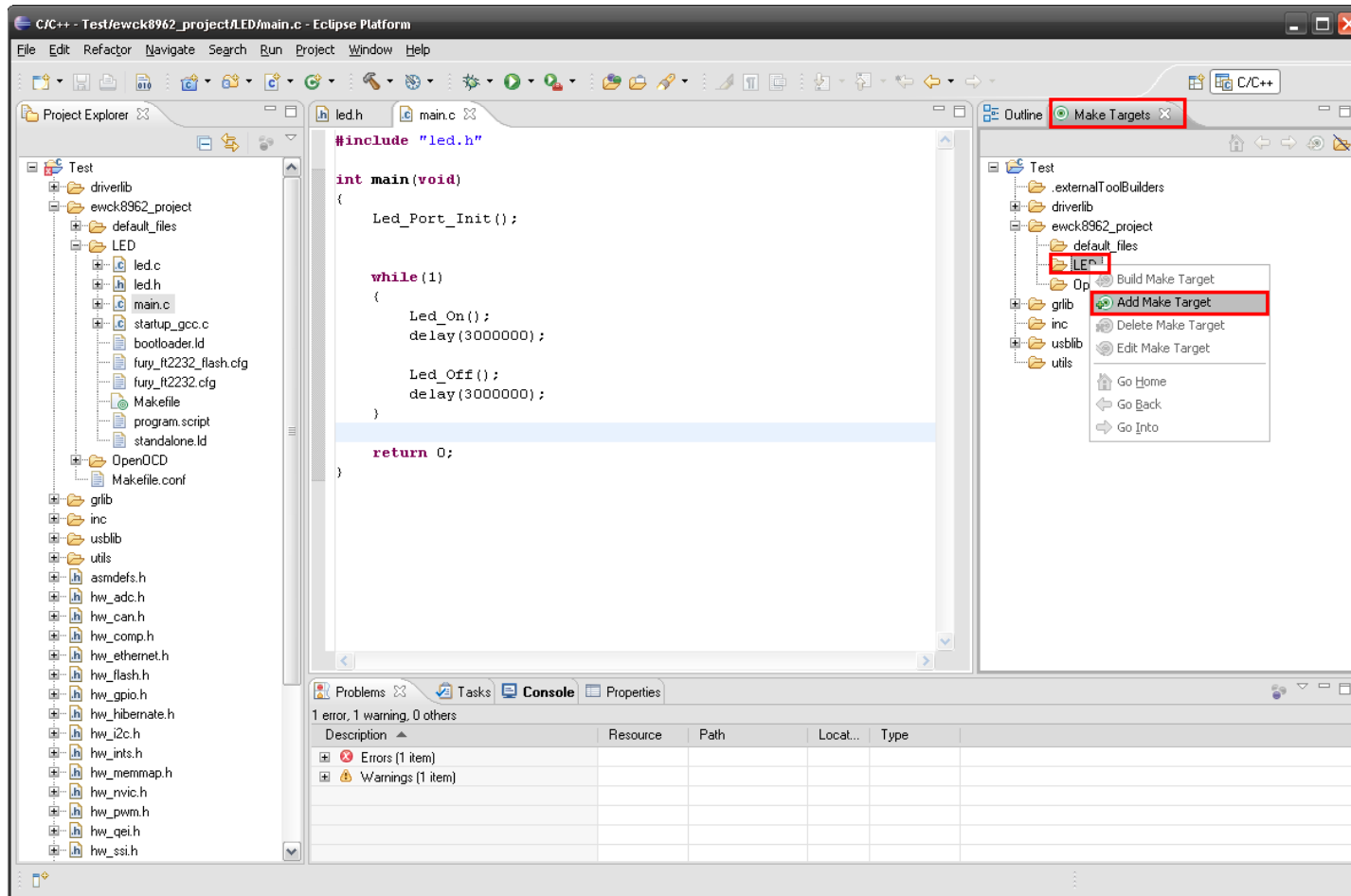
    while(1)
    {
        Led_On();
        delay(3000000);

