



CORTEX-M4 마이크로 컨트롤러

- 마이크로 컨트롤러의 정의
- Cortex-M4 마이크로 컨트롤러 개요
- STM32F405 마이크로 컨트롤러

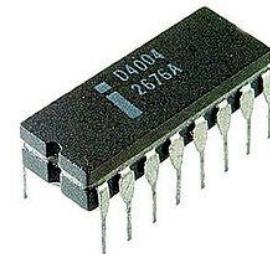


엣지아이랩

マイクロ 컨트롤러의 정의

- 마이크로 프로세서
 - 프로세서를 하나의 칩 안에 집적하여 넣어 소형화한 형태
 - 컴퓨터의 발전과 함께 고성능의 프로세서로 발전했으며, 최근에는 64비트의 고성능 프로세서들이 출시
 - 마이크로 프로세서는 점점 고성능화 하면서 범용 컴퓨터에 사용

- 마이크로 프로세서의 발전
 - 1971년 Intel사 4bit 마이크로프로세서 4004 개발
 - 이후 여러 회사에서 8bit 마이크로프로세서 개발
 - Intel : 8008('72), 8080('74), 8085('76)
 - Motorola : MC6800('74), MC6805('76), MC6809('77)
 - Zilog : Z80('76)
 - 이후 8, 16, 32, 64bit 마이크로프로세서 개발
 - Intel : 80186, 80286, 80386, 80486, Pentium,
 - Motorola : 68000, 68020, 68040, 68060,



인텔 4004
최초의 상용화한
마이크로 프로세서



인텔 80286
마이크로 프로세서

출처 : wikipedia
(www.wikipedia.com)

マイクロ 컨트롤러의 정의

- 마이크로 컨트롤러(MCU: Micro Controller Unit)
 - 지능화와 소형화를 위하여 마이크로 프로세서에 메모리와 각종 주변장치들을 함께 집적하여 넣은 칩
 - 마이크로 프로세서 코어, 여러 가지 크기와 다양한 종류의 메모리, 여러 종류의 주변장치 및 입출력 포트를 하나의 칩에 집적
 - 여러 응용분야에 필요한 주변기기들을 모두 제공
- 마이크로컨트롤러의 발전
 - 1975년 : Texas Instrument TMS1000 개발(마이크로컨트롤러의 시초)
(1971년 Intel 4bit 마이크로프로세서 4004 개발)
 - 1976년 : Intel 8bit M/C 8048(MCS-48) 개발
 Motorola 8bit MC6801 개발
 - 1980년 : Intel 8bit M/C 8051(MCS-51) 개발
 - 1982년 : Intel 16bit MCS-96 개발
 - 1988년 : Intel 32bit M/C 80960 개발
 - etc



Texas Instrument
TMS1000

출처 : wikipedia
(www.wikipedia.com)

마이크로 컨트롤러의 정의

- MCU 시스템 :

- 연산, 제어, 및 레지스터로 구성된 마이크로프로세서의 기능에 추가적으로 ROM 또는 RAM과 같은 메모리와 외부 입출력 장치를 포함한 특수 목적의 기능을 탑재한 집적 회로 소자를 MCU라 통칭하며 이를 사용하는 시스템
- 마이크로 프로세서와 특정 목적용 기능을 포함한 특정한 목적을 수행하는 소자



マイクロ 컨트롤러

● 마이크로 컨트롤러(MCU)의 특징

- 주변장치들을 센싱 및 제어하기 위한 I/O 능력이 강화
- 타이머/카운터, 통신 포트 내장 및 인터럽트 처리 능력 보유
- bit 조작 능력이 강화
- 제품의 소형화 및 경량화
- 제품의 가격이 저렴(부품비, 제작비, 개발비 및 개발 시간 절감)
- 융통성 및 확장성이 용이(프로그램만 변경)
- 신뢰성이 향상(부품 수 적어 시스템 단순, 고장률 적고, 보수 편리)

