1.6 실습(STM32CubeIDE)

이 책의 실습에서는 STMicroelectronics 사에서 공급하는 Nucleo-103RB 보드를 사용한다. 이 보드는 비교적 가격이 저렴하며, 국내에서 쉽게 구입이 가능하다. STMicroelectronics 사는 Arm 마이크로컨트롤러를 제조해서 판매하는 반도체 제조 회사들 중의 하나이며, 자사에서 제조하는 마이크로컨트롤러를 시험해 볼 수 있는 평가용 보드를 여러 가지 만들어서 저렴한 가격에 판매하고 있다. 그 보드들 중에서 이 책에서는 Cortex-M3 STM32F103RB 마이크로컨트롤러를 사용한 Nucleo-103RB 보드를 사용하며 그림 1.10은 보드의 형태를 보여준다.

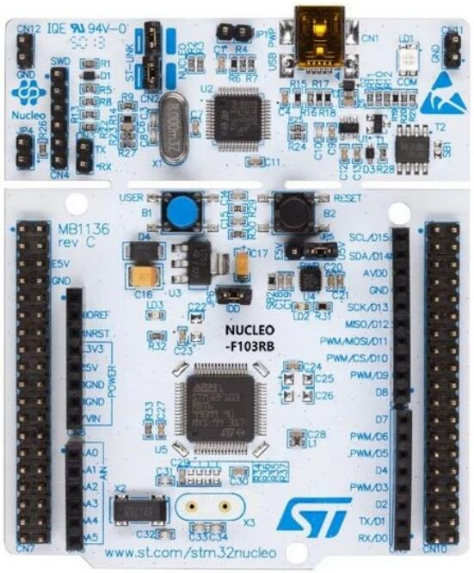

그림 1.10 Nucleo-103RB 보드

통상적으로 마이크로컨트롤러를 프로그램하기 위해서는 개발용 컴퓨터와 통신할 수 있는 디버거 인터페이스가 필요하다. STMicroelectronics 사에서 제조해서 판매하는 평가용 보드들은 이러한 디버거 인터페이스를 포함하고 있으므로, 시험을 위한 준비 비용을 절약할 수 있다. 그림 1.10의 보드에서 보면 2개의 마이크로컨트롤러가 탑재 되어 있으며, 보드가 2부분으로 나뉜 것을 볼 수 있다. 이 보드에서 미니 USB 커넥터가 붙어 있는 상단 부분이 디버거 인터페이스이며, 이 책에서 프로그래밍 방법을 학습하게 될 STM32F103RB는 보드의 아래 부분에 장착되어 있다. 또한, 이 보드의 양쪽에는 마이크컨트롤러의 핀들과 연결된 커넥터가 양측에 배치되어 있어서, 브레드 보드를 이용해서 주변 회로를 쉽게 구성할 수 있다. 또한, 초보자들이 많이 사용하는 아두이노(Arduino) 프로그래밍을 위해서 아두이노와 호환이 되는 커넥터도 있으나, 이 책에서는 아두이노 프로그램을 다루지 않으므로 아두이노 커넥터들은 사용하지 않는다.

■ Arm Cortex-M 마이크로컨트롤러 개발 소프트웨어

현재 많이 사용되는 Cortex-M 마이크로컨트롤러 개발 소프트웨어에는 KEIL, IAR 등의 회사에서 제공하는 개발용 소프트웨어들이 있으며, 이들 소프트웨어들은 라이센스를 구매해서 사용하는 상용 소프트웨어이다. 한편, STMicroelectronics 사에서는 STM32CubeIDE라는 자사의 마이크로컨트롤

러 프로그래밍을 위한 소프트웨어를 무료로 공급하고 있다. STM32CubeIDE에는 소스 코드의 편집과 컴파일, 디버깅 등의 기능을 가지고 있는 통합 개발 소프트웨어이며, 마이크로컨트롤러에 내장된 주변 기기들의 레지스터들을 설정해주는 함수들의 소스 코드를 자동으로 생성해주는 기능을 가지고 있다. 이 실습에서는 STM32CubeIDE를 이용해서 각종 실습 예제들을 실행하는 예를 보여준다.

STM32CubeIDE는 st.com 사이트에서 다운로드 받을 수 있다. 먼저, st.com 사이트에 가입해서 로그인 한 후, STM32CubeIDE를 검색하면 다운로드 페이지를 쉽게 찾을 수 있다. 다운로드 페이지는 Google에서 검색해도 찾을 수 있다. 다운로드 페이지에서 자신이 사용하는 컴퓨터의 운영체제 종류(MS Windows 혹은 Linux)에 따라서 STM32CubeIDE 소프트웨어를 다운로드 받을 수 있다. 소프트웨어를 다운로드 받은 후 설치 프로그램을 실행하면 어렵지 않게 설치가 가능하다. 이 프로그램은 자바 기반으로 실행되므로 최신 버전의 자바 런타임의 설치를 미리 하는 것을 추천하다. STM32CubeIDE 프로그램을 설치한 후 실행하면 그림 1.11과 같이 Workspace의 저장 위치를 묻는 화면이 나온다. Workspace는 프로젝트를 저장하기 위한 폴더이며, 이 폴더의 이름과 위치는 원하는 이름과 위치로 변경이 가능하다. 만약 변경을 원하지 않는다면 Launch 버튼을 누르면 다음 단계로 진행된다.

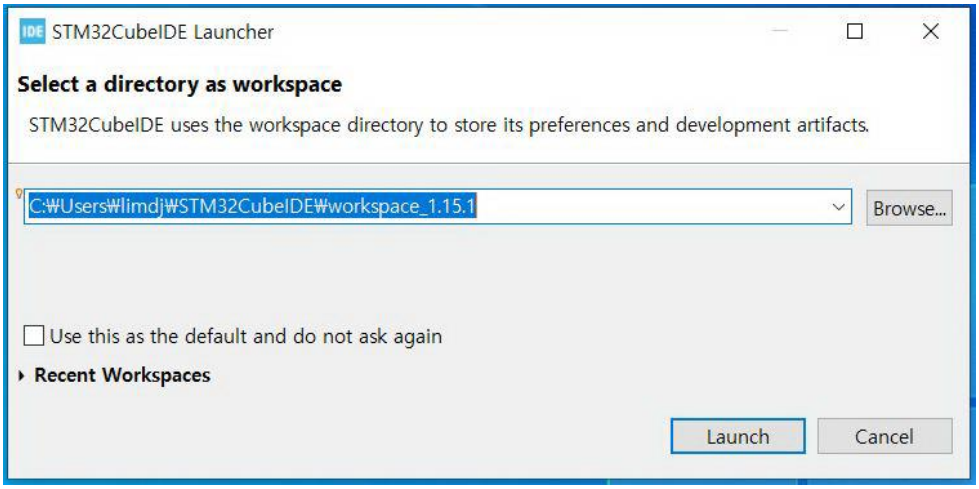

그림 1.11 Workspace의 선택

다음으로, 그림 1.12와 같은 첫 화면이 나오게 되며, Information Center 화면에서 첫 프로젝트를 시작할 수도 있다.

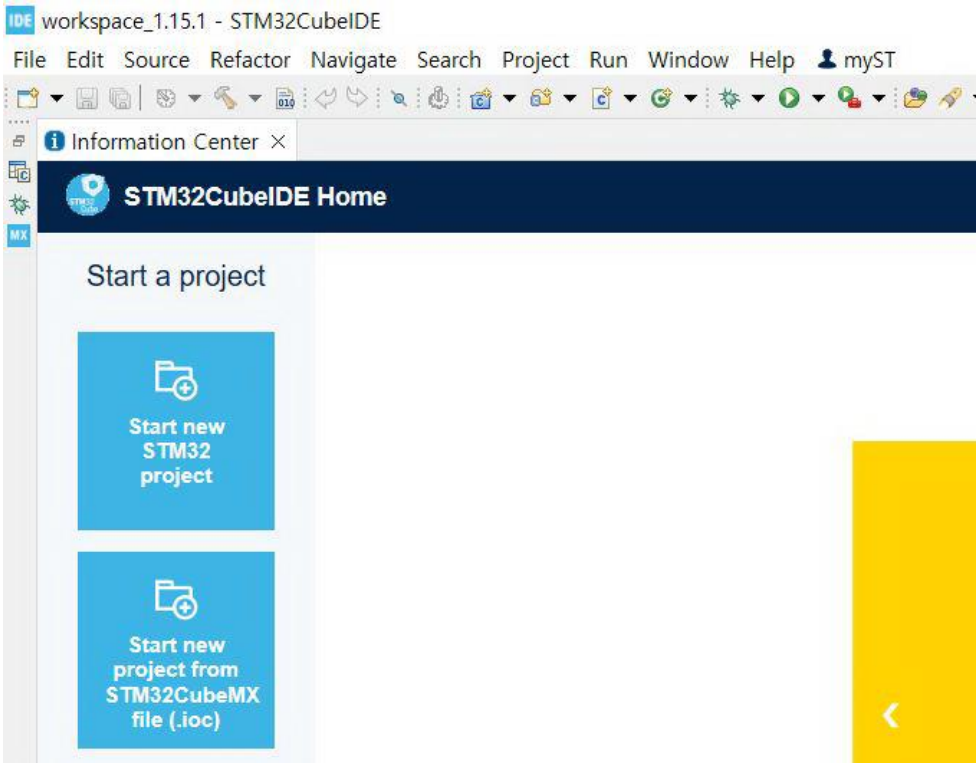

그림 1.12 Information Center 페이지

위와 같은 Information Center 페이지를 사용하기 원하지 않는다면 X 마크를 클릭해서 페이지를 제거할 수 있다. Information Center 페이지를 제거하면 그림 1.13과 같은 화면을 볼 수 있다.

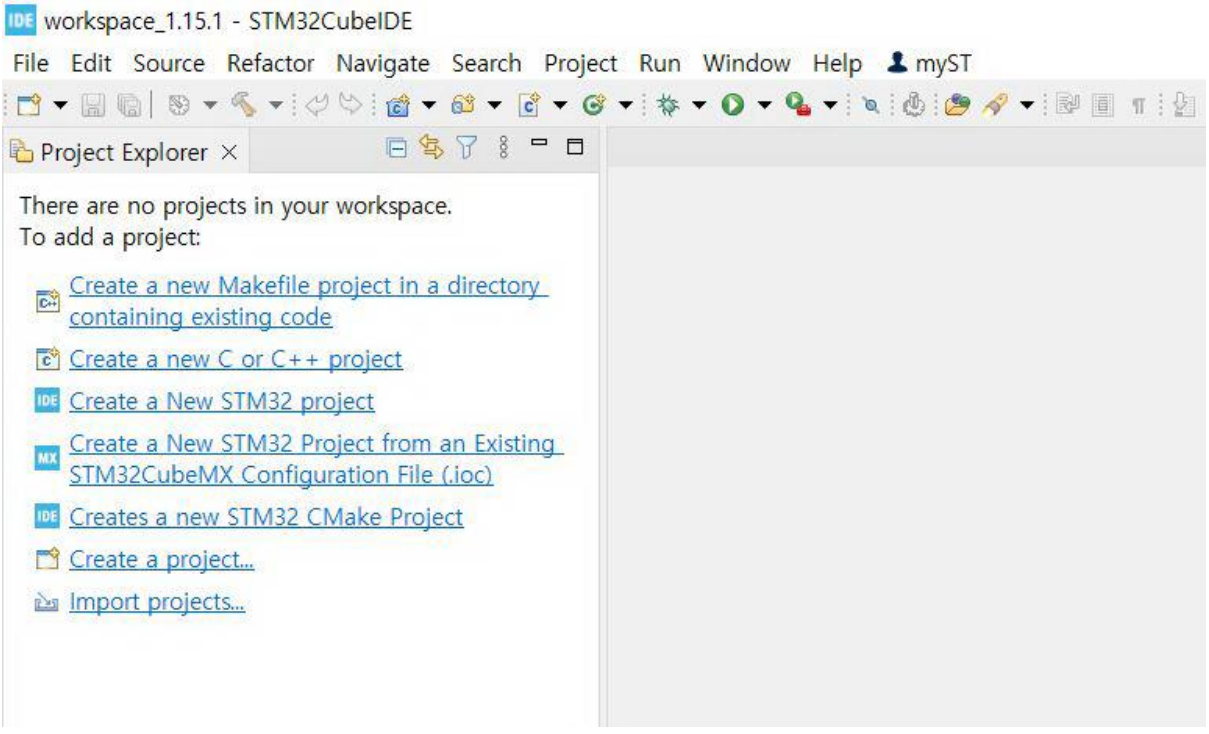

그림 1.13 시작 화면

그림 1.13의 화면에서 File 메뉴 탭을 클릭하면 그림 1.14와 같은 메뉴가 나오며, 여기에서 New 항목의 STM32 Project를 선택한다.