        Led_Off();
        delay(3000000);
    }

    return 0;
}
```

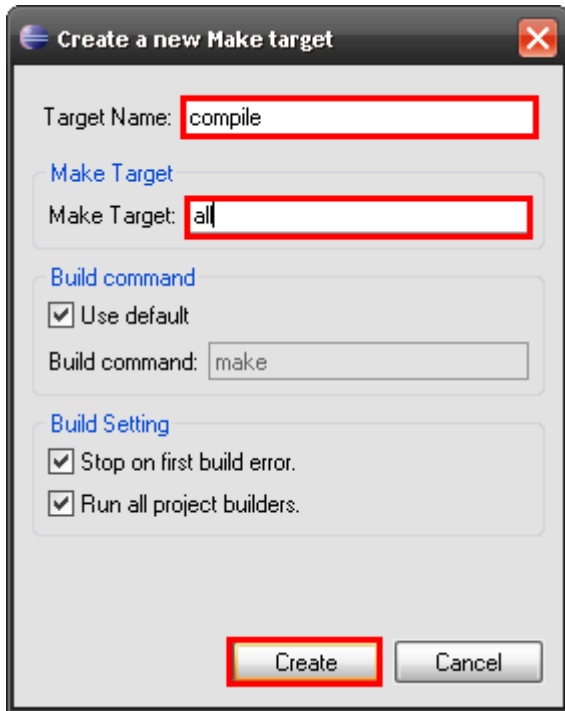
5. LED 제어 프로그램 추가하기

- 파일 추가와 소스 코드의 작성을 마쳤다면 컴파일 과정을 위한 설정 작업을 진행하도록 한다.



5. LED 제어 프로그램 추가하기

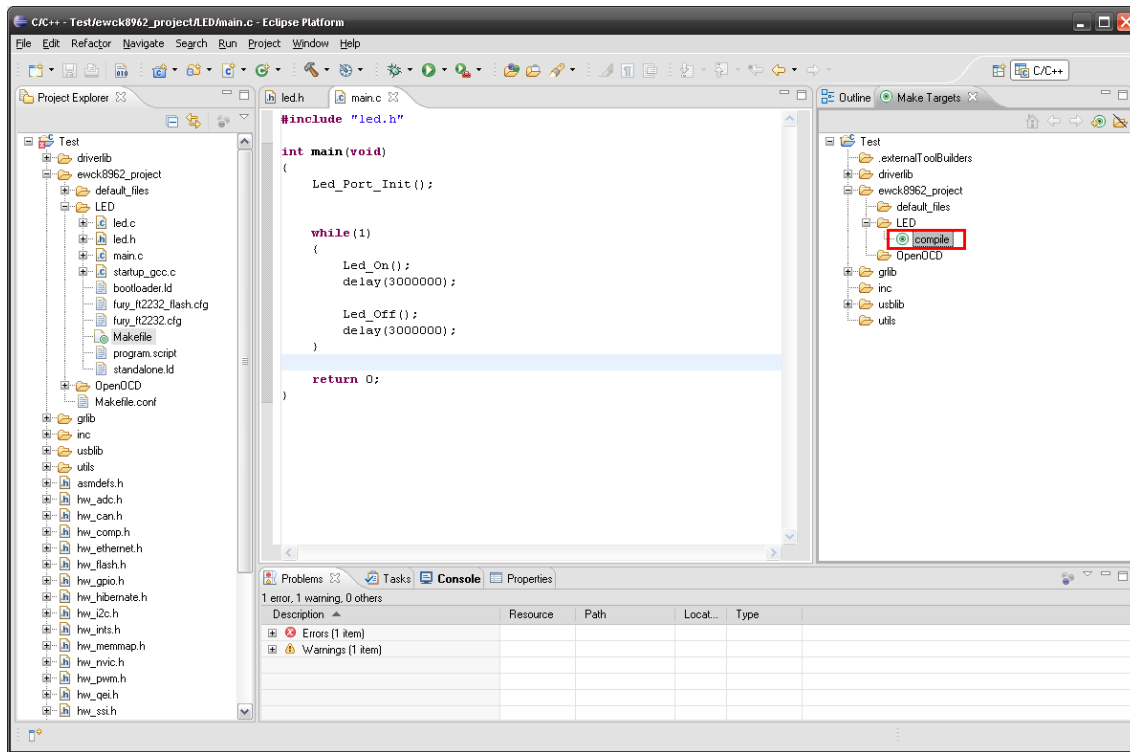
- 파일 추가와 소스 코드의 작성을 마쳤다면 컴파일 과정을 위한 설정 작업을 진행하도록 한다.



- Target Name
추가 할 항목 이름
- Make Target
Makefile에 작성되어진 명령 항목
- Build Command
Make 유틸리티 이름
(변경 할 필요 없음.)

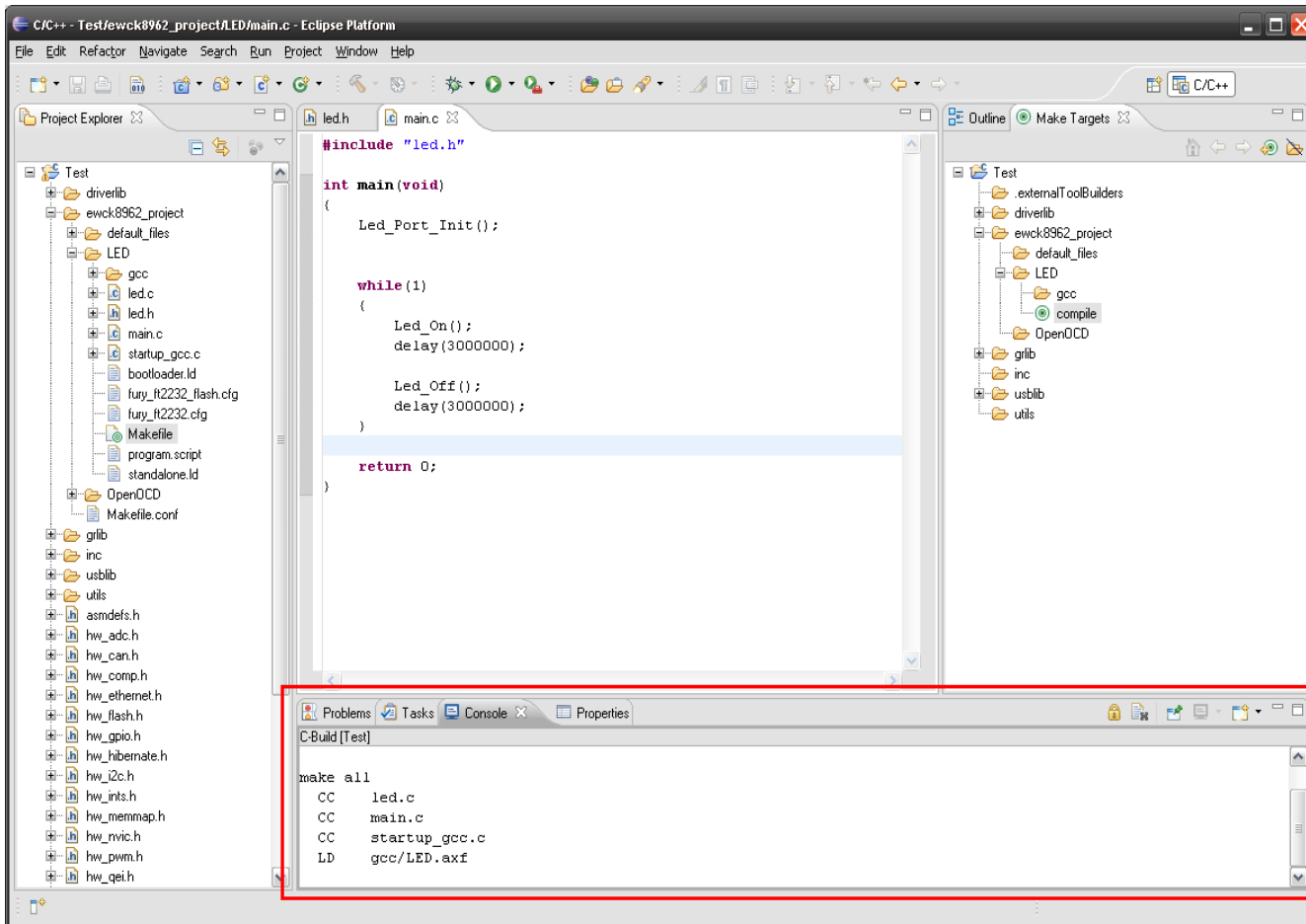
5. LED 제어 프로그램 추가하기

- 파일 추가와 소스 코드의 작성을 마쳤다면 컴파일 과정을 위한 설정 작업을 진행하도록 한다.
- compile 항목이 Make 메뉴에 추가가 되어진 것을 확인할 수 있다.



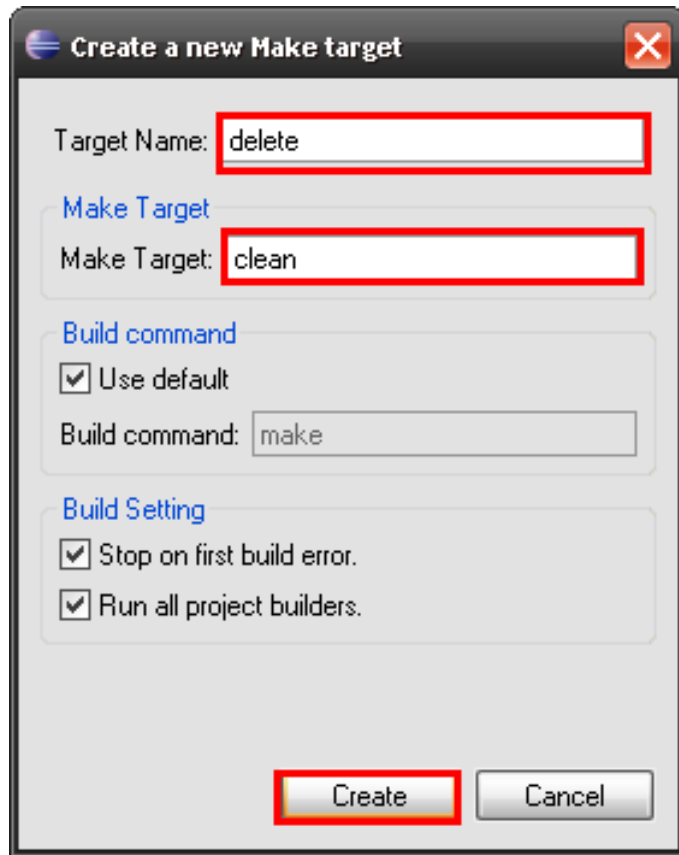
5. LED 제어 프로그램 추가하기

- 파일 추가와 소스 코드의 작성을 마쳤다면 컴파일 과정을 위한 설정 작업을 진행하도록 한다.
- 추가되어진 compile를 더블 클릭하게 되면 컴파일이 되어지는 것을 확인할 수 있다.



5. LED 제어 프로그램 추가하기

- 위와 동일한 방법으로 컴파일 내용물을 지우는 항목과 보드에 업로드하는 항목을 추가하도록 한다.