- マイクロ 프로세서 vs 마이크로 컨트롤러

구분	마이크로 프로세서	마이크로 컨트롤러
특징	<ul style="list-style-type: none">CPU를 단일 IC칩으로 만든 반도체MPU(Micro-Processor Unit)주변 하드웨어 필요범용적인 목적의 컴퓨터에 사용	<ul style="list-style-type: none">CPU+메모리+하드웨어 제어회로MCU(Micro-Controller Unit)Single-Chip Microcomputer (1개의 칩으로 컴퓨터 구현 가능)특별한 목적의 기기 제어용으로 사용

マイクロ 컨트롤러

● 마이크로컨트롤러(MCU)의 응용

- 산업 : 모터 제어, 로봇 제어, 프로세스 제어, 수치 제어, 장난감 등
- 계측 : 의료용 계측기, 오실로스코프 등
- 가전제품 : 전자레인지, 가스 오븐, 전자 밥솥, 세탁기 등
- 군사 : 미사일 제어, Torpedo 제어, 우주선 유도 제어 등
- 통신 : 휴대폰, 모뎀, 유무선 전화기, 중계기 등
- 사무기기 : 복사기, 프린터, plotter, 하드디스크 구동장치 등
- 자동차 : 점화 타이밍 제어, 연료 분사 제어, 변속기 제어 등
- 생활 : 전자시계, 계산기, 게임기, 금전등록기, 온도조절기 등

マイクロ 컨트롤러

- 마이크로컨트롤러(MCU) 제조사
 - Motorola(FreeScale) : MC6805, MC68HC16, MC68332, HCS12
 - Samsung : KS51, KS88, KS16, KS32
 - Microchip : PIC16/17
 - Atmel : AVR시리즈, 8051시리즈
 - Zilog : Super-8
 - STMicro : STM32 시리즈
 - Texas Instrument : MSP시리즈

マイクロ 컨트롤러

- 마이크로 컨트롤러(MCU)의 발전 방향
 - 고성능화
 - 32비트 ARM 코어를 내장
 - 다기능화
 - 다양한 특수 기능들을 내장
 - 소형화
 - 초소형 임베디드 시스템 장착
 - 저전력화
 - 소형 배터리로 장시간 동작이 가능
 - 저가격화
 - 1\$ 이하의 가격

Cortex-M4 마이크로컨트롤러

- Cortex-M4 마이크로컨트롤러

- ARM사에서 개발된 RISC(Reduced Instruction Set Computer) 구조의 저전력 CMOS 32-Bit 프로세서인 Cortex-M4 Core를 내장한 ST사의 고성능 마이크로컨트롤러 중 하나
- ST사에서는 ARM사의 Cortex 프로세서군의 하나인 Cortex-M4를 라이선스하여 이를 기반으로 STM32F4xx라는 제품명으로 다양한 프로세서를 출시
- STM32F405는 STM32 제품군들 중에서 상위의 모델에 해당하며, 최대 168MHz 속도로 동작, 이 칩은 Flash 메모리와 SRAM을 제공하며 이 외에 다양한 주변장치를 제공