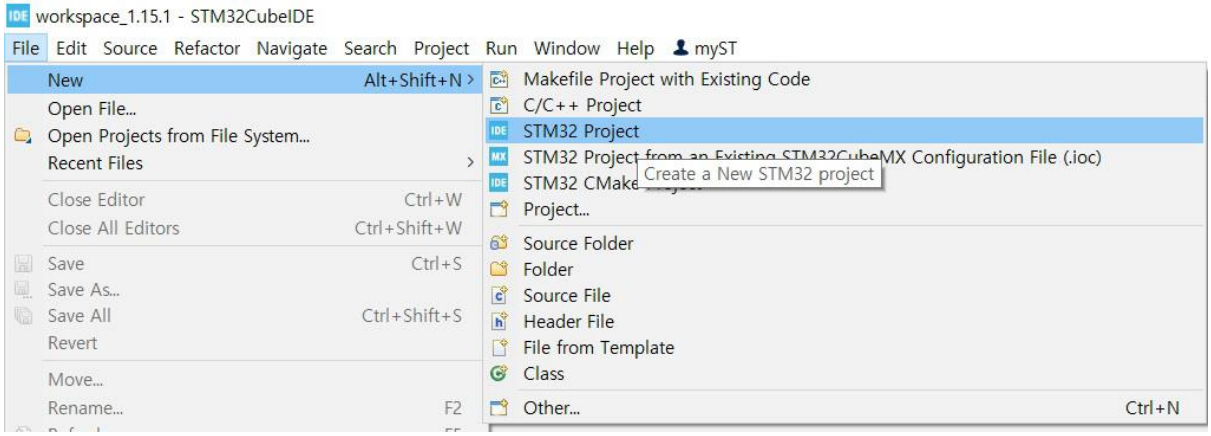

그림 1.14 New STM32 Project

다음, 그림 1.15와 같은 Target Selection 화면이 나오며, 이 화면에서 Board Selector 탭을 선택해서 Nucleo-64 Type을 선택한다. 우측에서는 여러가지의 보드가 나타나는데, 이중에서 NUCLEO-F103RB 보드를 선택하고 Next 버튼을 클릭한다.

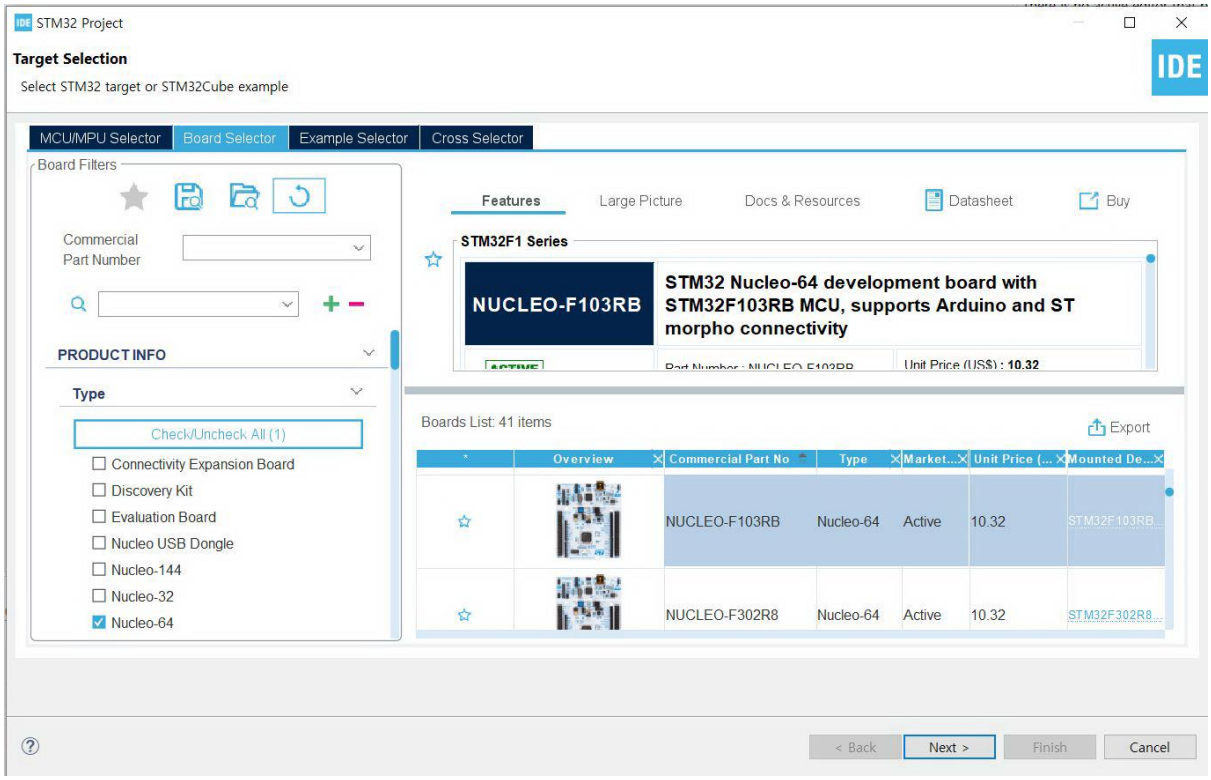

그림 1.15 Target Selection

다음, 그림 1.16의 프로젝트 설정 화면이 나오면 프로젝트의 이름을 입력한 후, Finish 버튼을 클릭한다.

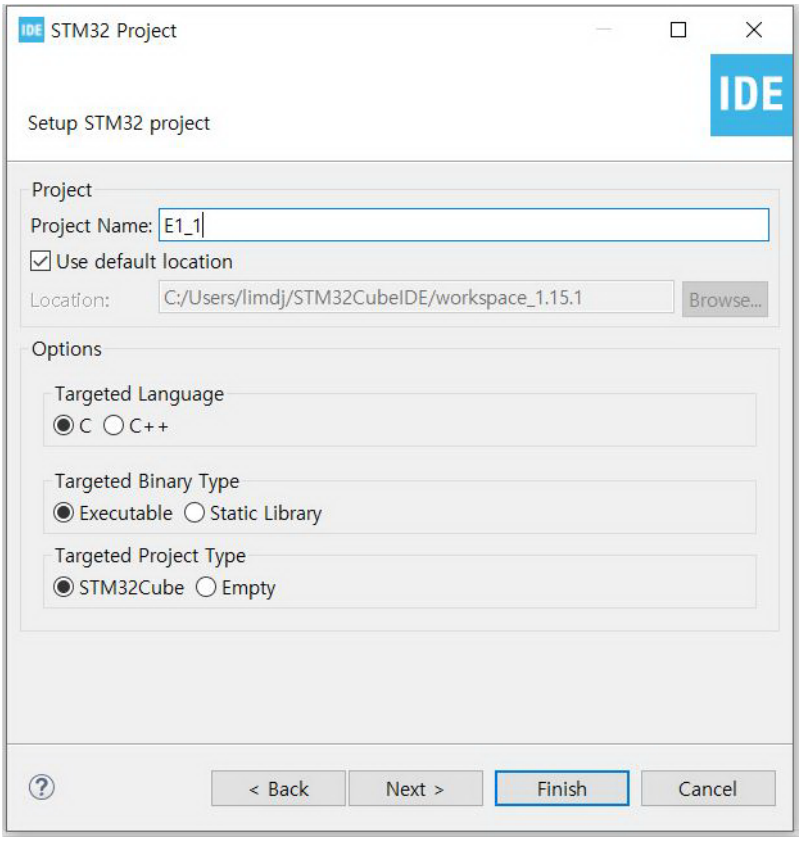

그림 1.16 프로젝트 설정

다음, 그림 1.17의 주변 장치의 설정에 대한 질문이 나오면 YES를 클릭한다. 주변 장치 설정을 기본 값으로 한다는 의미이다.

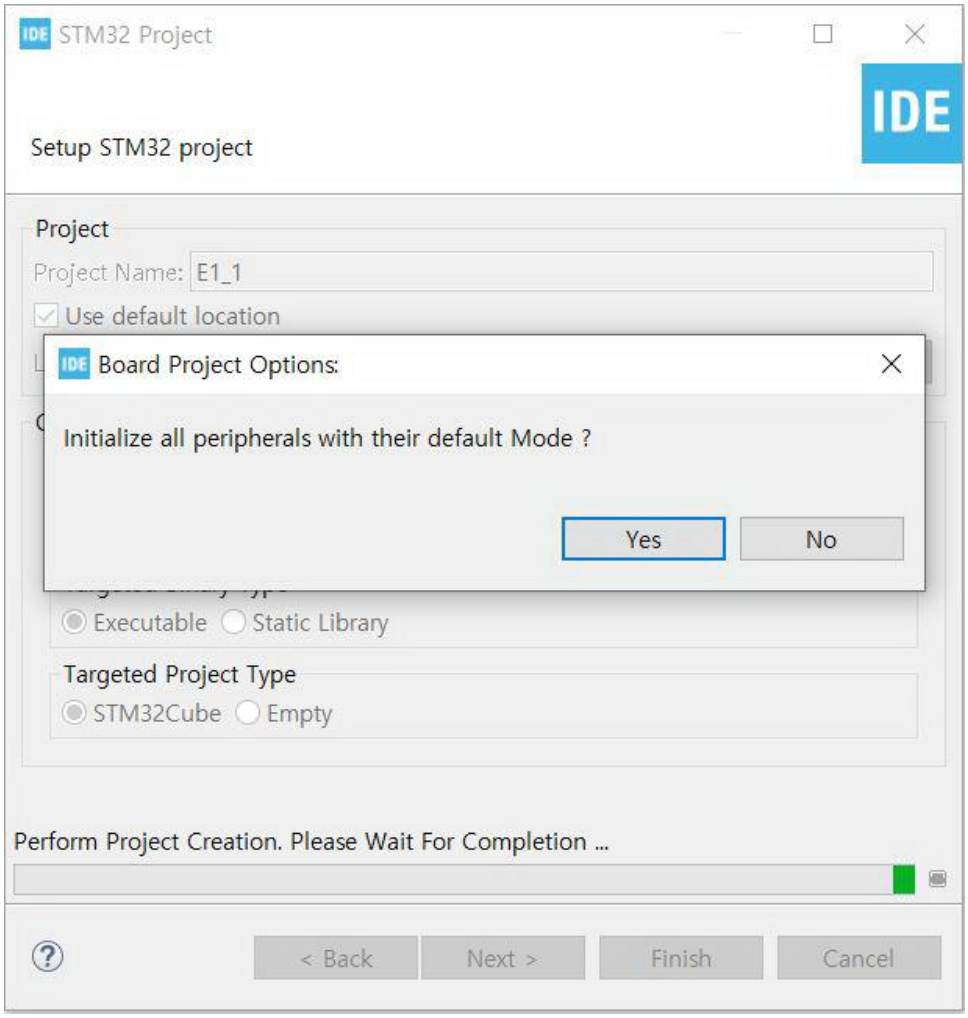

그림 1.17 주변 장치 초기화 설정

다음, 그림 1.18의 Device Configuration Tool editor 설정과 관련한 질문이 나오면 YES를 클릭한다.