The screenshot shows the 'Create a new Make target' dialog box. The 'Target Name' field contains 'delete' and the 'Make Target' field contains 'clean'. Both fields are highlighted with red rectangles. The 'Build command' section has 'Use default' checked and 'make' in the text box. The 'Build Setting' section has 'Stop on first build error.' and 'Run all project builders.' both checked. The 'Create' button at the bottom is highlighted with a red rectangle.

Create a new Make target

Target Name: delete

Make Target

Make Target: clean

Build command

☒ Use default

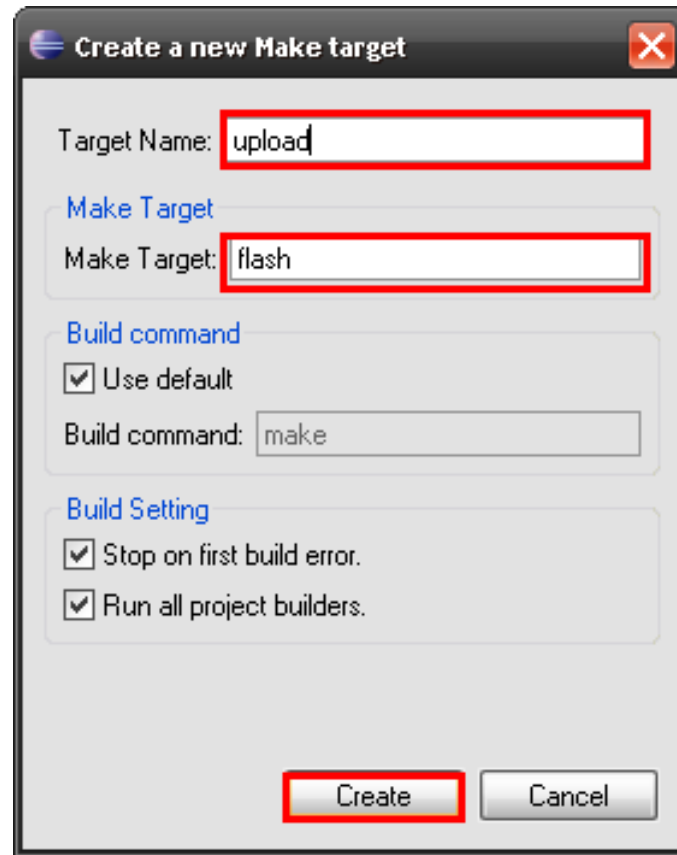
Build command: make

Build Setting

☒ Stop on first build error.

☒ Run all project builders.

Create Cancel



The screenshot shows the 'Create a new Make target' dialog box. The 'Target Name' field contains 'upload' and the 'Make Target' field contains 'flash'. Both fields are highlighted with