출처 : www.mouser.com/ARMCortexM4

- About ARM

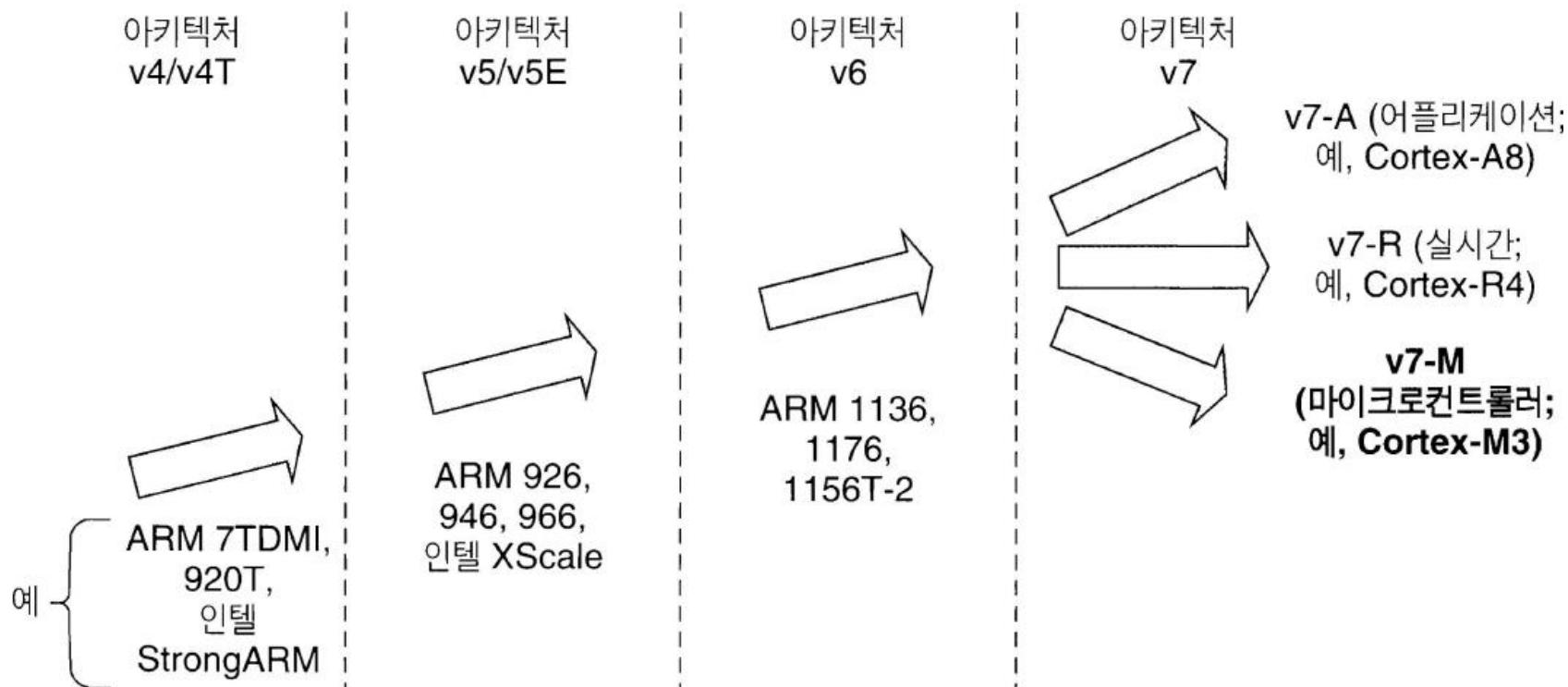
- `1990년 Apple, VLSI, Acorn사의 합작 벤처로 Advanced RISC Machines Ltd.라는 이름으로 설립
- 1991년 최초 제품인 ARM6 프로세서군 소개하였고 VLSI에서 최초 라이센스함
- 이후 TI, NEC, Sharp, ST 등의 유수회사에서 ARM 프로세서를 라이센스함
- ARM는 반도체를 직접 제작하지 않고 지적 재산형태인 프로세서 코어에 대한 지적 재산만을 판매하는 독특한 영업방식을 고수
- 프로세서 이외에 시스템 레벨 IP, 소프트웨어 IP 관련 기술을 추가로 라이선스하고 있음

ARM & ARM Architecture

- ARM 프로세서 작명
 - ARM7TDMI(-S), ARM926EJ(-S), ARM1176JZ(-S)
 - T : Thumb 명령어 지원
 - D : JTAG Debug
 - M : 고속 곱셈기
 - I : 임베디드 ICE
 - -S : 신세사아저블 코어
 - E : 추가 명령어셋 지원
 - J : Java(Jazelle) 기술
 - Z : TrustZone

ARM & ARM Architecture

● ARM Architecture



ST STM32 마이크로컨트롤러

- ST STM32 마이크로컨트롤러 내부 구조



Notes:

- HS requires an external PHY connected to the ULPI interface
- Crypto/hash processor on STM32F417 and STM32F415