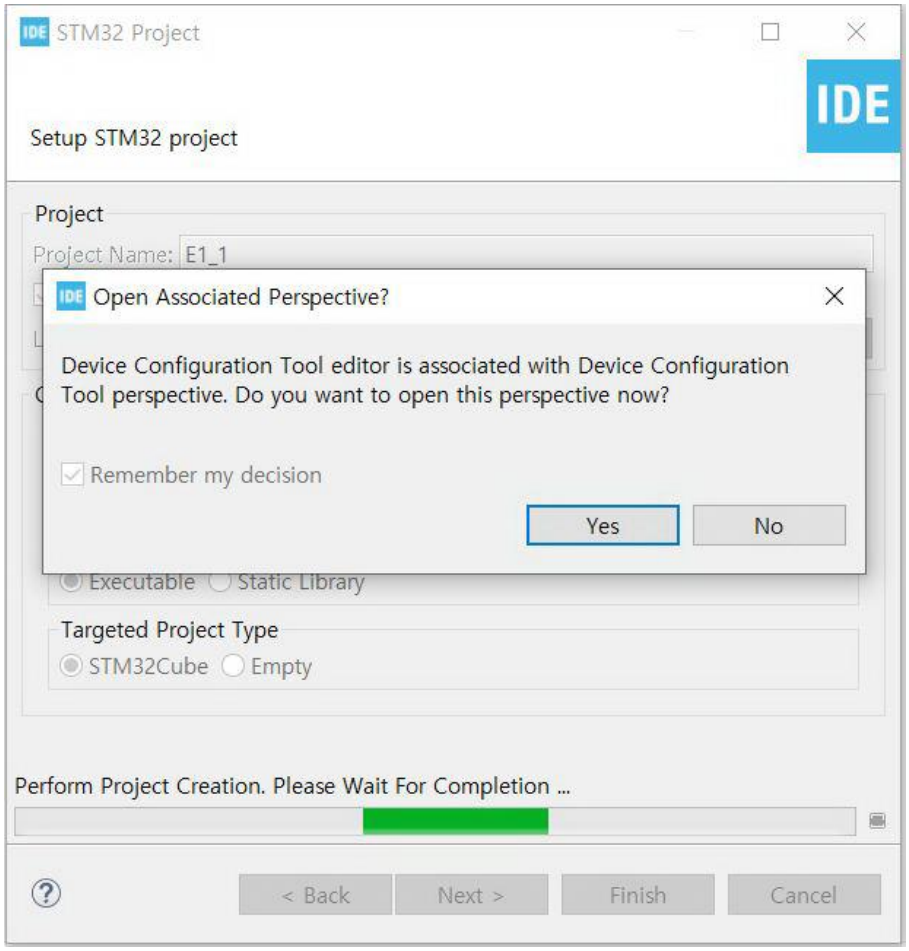

그림 1.18 화면 설정

다음, 그림 1.19는 Device Configuration Tool 화면을 보여주며, 이 화면에서 마이크로컨트롤러에 관한 여러 가지 설정을 할 수 있다.

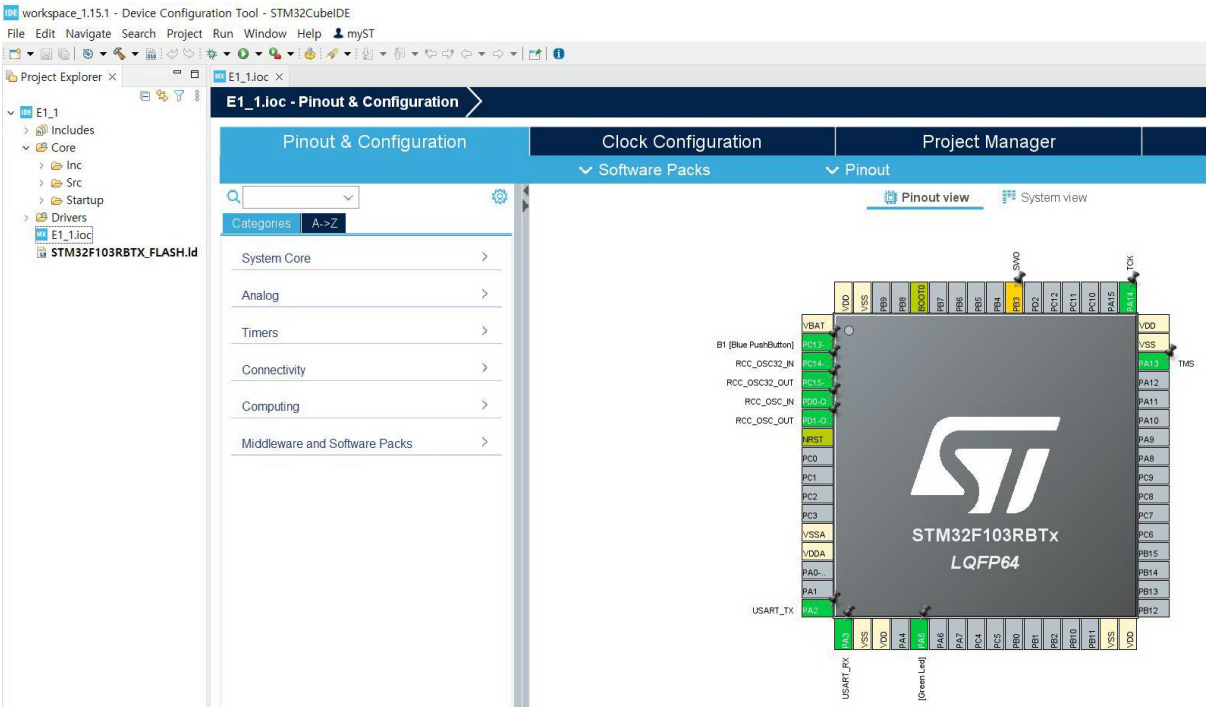

그림 1.19 Device Configuration Tool

다음, 그림 1.20과 같이 Clock Configuration 탭을 클릭해서 클럭 설정에 관한 화면이 보이도록 한다. 클럭 설정 화면에서 PLL Source Mux의 입력을 HSE로 선택하고, PLLMul을 X9로 선택하면, HCLK의 값이 72MHz로 설정이 된다. 이 책에서 실행하는 예제들은 시스템 클럭이 72MHz로 설정된 것을 가정한다. 타이밍과 관련이 없는 프로그램의 경우 클럭 설정을 변경하지 않아도 정상 실행이 가능하지만, 타이밍이 중요한 예제의 경우 클럭이 다른 값으로 설정된다면 정상적으로 동작하지 않을 수 있다.

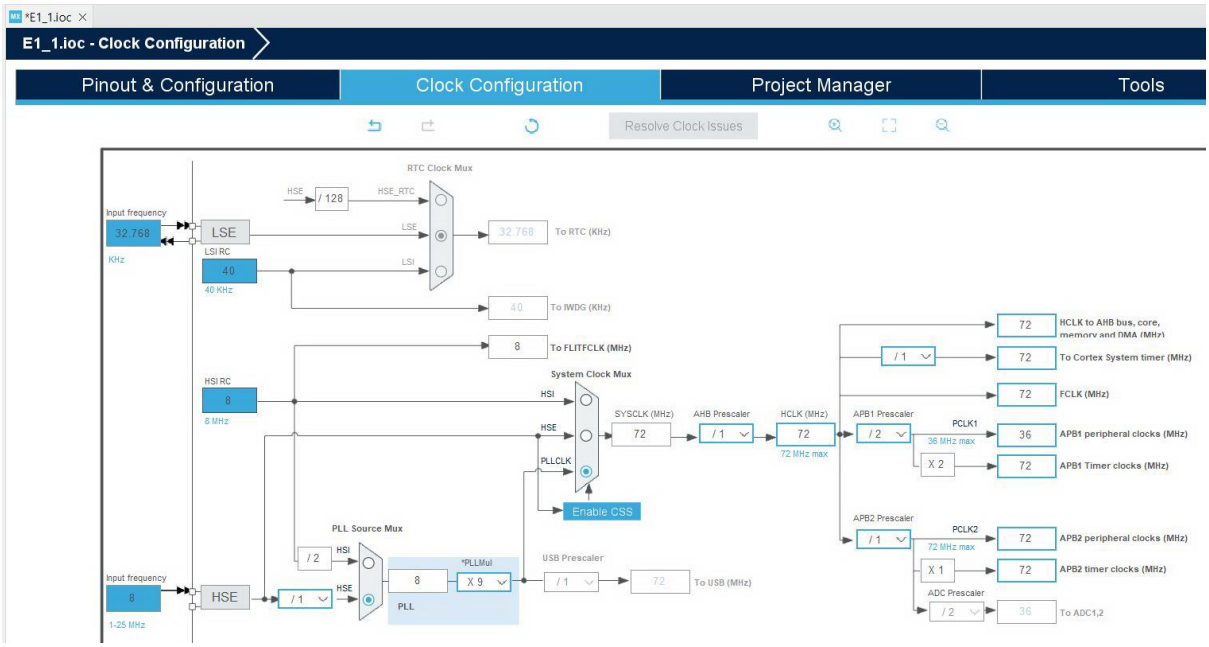

그림 1.20 클럭 설정 화면

위의 그림과 같이 클럭 설정이 완료되면 File 메뉴에서 Save를 클릭하거나, 디스켓 모양의 아이콘을 클릭해서 설정을 저장한다. 설정 저장이 완료되면, 그림 1.21과 같이 코드 생성을 묻는 화면이 나온다.