red rectangles. The 'Build command' section has 'Use default' checked and 'make' in the text box. The 'Build Setting' section has 'Stop on first build error.' and 'Run all project builders.' both checked. The 'Create' button at the bottom is highlighted with a red rectangle.

Create a new Make target

Target Name: upload

Make Target

Make Target: flash

Build command

☒ Use default

Build command: make

Build Setting

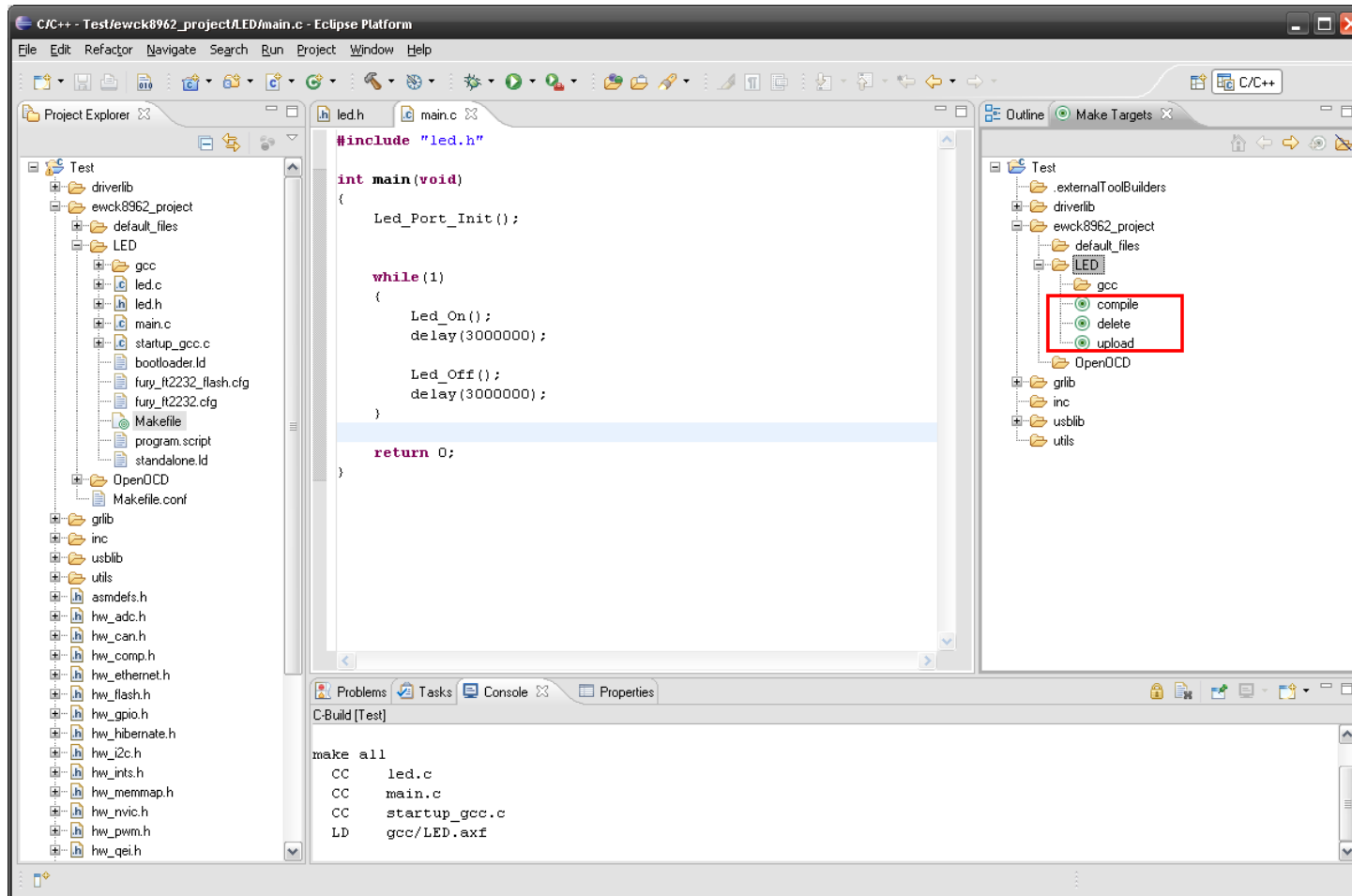
☒ Stop on first build error.

☒ Run all project builders.

Create Cancel

5. LED 제어 프로그램 추가하기

- 위와 동일한 방법으로 컴파일 내용물을 지우는 항목과 보드에 업로드하는 항목을 추가하도록 한다.



5. LED 제어 프로그램 추가하기

■ Makefile

The screenshot displays the Eclipse IDE interface for a C/C++ project named "Test/ewck8962_project/LED/Makefile". The main editor shows the content of the Makefile, which includes the following targets and rules:

```
INC_PATHS = . $(ROOT) $(ROOT)/inc $(ROOT)/driverlib $(EXTDIR) ../include
CFLAGS += $(addprefix -I , $(INC_PATHS))
LDFLAGS += -Map $(COMPILER)/$(TARGET_NAME).map
vpath %.c $(EXTDIR)

#
# The default rule, which causes example to be built.
#
all: $(COMPILER) $(COMPILER)/$(TARGET)

#
# The rule to clean out all the build products.
#
clean:
    rm -rf $(COMPILER)
    rm -rf $(wildcard ~/.*) $(wildcard *.bak) $(wildcard *.dep)

flash: clean all
    @echo -----
    @echo Flash programming...
    @../OpenOCD/OpenOCD_ftdi.exe --file fury_ft2232_flash.cfg

#
# The rule to create the target directory.
#
$(COMPILER):
    @mkdir $(COMPILER)
```

The Project Explorer on the left shows the project structure, including the "LED" directory with files like gcc, led.c, led.h, main.c, startup_gcc.c, bootloader.ld, fury_ft2232.c, fury_ft2232.h, Makefile, program.scrip, and standalone.k. The Make Targets view on the right shows the "LED" target with sub-targets like gcc, compile, delete, and upload. The Console at the bottom shows the output of the "make all" command:

```
C-Build [Test]
make all
CC led.c
CC main.c
CC startup_gcc.c
```

The status bar at the bottom indicates the file is "Writable", "Smart Insert" is enabled, and the cursor is at line 66, column 7.

5. LED 제어 프로그램 추가하기

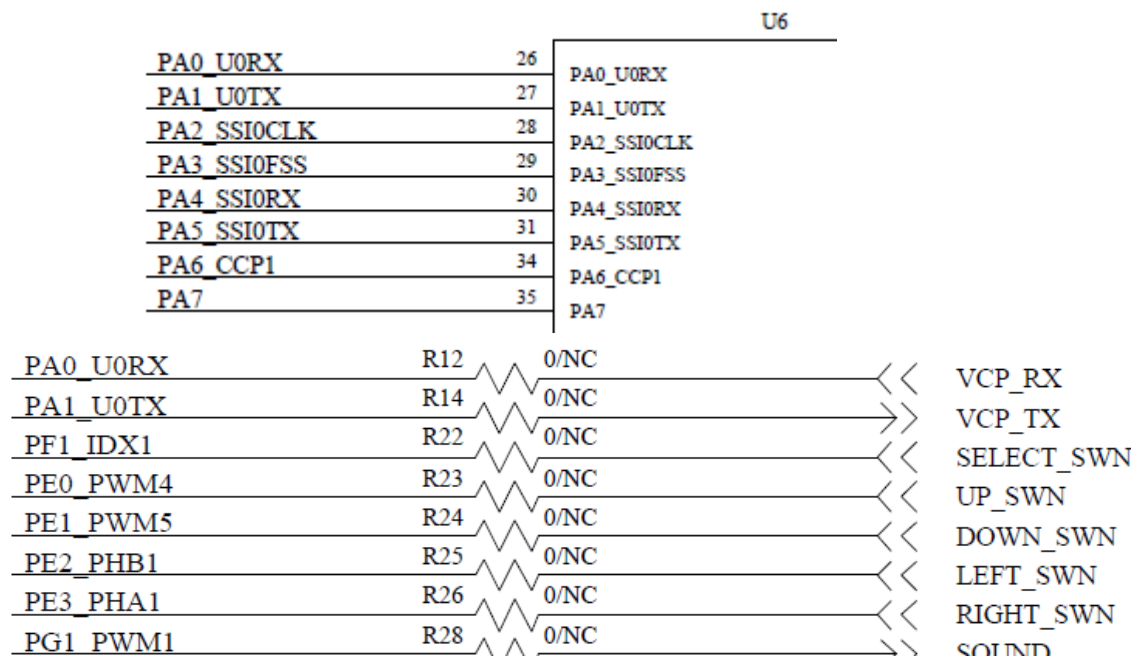
■ Makefile

- 위와 같이 Make 항목을 추가하기 위해서는 Makefile에 위와 같이 Make Target에 들어갈 항목들이 반드시 존재해야하며, 부가적으로 필요한 항목이 있다면 Makefile 작성법에 맞도록 추가하여 Make 항목을 추가하여야만 Make 유틸리티를 이용해 해당 내용을 실행할 수 있다는 점을 명심하도록 한다.

6. Uart 실습

■ Uart 제어

- 앞으로 좀 더 편한 프로그래밍을 하기 위해 이번 장에서 Uart 제어를 하도록 해 보자.
- 실습할 값들을 확인하거나 코드에 중간에 필요한 메시지를 출력하고 싶은 경우를 대비하여 UART를 이용하여 하이퍼터미널에 값을 출력하여 보다 쉽게 확인하기 위함이다.
- EWCK8962 Uart 회로도



6. Uart 실습

■ Uart 실습

■ 제어 함수

- void SysCtlPeripheralEnable(unsigned long ulPeripheral);
 - 주변장치에 클럭을 제공해 줄 수 있게 만들어 주는 함수이다.
 - ulPeripheral: 클럭을 제공할 주변장치
 - sysctl.h 참고
- void GPIOPinTypeUART(unsigned long ulPort, unsigned char ucPins)
 - GPIO의 해당 포트에 핀을 타입을 UART로 결정하는 함수이다.
 - ulPort: GPIO Port
 - ucPins: GPIO Pin
 - gpio.h 참고
- void UARTConfigSetExpClk(unsigned long ulBase, unsigned long ulUARTClk, unsigned long ulBaud, unsigned long ulConfig)
 - Uart 채널에 클럭을 설정하고 초기하는 함수
 - ulBase: Uart의 Base 번지 주소
 - ulUARTClk: Uart에 제공할 클럭
 - ulBaud: Baud Rate 값
 - ulConfig: Uart 프p임을 설정
 - uart.h 참고

6. Uart 실습

■ Uart 실습

■ 제어 함수

- long UARTCharGetNonBlocking(unsigned long ulBase)
 - Uart에 FIFO에 Char 값을 넣어주는 함수
 - ulBase: Uart의 Base 번지 주소
 - 리턴값: FIFO의 상태가 FULL인지 확인해 주는 값
 - » 0 : FULL
 - uart.h 참고

6. Uart 실습

■ uart_test.h

```
#include "lm3s8962.h"
#include "hw_types.h"
#include "hw_memmap.h"
#include "sysctl.h"
#include "gpio.h"
#include "uart.h"
#include "string.h"

#ifndef __UART_TEST_H__
#define __UART_TEST_H__

void Clock_Init(void);
void UART_Init(void);
void UART_Send(const char *pucBuffer);
void delay(int time);

#endif
```

6. Uart 실습

■ uart_test.c

```
#include "uart_test.h"

void Clock_Init(void){
    SysCtlClockSet(SYSCTL_SYSDIV_1 |
        SYSCTL_USE_OSC |
        SYSCTL_OSC_MAIN |
        SYSCTL_XTAL_8MHZ);
}

void UART_Init(void){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinTypeUART(GPIO_PORTA_BASE,
        GPIO_PIN_0 | GPIO_PIN_1);
    UARTConfigSetExpClk(UART0_BASE,
        SysCtlClockGet(),
        115200,
        (UART_CONFIG_WLEN_8 |
        UART_CONFIG_STOP_ONE |
        UART_CONFIG_PAR_NONE));
}
```


6. Uart 실습

■ uart_test.c

- **uart_test.c** 파일에는 **uart**를 초기화하고 **uart 0번 채널**을 통해서 **하이퍼터미널**에 데이터를 출력할 수 있도록 **API함수**들을 이용하여 작성되어 있다. 작성되어 있는 함수의 설명은 아래와 같다.
 - **Clock_Init**: 보드에 클럭을 제공해 주는 함수
 - **UART_Init**: Uart 0번 채널을 초기화 시키는 함수
 - **UART_Send**: Uart를 통해서 하이터터미널에 문자열을 출력 시켜 주는 함수
 - **delay**: 동작을 확인하기 위한 CPU 점유 딜레이