출처 : www.microcontroller.com

ST STM32 마이크로컨트롤러

- ST STM32 마이크로컨트롤러의 특징
 - 고성능 저전력 Cortex-M4 32bit Microcontroller
 - 향상된 Harvard 아키텍쳐(1.25DMIPS/MHz@168MHz)
 - 비휘발성 프로그램과 데이터 메모리
 - 최대 1Mbyte 내부 프로그램 가능한 ISP Flash memory
 - 선택적인 Boot code section (used In-System Programming by On-chip Boot Program)
 - ISP (In System Programming)를 통해 어플리케이션 영역과 부트영역에 있어 F/W 다운로드 가능
 - 192+4kbyte 내부 SRAM
 - JTAG Interface
 - 내장 메모리의 Programming과 On-Chip Debug를 위한 JTAG 지원

ST STM32 마이크로컨트롤러

- ST STM32 마이크로컨트롤러의 주변 장치 특징
 - 12개의 16비트 타이머/카운터(각각 4개의 IC/OC/PWM)
 - 2개의 32비트 타이머/카운터(각각 4개의 IC/OC/PWM)
 - 2개의 워치독 타이머
 - 1개의 systick timer (24bit down counter)
 - 분리된 오실레이터에 의한 Real Time Count
 - Output Compare Modulator
 - 세 개의 12bit ADC, 24 채널
 - 두 개의 12bit DAC
 - 세 개의 Two-wire Serial 인터페이스
 - 네 개의 시리얼 USART/ 두 개의 시리얼 UART

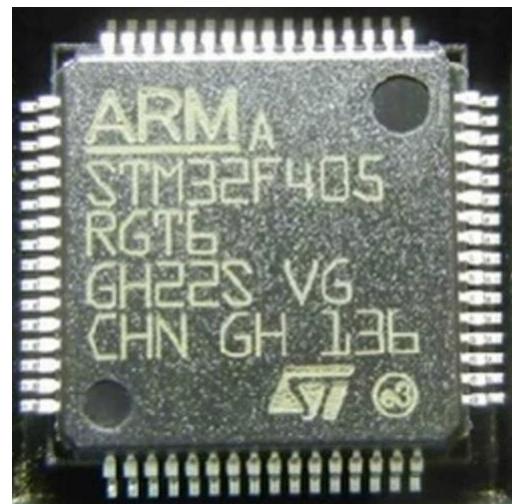
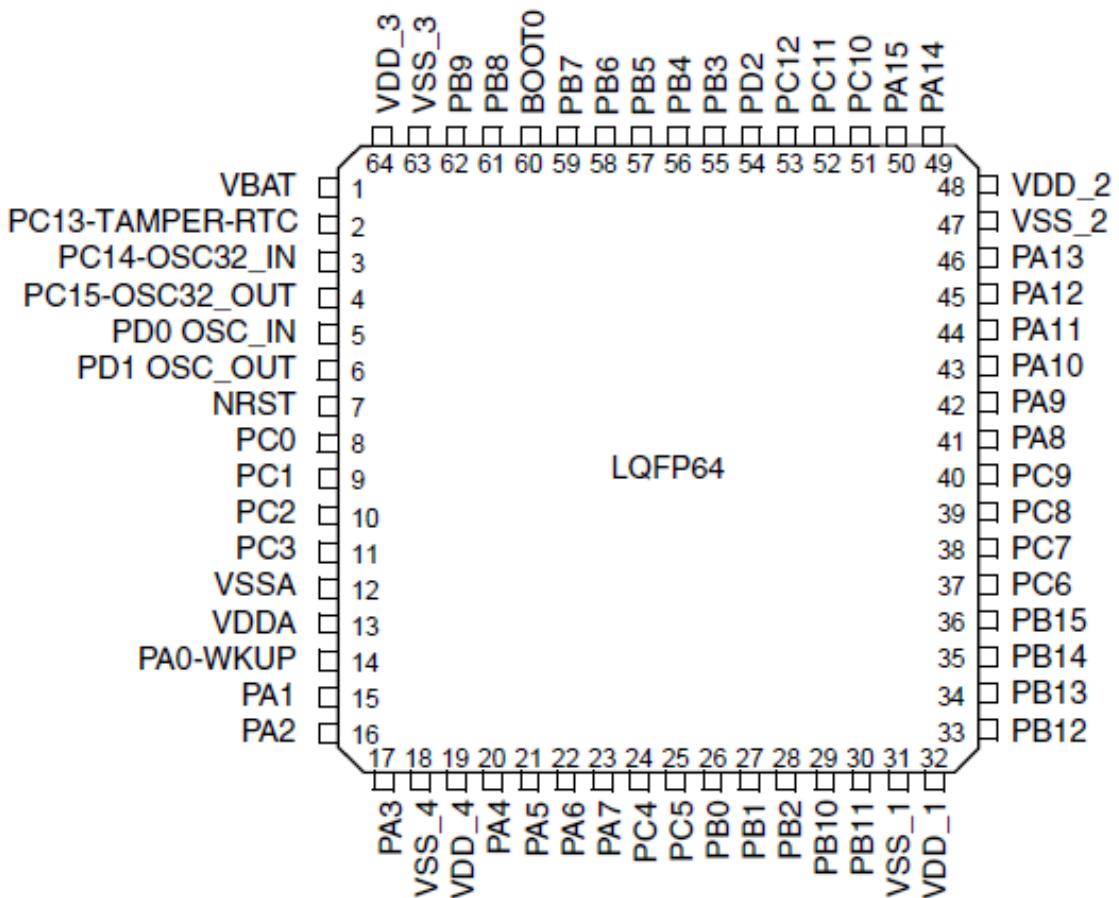
ST STM32 마이크로컨트롤러