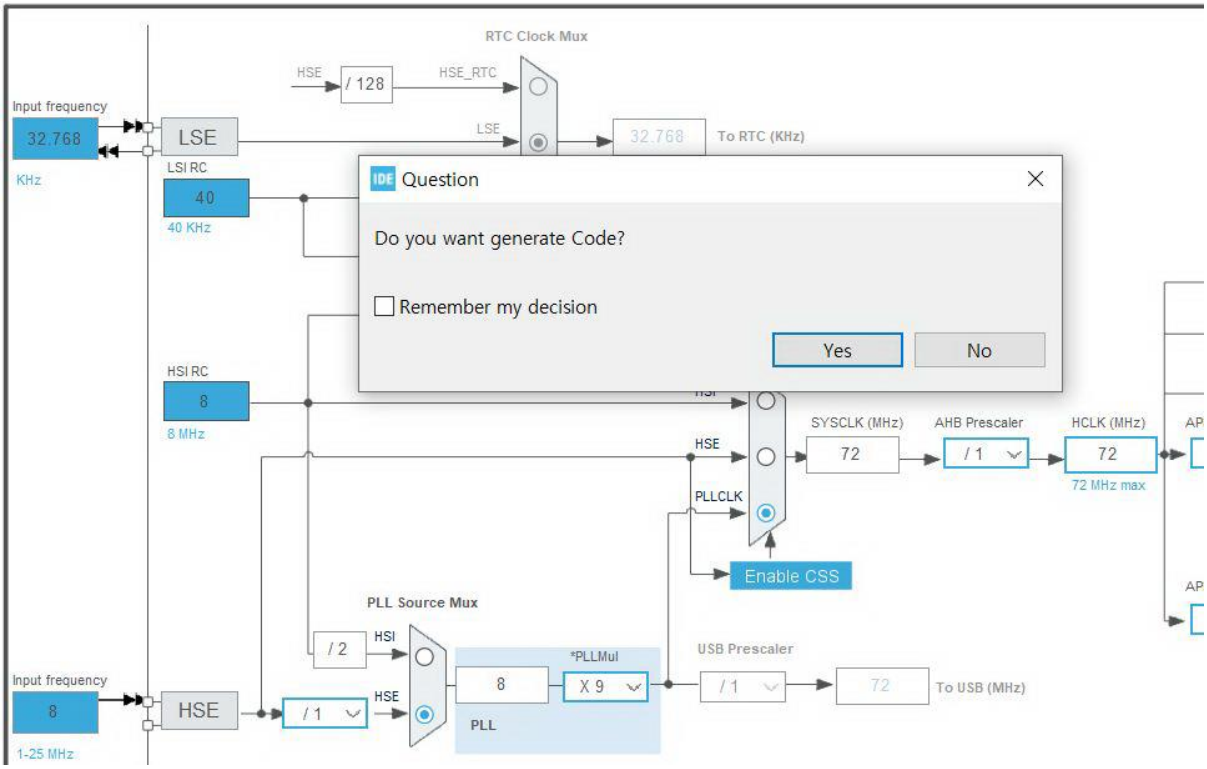

그림 1.21 코드 생성을 묻는 화면

위의 화면에서 YES를 클릭하면, 화면 설정에 대한 질문이 다시 나오며 YES를 클릭하면 코드 생성이 진행된다. 여기에서 Remember my decision을 체크하면, 다음에는 화면 설정에 대한 질문이 나오지 않는다. 코드 생성이 완료되면 그림 1.22와 같이 main.c 소스 코드가 생성된 것을 볼 수 있다. 이 코드는 주변 장치 초기화 동작 이외에는 아무 동작도 하지 않는 코드이며, 필요한 동작은 프로그래머가 소스 코드를 추가해서 하게 된다. 그러나, 주변 장치의 초기화에 필요한 함수들은 이미 생성된 상태이므로, 그러한 함수를 호출하는 것 만으로 초기화를 진행할 수 있다. 특히, 시스템 클럭 설정을 레지스터 레벨에서 코딩하는 것은 매우 복잡한 작업이지만, 그림 1.20과 같이 그림에서 설정을 하면, 이 설정을 실행하는 함수의 소스 코드가 자동으로 생성이 되며, 프로그래머는 함수를 부르는 것 만으로 설정을 할 수 있다. 그림 1.22의 소스 코드에서 함수 SystemClock_Config가 클럭을 설정하는 함수이며, 이 함수를 호출하는 코드는 이미 main.c에 입력된 것을 볼 수 있다. 이와 같은 작업이 가능한 것은 STM32CubeIDE에서 제공하는 HAL 라이브러리가 있기 때문이며, STM32CubeIDE에서는 HAL 라이브러리 함수를 이용해서 레지스터 레벨의 설정을 프로그래머가 직접 하지 않고 함수를 부르는 것 만으로도 마이크로컨트롤러 프로그램 작성이 가능하다. 이와 같은 기능은 프로그램을 개발하는 과정에 소요되는 시간을 많이 단축할 수

있으므로, 이와 같은 방법을 이용해서 실제 제품 개발에 이용하는 사례가 확산되고 있다. 이와 같은 HAL 라이브러리가 편리 하지만, 모든 일반적인 경우를 대비해서 작성된 라이브러리 이므로 소스 코드가 매우 방대하며, 실행 속도가 다소 느려질 수 있는 단점이 있다. 또한, 정상적으로 동작하지 않는 경우에 라이브러리 내부에서 실행되는 내용을 정확하게 파악하지 못하면 문제점 파악이 쉽지 않다. 이 책에서는 STM32CubeIDE를 사용한 여러 가지 실습 예제를 보여주지만, HAL 라이브러리의 사용은 최소한으로 제한할 예정이다. 이 책은 Cortex-M 마이크로컨트롤러의 동작을 구체적으로 설명하고 예제로 보여주는 것이 목적이므로, HAL 라이브러리를 사용하지 않고 각 레지스터들을 직접 설정하는 방법을 보여주게 된다.

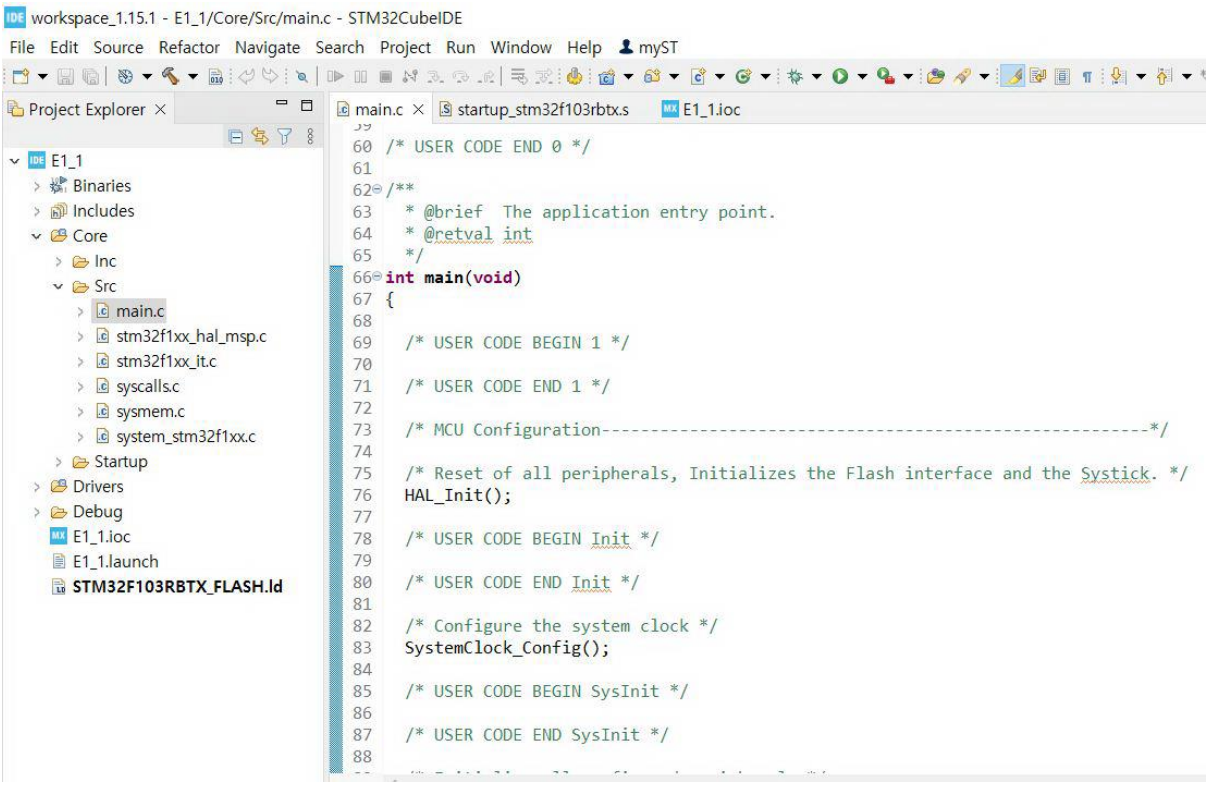

그림 1.22 생성된 코드

위와 같이 생성된 코드는 초기화 이외에 아무 동작도 하지 않지만, 빌드를 진행해서 타겟 보드에서 실행해 본다. Project Explorer에서 프로젝트 이름을 선택한 후, 마우스 오른쪽 버튼을 클릭하면 그림 1.23과 같이 메뉴가 나오며, 이 메뉴에서 Build Project를 클릭한다.

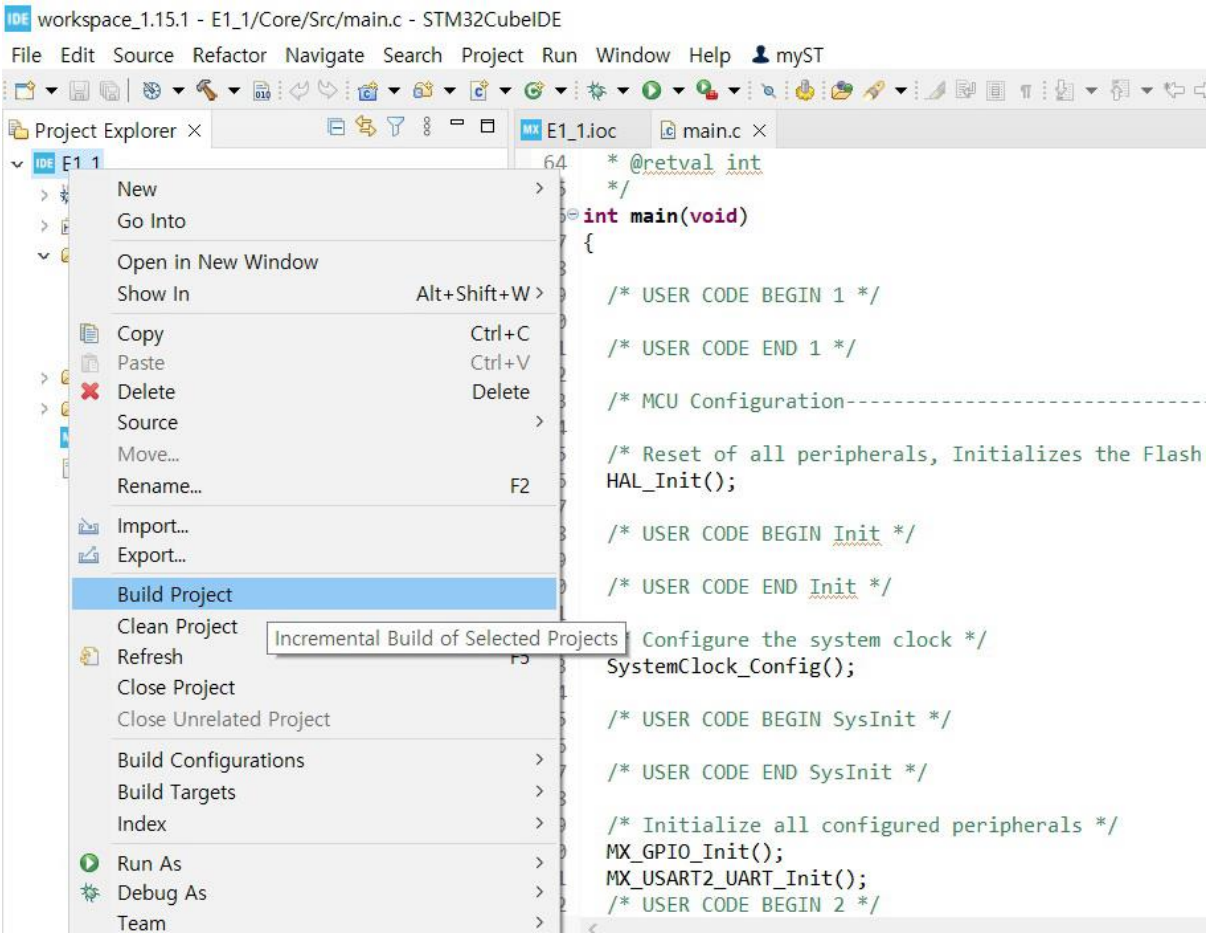

그림 1.23 Build Project

또는, 그림 1.24와 같이 Project 메뉴에서 Build Project를 선택해도 빌드가 시작된다.