```
void UART_Send(const char *pucBuffer){
    int strLength = strlen(pucBuffer);

    while(strLength){
        UARTCharPutNonBlocking(UART0_BASE, *pucBuffer++);
        strLength--;
    }
}

void delay(int time)
{
    int i;
    for(i = 0; i < time; i++);
}
```

6. Uart 실습

■ main.c

- **uart_test.c** 파일에는 **uart**를 초기화하고 **uart 0번 채널**을 통해서 하이퍼터미널에 데이터를 출력할 수 있도록 **API함수들을 이용하여 작성되어 있다**. 작성되어 있는 함수의 설명은 아래와 같다.
 - **Clock_Init**: 보드에 클럭을 제공해 주는 함수
 - **UART_Init**: Uart 0번 채널을 초기화 시키는 함수
 - **UART_Send**: Uart를 통해서 하이터터미널에 문자열을 출력 시켜 주는 함수
 - **delay**: 동작을 확인하기 위한 CPU 점유 딜레이

```
#include "uart_test.h"

int main(void){
    Clock_Init();
    UART_Init();

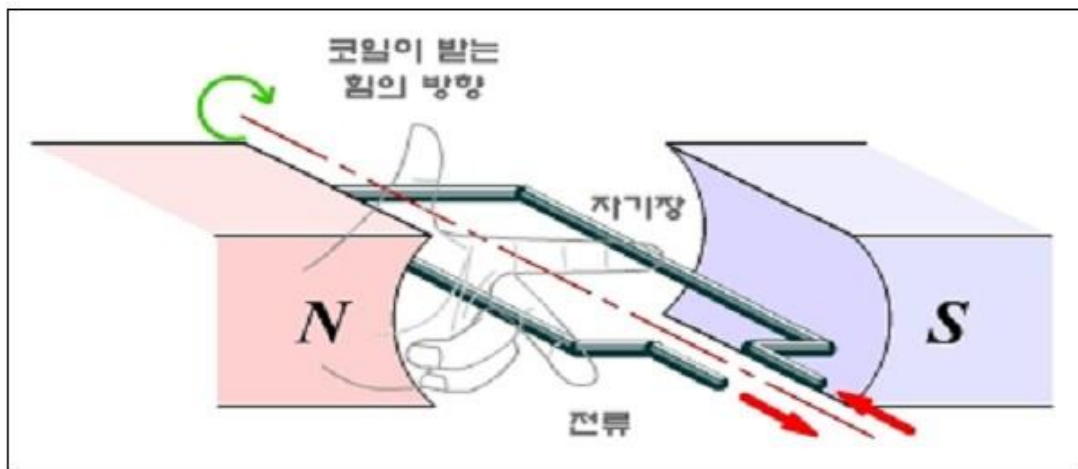
    while(1){
        UART_Send("Uart Test!!\n");
        delay(1000000);
    }

    return 0;
}
```

7. DC 모터 실습

■ DC 모터 제어

- DC 모터는 자기장 중에 놓은 도체를 직류 전류를 흘리면 플레밍의 왼손 법칙에 의해 도체에 전자력이 발생하여 회전하게 되는 원리를 이용하여 작동하게 된다.

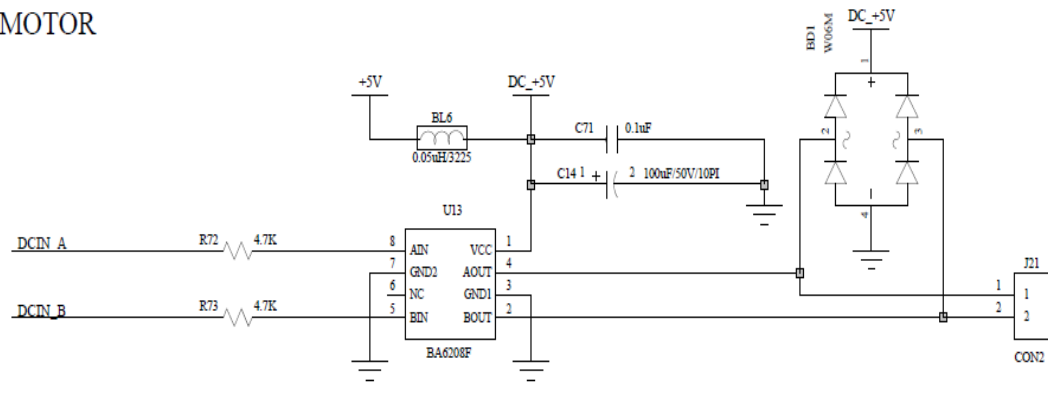


- EWCK에서는 DC모터를 제어하기 위한 2개의 핀을 GPIO포트에 연결시켜 두었고 있다.
- 이 2개의 포트 핀은 각각 PORTD에 7번 핀과 PORTF의 1번 핀이면 이 포트를 제어하기 위해서는 LED를 제어할 때 사용된 GPIO API 함수를 통해 쉽게 제어 할 수 있다.
- DC 모터를 구동하는 방법은 플레밍의 왼손 법칙의 원리 처럼 회전 축에 전류를 공급하면 가능해진다.
- 예를 들어 EWCK에서는 제공하는 2개에 핀 중 PORTD에 7번 핀의 값을 1로 설정하고 PORTF의 1번 핀의 값을 0으로 설정하였을 경우 전류가 PORTD에 7번 핀에서 PORTF의 1번핀 방향으로 흐리게 되어 정방향으로 회전하게 된다.
- 이와 반대로 모터를 역방향으로 회전 시키기 위해서는 PORTD에 7번 핀과 PORTF의 1번 핀의 값을 반대로 설정하게 된다.

7. DC 모터 실습

EWCK8962 DC MOTOR 회로도

DC MOTOR



ADC0	R105	0/NC	CDS
PD7_IDX0	R21	0/NC	DCIN_A
PF1_IDX1	R27	0/NC	DCIN_B
PB4_C0-	R106	0/NC	STEP_DISABLE
PB5_C0O	R107	0/NC	STEP_DIR
PB6_C0+	R108	0/NC	STEP_CLK
PF0_PWM0	R113	0/NC	STEP_SYNC
PB0_PWM2	R114	0/NC	STEP_M1
PB1_PWM3	R115	0/NC	STEP_M0

7. DC 모터 실습

■ DC 모터 실습

▪ 제어 함수

- void SysCtlPeripheralEnable(unsigned long ulPeripheral);
 - 주변장치에 클럭을 제공해 줄 수 있게 만들어 주는 함수이다.
 - ulPeripheral: 클럭을 제공할 주변장치
 - sysctl.h 참고
- void GPIOPinTypeGPIOOutput(unsigned long ulPort, unsigned char ucPins);
 - GPIO의 해당 포트에 핀을 출력상태로 결정하게 만들어 주는 함수이다.
 - ulPort: GPIO Port
 - ucPins: GPIO Pin
 - gpio.h 참고
- void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins, unsigned char ucVal);
 - GPIO의 해당 포트에 핀을 값을 제공해 주는 함수이다.
 - ulPort: GPIO Port
 - ucPins: GPIO Pin
 - ucVal: GPIO 값
 - gpio.h 참고

7. DC 모터 실습

■ dcmotor.h