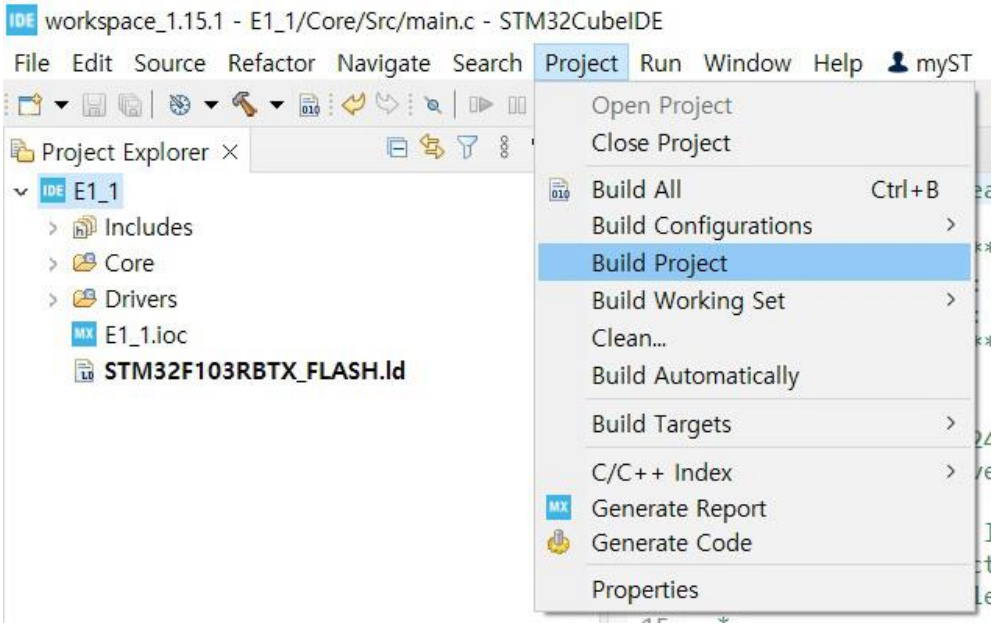

그림 1.24 Build Project

빌드가 끝나고 나면, 그림 1.25와 같이 Console 화면에서 빌드가 오류 없이 끝났음을 볼 수 있다.

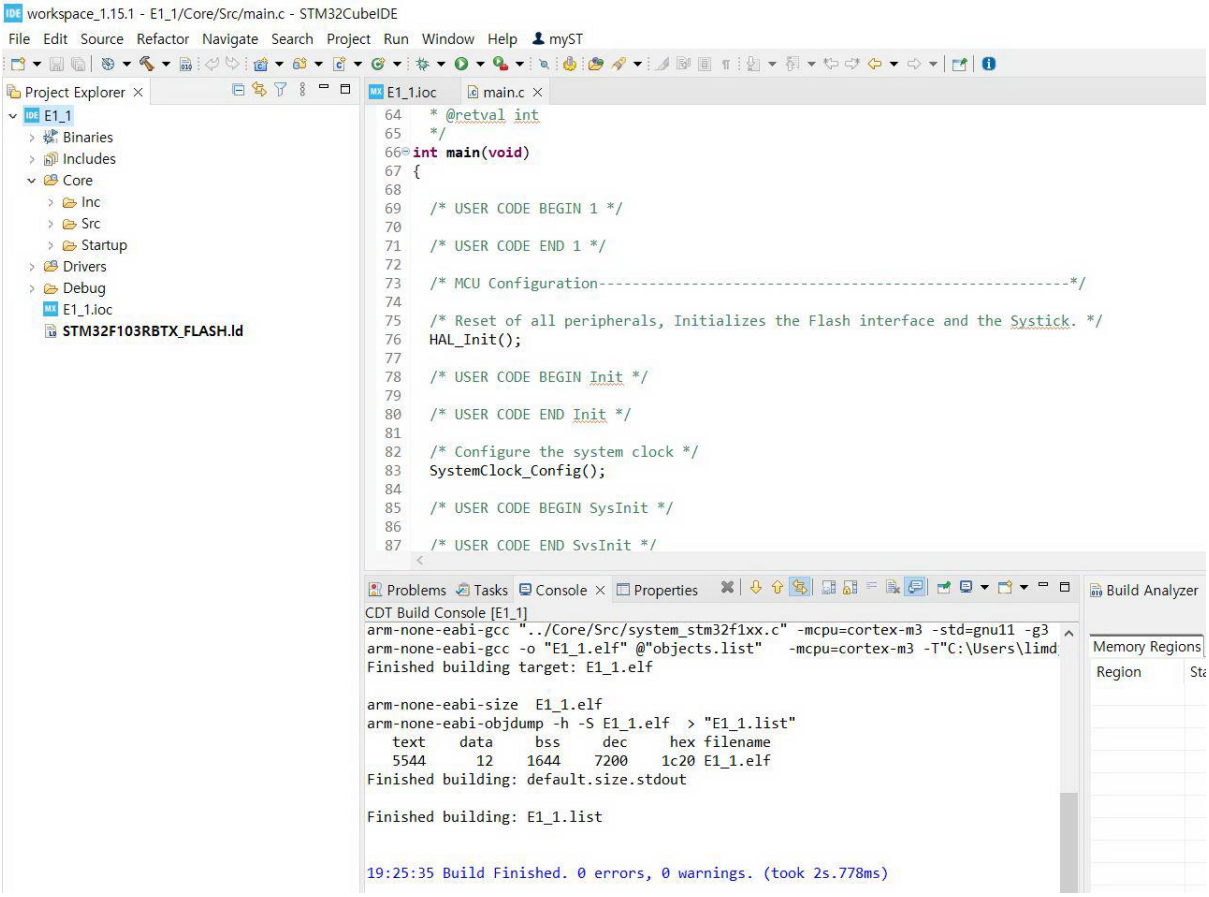

그림 1.25 빌드 완료

성공적으로 빌드가 완료된 실행 파일을 Nucleo-64 보드에서 실행해 보기 위해서는 보드를 컴퓨터와 연결해야 한다. 보드와 컴퓨터의 연결은 USB 케이블을 이용해서 보드 상단의 mini USB 커넥터와 컴퓨터의 USB 단자를 연결하는 것으로 완료된다. 보드 상단의 mini USB 커넥터에는 3가지 기능이 있다. 첫째, USB의 5V 전원을 이용해서 Nucleo-64 보드에 전원을 공급한다. 둘째, 컴퓨터와 디버거를 연결해서 실행 파일을 다운 로드하고 디버깅 기능을 실행할 수 있도록 한다. 세번째, 마이크로컨트롤러의 시리얼 포트를 컴퓨터의 시리얼 포트와 연결한다. 시리얼 포트의 이용 방법에 대해서는 뒤에서 설명하기로 한다. USB 케이블로 보드와 컴퓨터를 연결하면 보드의 전원 LED가 켜져서 전원이 정상적으로 공급되는 것을 알 수 있다. 다음, 그림 1.26과 같이 Run 메뉴에서 Debug를 선택한다.

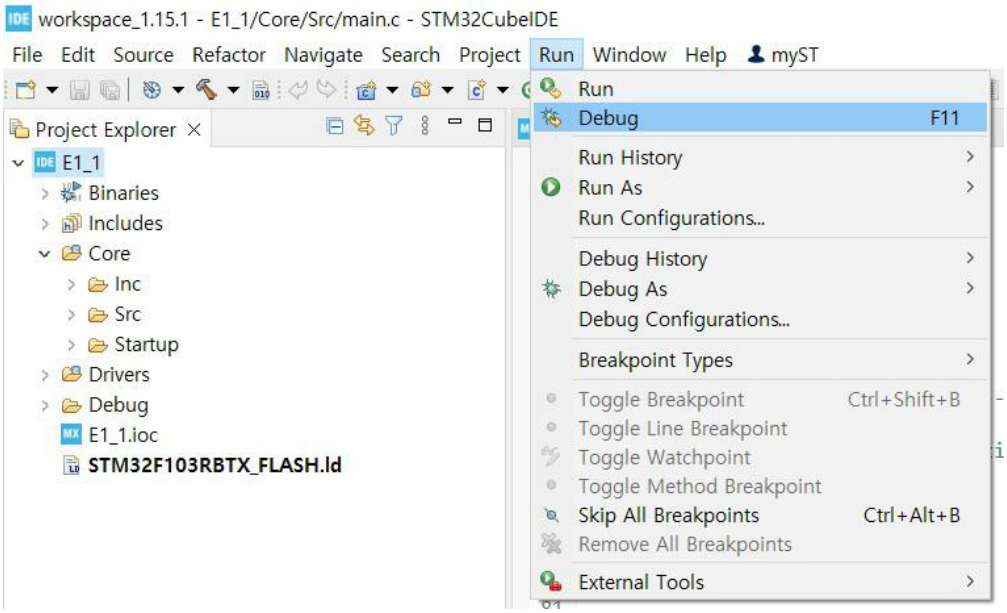

그림 1.26 Debug 시작

또는, 그림 1.27과 같이 Project Explorer에서 프로젝트 이름을 선택한 후, 마우스 오른쪽 버튼을 누르면 나오는 메뉴에서 Debug As를 선택한다.

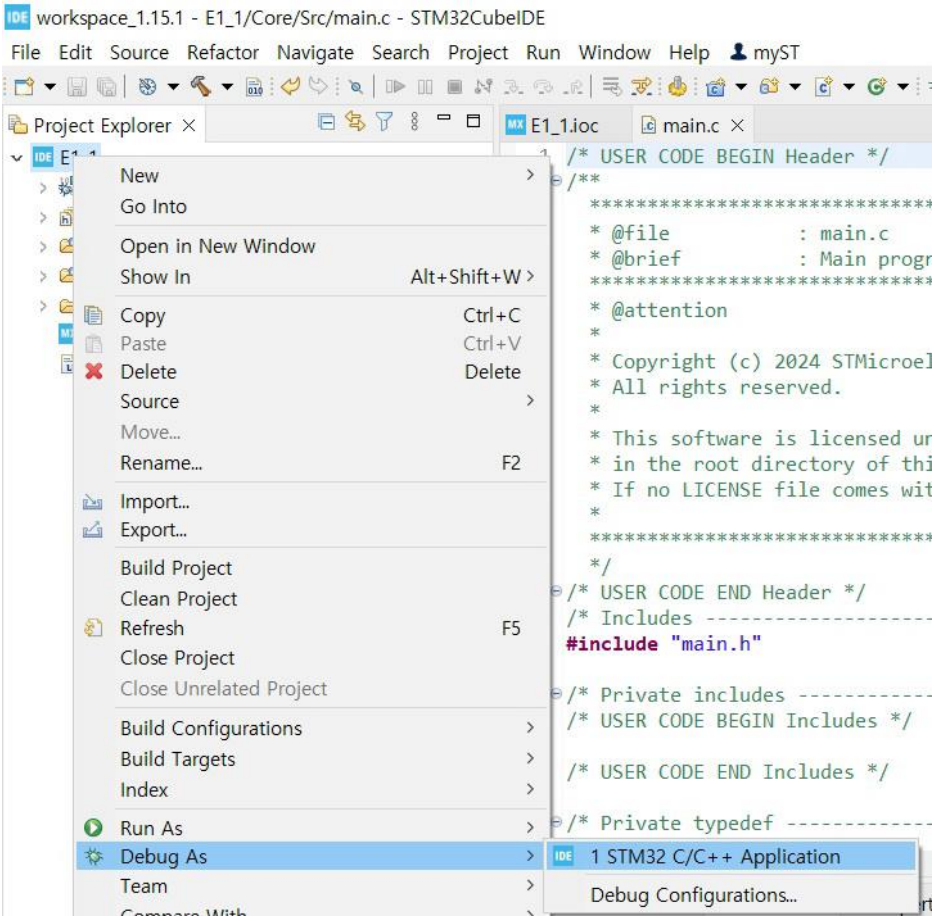

그림 1.27 Debug 시작

다음, 그림 1.28과 같이 실행을 위한 설정 화면이 나오고, OK를 클릭한다.

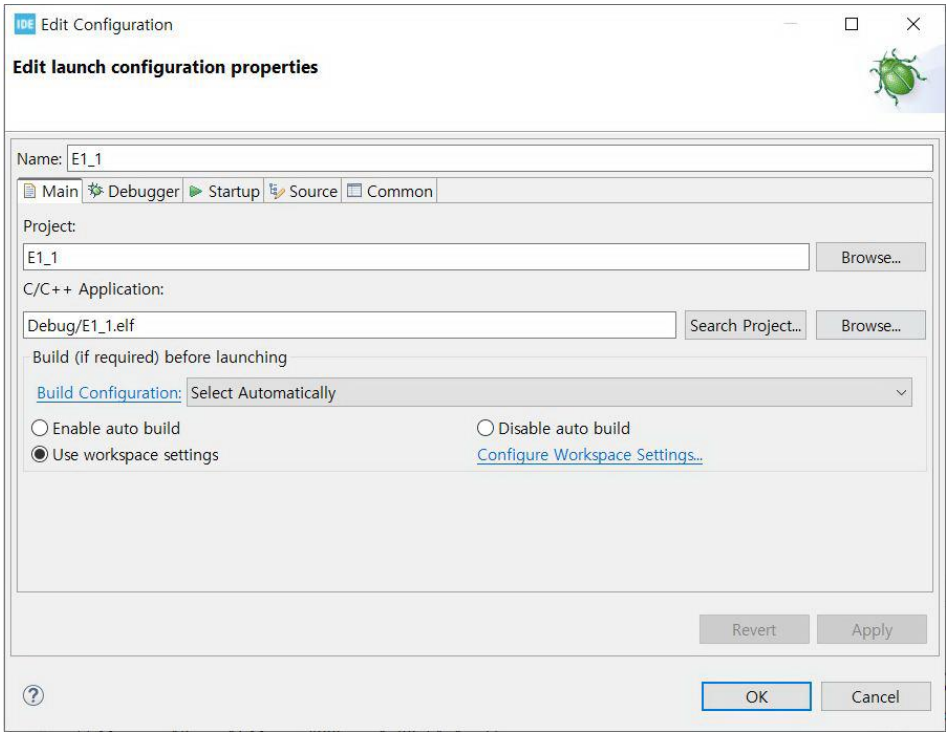

그림 1.28 실행 설정

디버그 화면으로 전환하기 전에 그림 1.29와 같이 화면 설정에 대한 질문 화면이 나오며, Switch를 선택한다. 이전 선택 화면과 마찬가지로 Remember my decision을 체크하면 다음에는 묻지 않는다.

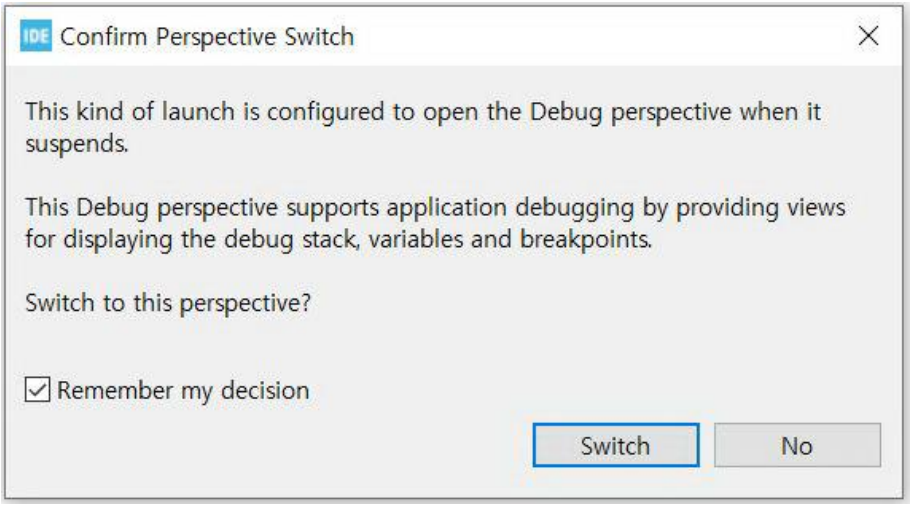

그림 1.29 화면 설정

그림 1.30은 디버그 화면을 보여준다. 디버거가 접속이 되고 실행 파일의 다운 로드가 완료되는 데는 시간이 걸리므로 완료될 때까지 기다려야 한다. 다운 로드가 완료되면 Download verified successfully 메시지가 화면에 나온다.

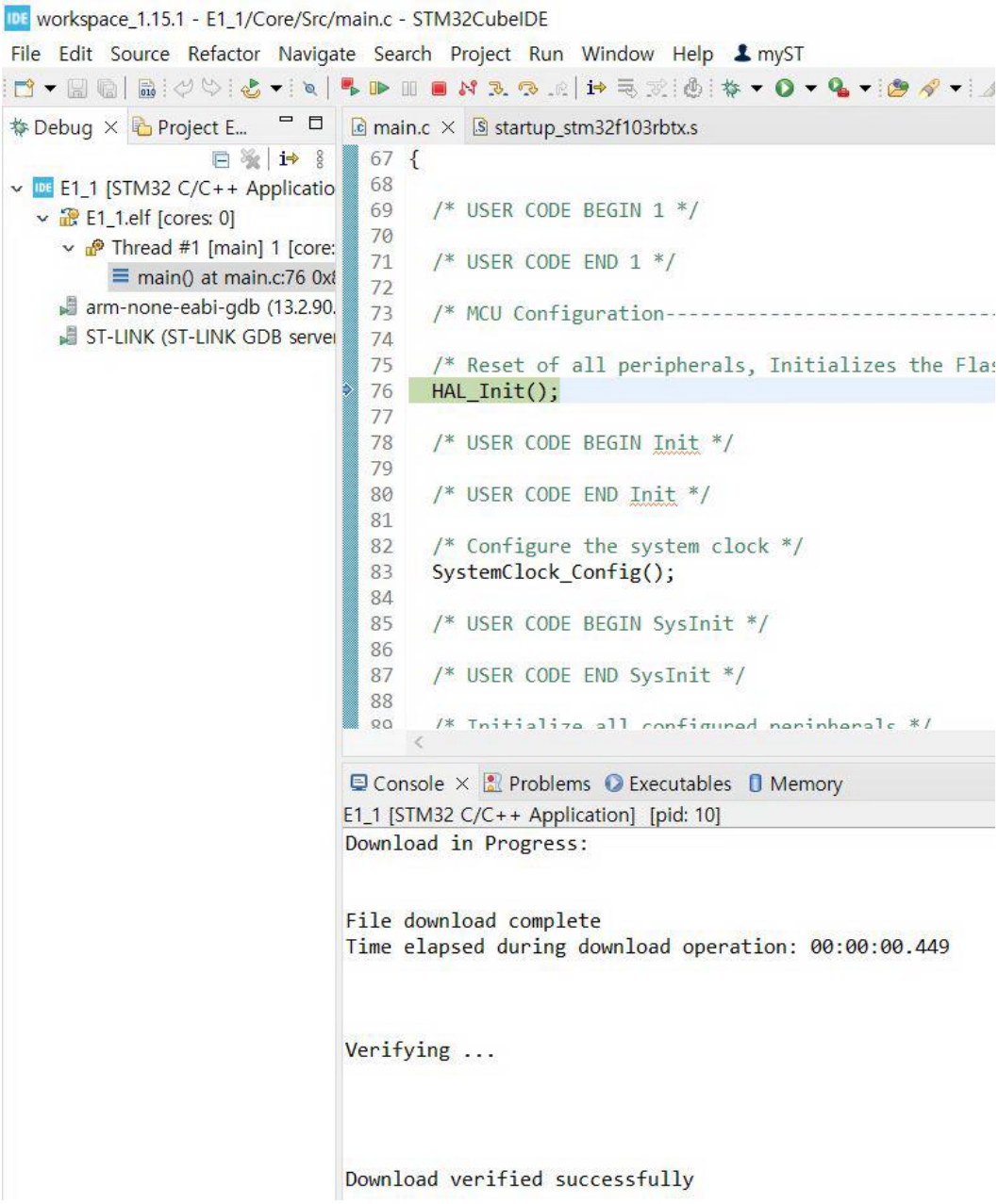

그림 1.30 다운 로드 완료

다운 로드가 완료된 후에는 프로그램이 실행되지 않고 프로그램의 첫 부분에서 대기하고 있는 상태이다. 프로그램의 실행을 진행하기 위해서는 그림 1.31과 같이 Run 메뉴에서 Resume을 선택하거나, 메뉴바에서 녹색의 삼각형 버튼을 누른다.