```
#include "lm3s8962.h"
#include "hw_types.h"
#include "hw_memmap.h"
#include "sysctl.h"
#include "gpio.h"

#ifndef __DCMOTOR_H__
#define __DCMOTOR_H__

#define SYSCTL_PERIPH_DCA    SYSCTL_PERIPH_GPIOD
#define GPIO_PORT_BASE_DCA  GPIO_PORTD_BASE
#define GPIO_PIN_DCA        GPIO_PIN_7

#define SYSCTL_PERIPH_DCB    SYSCTL_PERIPH_GPIOF
#define GPIO_PORT_BASE_DCB  GPIO_PORTF_BASE
#define GPIO_PIN_DCB        GPIO_PIN_1
```

7. DC 모터 실습

■ dcmotor.h

```
void Delay(int count);  
void Clock_Init(void);  
void DcMotor_Init(void);  
void DcMotor_Front_On(void);  
void DcMotor_Back_On(void);  
void DcMotor_Off(void);
```

```
#endif
```

7. DC 모터 실습

■ dcmotor.c

```
#include "dcmotor.h"

void Delay(int count){
    int i;

    for(i = 0; i < count; i++);
}

void Clock_Init(void){
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
        SYSCTL_XTAL_8MHZ);
}

void DcMotor_Init(void){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_DCA| SYSCTL_PERIPH_DCB);
    GPIOPinTypeGPIOOutput(GPIO_PORT_BASE_DCA, GPIO_PIN_DCA);
    GPIOPinTypeGPIOOutput(GPIO_PORT_BASE_DCB, GPIO_PIN_DCB);
}
```


7. DC 모터 실습

■ dcmotor.c

```
void DcMotor_Front_On(void){
    GPIOWrite(GPIO_PORT_BASE_DCA, GPIO_PIN_DCA, GPIO_PIN_DCA);
    GPIOWrite(GPIO_PORT_BASE_DCB, GPIO_PIN_DCB, 0);
}
void DcMotor_Back_On(void){
    GPIOWrite(GPIO_PORT_BASE_DCB, GPIO_PIN_DCB, GPIO_PIN_DCB);
    GPIOWrite(GPIO_PORT_BASE_DCA, GPIO_PIN_DCA, 0);
}

void DcMotor_Off(void){
    GPIOWrite(GPIO_PORT_BASE_DCA, GPIO_PIN_DCA, 0);
    GPIOWrite(GPIO_PORT_BASE_DCB, GPIO_PIN_DCB, 0);
}
```

7. DC 모터 실습

■ dcmotor.c

- motor.c 파일에는 직접적으로 DC 모터를 제어하기 위해 API함수들을 이용하여 작성되어 있는 코드들이 있다.
- 작성되어 있는 함수의 설명은 아래와 같다.
 - Clock_Init: 보드에 클럭을 제공해 주는 함수
 - DcMotor_Init: DC 모터를 초기화 시켜 주는 함수
 - DcMotor_Front_On: DC 모터를 정방향으로 작동시키는 함수
 - DcMotor_Back_On: DC 모터를 역방향으로 작동시키는 함수
 - DcMotor_Off: DC 모터를 정지 시키는 함수
 - Delay: 동작을 확인하기 위한 CPU 점유 딜레이

7. DC 모터 실습

■ main.c

```
#include "dcmotor.h"

int main(void){

    Clock_Init();
    DcMotor_Init();

    while(1){
        DcMotor_Front_On();
        Delay(4000000);
        DcMotor_Back_On();
        Delay(4000000);
        DcMotor_Off();
        Delay(4000000);
    }

    return 0;
}
```



Thank You