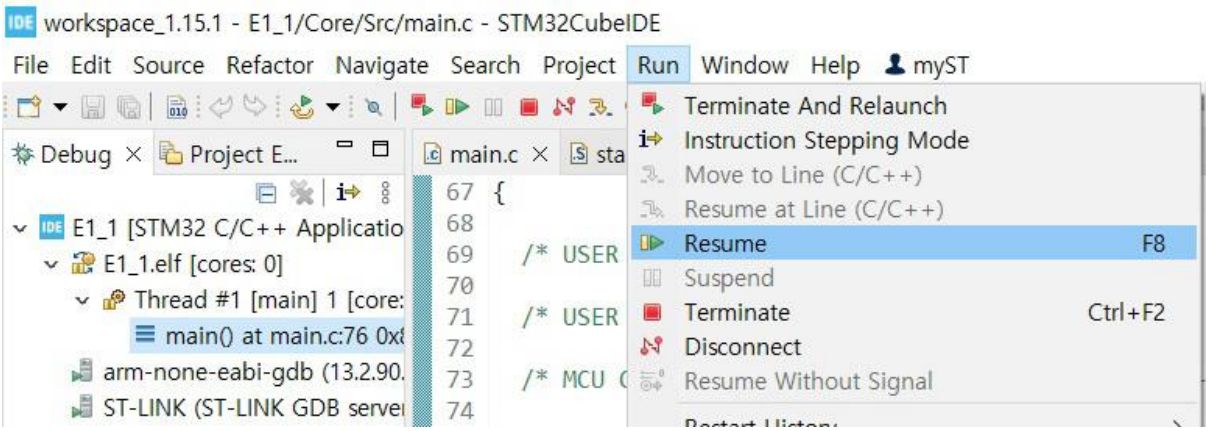

그림 1.31 실행

프로그램의 실행 중 실행을 중단하기 위해서는 그림 1.32와 같이 Run 메뉴에서 Terminate를 선택하거나, 메뉴바에서 빨간색의 사각형 버튼을 누른다.

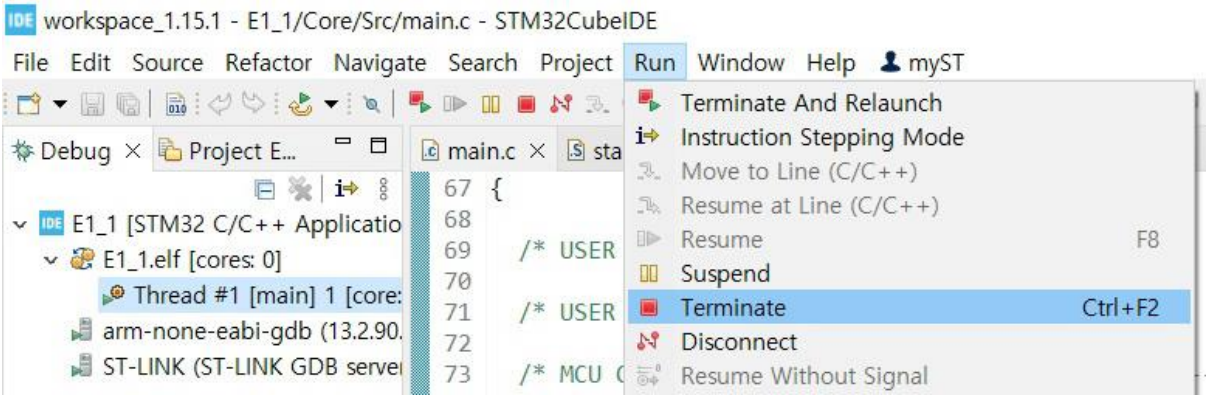

그림 1.32 실행

[예제 1.1] 앞에서 만든 Project E1_1의 main.c 파일에서 main 프로그램의 while 루프 내에서 /* USER CODE BEGIN 3 */의 바로 아래에 프로그램 1.1과 같이 입력한 후, 빌드를 해서 Nucleo-64 보드에 실행 파일을 다운 로드하고 실행해 본다. 보드에는 사용자가 사용할 수 있는 녹색 LED가 1개 있으며, 이 녹색 LED가 1초에 한번씩 깜빡이는 것을 관찰할 수 있다. LED는 GPIO 포트에 연결되어 있고, LED를 켜거나 끄기 위해서는 GPIO에 관한 여러 가지 설정이 필요하다. 그러나, 이 예제와 같이 HAL 라이브러리를 사용하면, 설정에 관한 코드는 모두 자동 생성이 되어있으므로, HAL 라이브러리 함수를 이용해서 동작과 관련된 코드를 입력하면 간단하게 LED를 켜거나 끌 수 있다.

프로그램 1.1

```
/* USER CODE BEGIN 3 */
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
        HAL_Delay(500);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
        HAL_Delay(500);
 }
 /* USER CODE END 3 */
```

[예제 1.2] 예제 1.1의 프로그램과 동일한 동작을 하는 프로그램을 HAL 라이브러리를 사용하지 않고 레지스터 설정에 의해서 동작하는 프로그램을 작성하고 실행해 본다. 먼저, STM32CubeIDE에서 이전과 동일한 방법으로 프로젝트를 생성한다. 프로젝트의 이름은 E1_2로한다. 그림 1.33과 같이 한 개의 워크스페이스에 2개의 프로젝트가 있는 것을 볼 수 있다. 이와 같이 한 개의 워크스페이스에는 여러 개의 프로젝트가 포함될 수 있다. 여러 개의 프로젝트가 한 개의 워크스페이스에 있을 경우, 현재 빌드하고 실행되는 프로젝트가 올바로 선택되어 있는지에 대해서 주의가 필요하다. 종종, 현재 작업 중인 프로젝트가 아닌 다른 프로젝트를 빌드하는 실수를 할 수 있다.

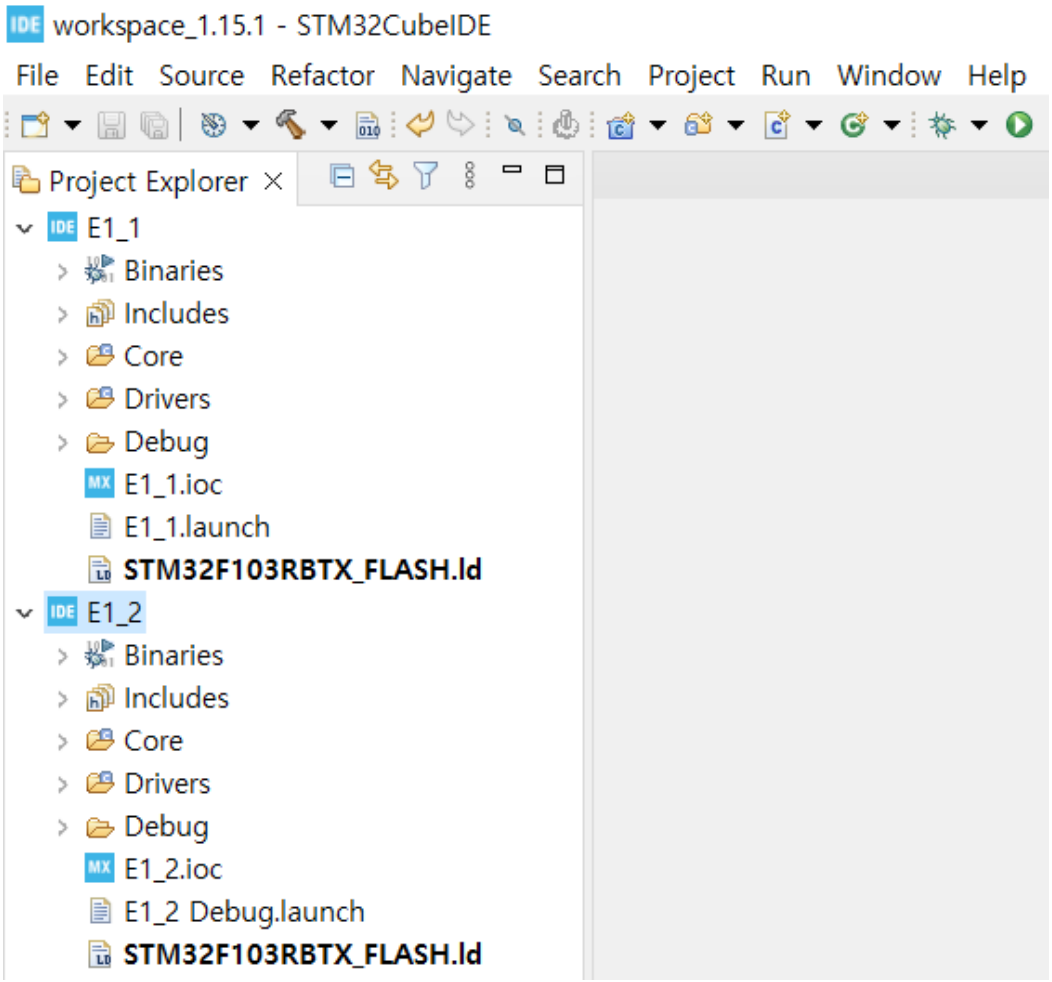

그림 1.33 새로운 프로젝트 추가 생성

다음, Project Explorer에서 E1_2 프로젝트의 Core 폴더 아래의의 Src 폴더를 선택한 후, 그림 1.34와 같이 마우스 오른쪽 버튼을 눌러서 나오는 메뉴에서 New와 Source File을 선택한다. File 메뉴에서도 동일한 선택을 할 수 있다.

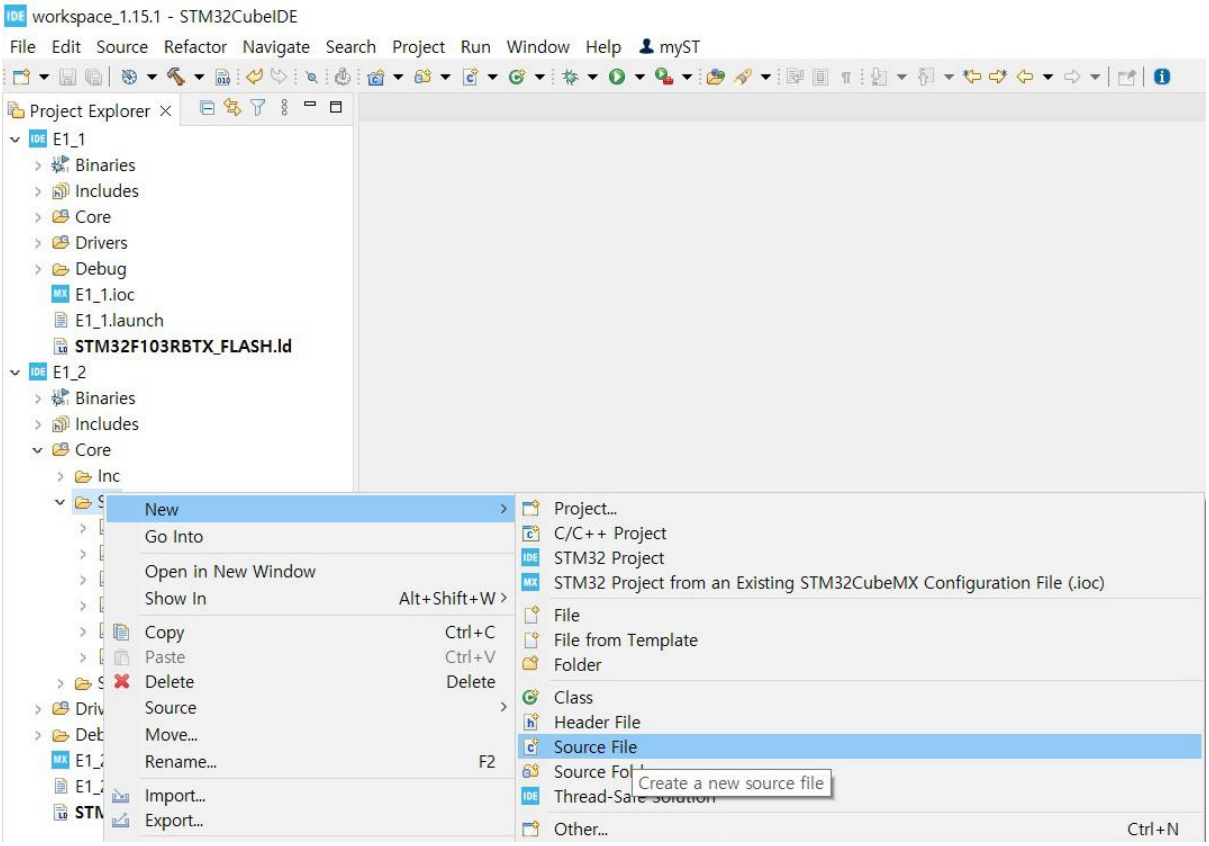

그림 1.34 소스 파일 추가

다음, 그림 1.35와 같이 소스 파일 이름을 E1_2.c 로 입력하고 Finish 버튼을 누른다.

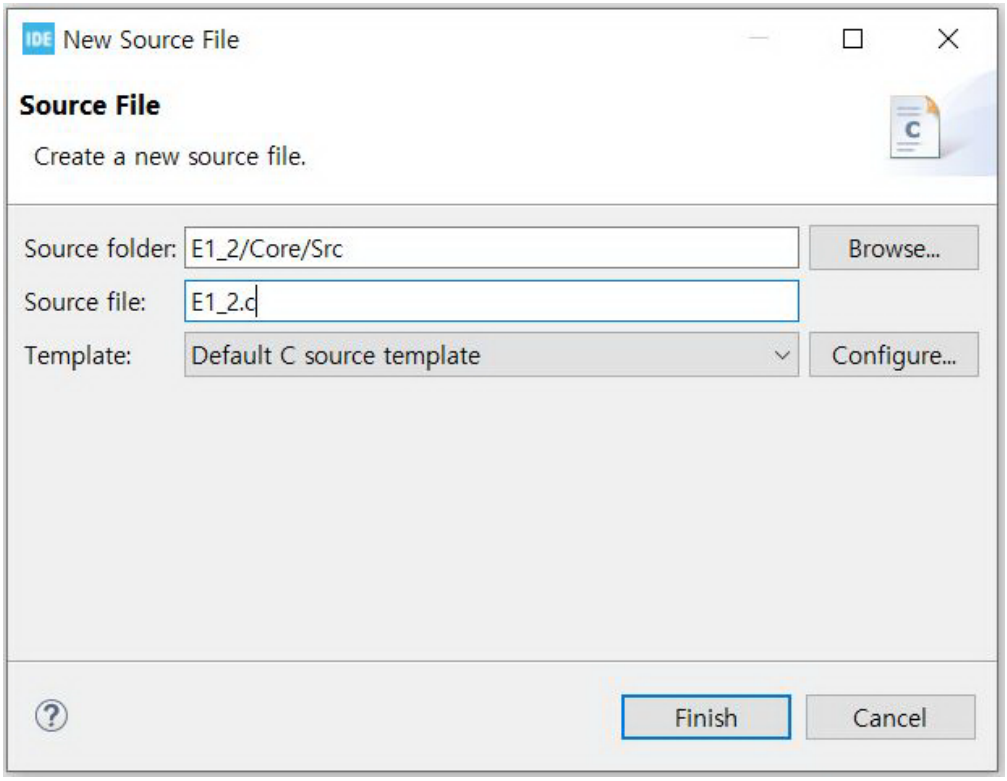

그림 1.35 파일 이름 입력

다음, 그림 1.36과 같이 새로 생성된 소스 파일에 아래 주어진 프로그램 1.2의 내용을 복사해서 넣는다.

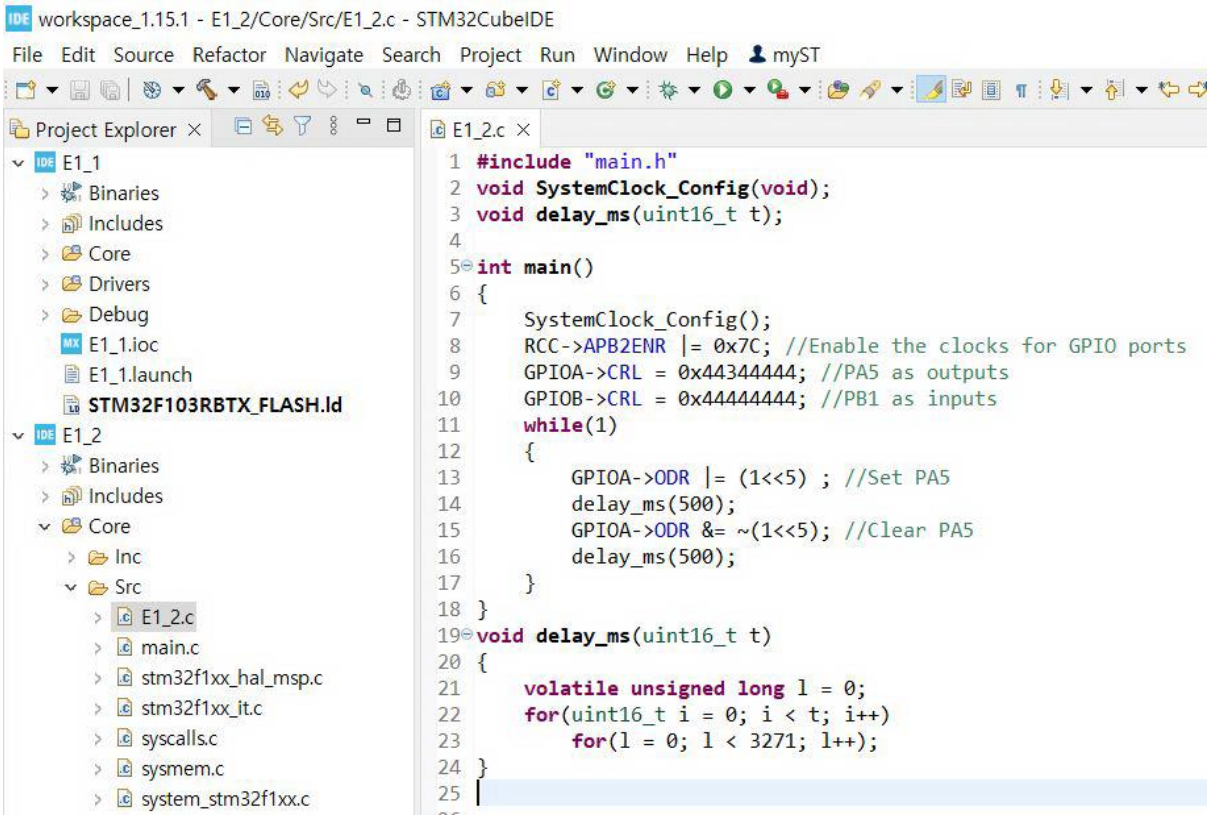

그림 1.36 소스 파일 내용 복사

이와 같은 상태에서 빌드를 하면 동일 프로젝트내에 2개의 main 함수가 있게 되므로 오류가 발생한다. 따라서, 그림 1.37과 같이 원래의 main.c 에서 main 함수의 이름을 다른 것으로(예를 들면 not_main 등) 바꿔준다. 이때, 파일 main.c 를 삭제하지 않는 이유는 시스템 클럭을 설정하는 함수가 main.c에 들어있고, 시스템 클럭을 설정하기위해서 이 함수를 호출해야하기 때문이다.

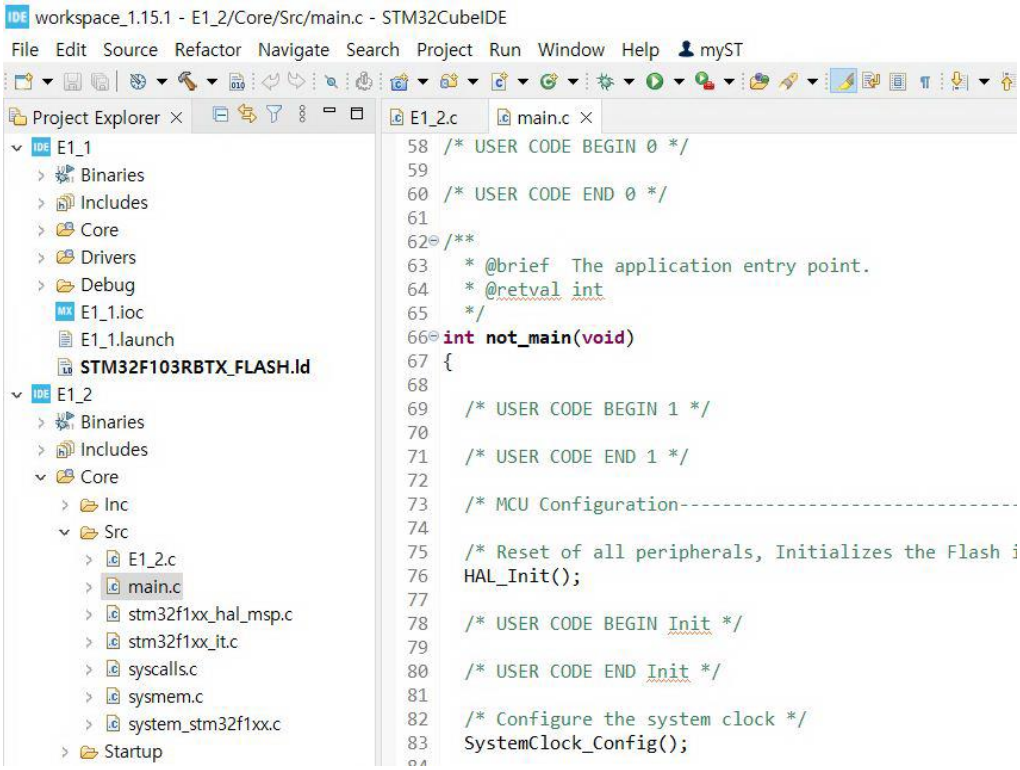

그림 1.37 자동 생성된 메인 함수 이름 변경

다음으로 빌드를 실행한 후, 예제 1.1과 같은 방법으로 Nucleo-64 보드에서 실행한다. 앞에서 언급한 바와 같이, 이 책에서는 HAL 라이브러리를 사용하지 않으므로, 앞으로 이 책에서 다루는 거의 모든 예제들은 이 예제에서 실행한 방법으로 실행이 가능하다.

프로그램 1.2

```
#include "main.h"
void SystemClock_Config(void);
void delay_ms(uint16_t t);

int main()
{
	SystemClock_Config();
	RCC->APB2ENR |= 0x7C; //Enable the clocks for GPIO ports
	GPIOA->CRL = 0x44344444; //PA5 as outputs
	GPIOB->CRL = 0x44444444; //PB1 as inputs
	while(1)
	{
		GPIOA->ODR |= (1<<5) ; //Set PA5
		delay_ms(500);
		GPIOA->ODR &= ~(1<<5); //Clear PA5
		delay_ms(500);
	}
}
void delay_ms(uint16_t t)
{
	volatile unsigned long l = 0;
	for(uint16_t i = 0; i < t; i++)
		for(l = 0; l < 3271; l++);
}
```