- ST STM32 마이크로컨트롤러의 주변 장치 특징
 - 세 개의 Master/Slave SPI 시리얼 인터페이스(42Mbit/s)
 - 프로그램 가능한 워치독(Watchdog) 타이머
 - 두 개의 CAN 인터페이스
 - 한 개의 USB 2.0 full speed 인터페이스
 - 한개의 10/100 Ethernet MAC
 - 8~14bit 병렬 카메라 인터페이스(54 Mbytes/s)

STM32F405 외형과 핀기능

- 디바이스 패키지

- 64pin
- LQFP
- 5개의 범용 입출력 포트 제공



출처 : wikipedia
(www.wikipedia.com)

STM32F405 핀 기능

- 범용 입출력 신호

- 포트A (PA15~PA0)

- 내부 풀업, 풀다운 저항이 있는 16비트 양방향 입출력 단자
 - ADC, UART, SPI, I2C, USB, CAN, ETHERNET, DCMI, JTAG용 단자로 사용

- 포트B (PB15~PB0) :

- 내부 풀업, 풀다운 저항이 있는 16비트 양방향 입출력 단자
 - ADC, UART, SPI, I2C, USB, CAN, ETHERNET, FSMC, DCMI, JTAG용 단자로 사용

- 포트C (PC15~PC0)

- 내부 풀업, 풀다운 저항이 있는 16비트 양방향 입출력 단자
 - ADC, UART, SPI, I2C, USB, ETHERNET, SDIO, DCMI용 단자로 사용

- 포트D (PD2)

- 내부 풀업, 풀다운 저항이 있는 1비트 양방향 입출력 단자
 - UART, I2C, USB, DCMI, 용 단자로 사용

- 포트H (PH0~PH1)

- 내부 풀업, 풀다운 저항이 있는 2비트 양방향 입출력 단자
 - 외부 클럭 입력용 단자로 사용

STM32F405의 기능 핀

- USART, UART
 - 핀 이름 : USARTxTX, USARTxRX, USARTx_CTS, USARTx_RTS, USARTx_CK, USARTxTX, USARTxRX
 - 동기식 및 비동기식 시리얼 통신에 사용되는 핀으로, 총 4개의 USART 포트와 2개의 UART 포트를 제공하며 고속 시리얼 통신을 위해 별도로 RTS, CTS 포트를 함께 제공
- ADC
 - 핀 이름 : VDDA, VSSA, ADCx_IN
 - 최대 12bit의 해상도를 가지며, 입력 신호의 범위는 VDDA의 전압에 의해 결정
- I2C
 - 핀 이름 : I2Cx_SCL, I2Cx_SDA
 - TWI(Two-wire Serial Interface)를 지원하며 기본 모드에서 100kHz, 고속 모드에서 400kHz의 속도를 지원

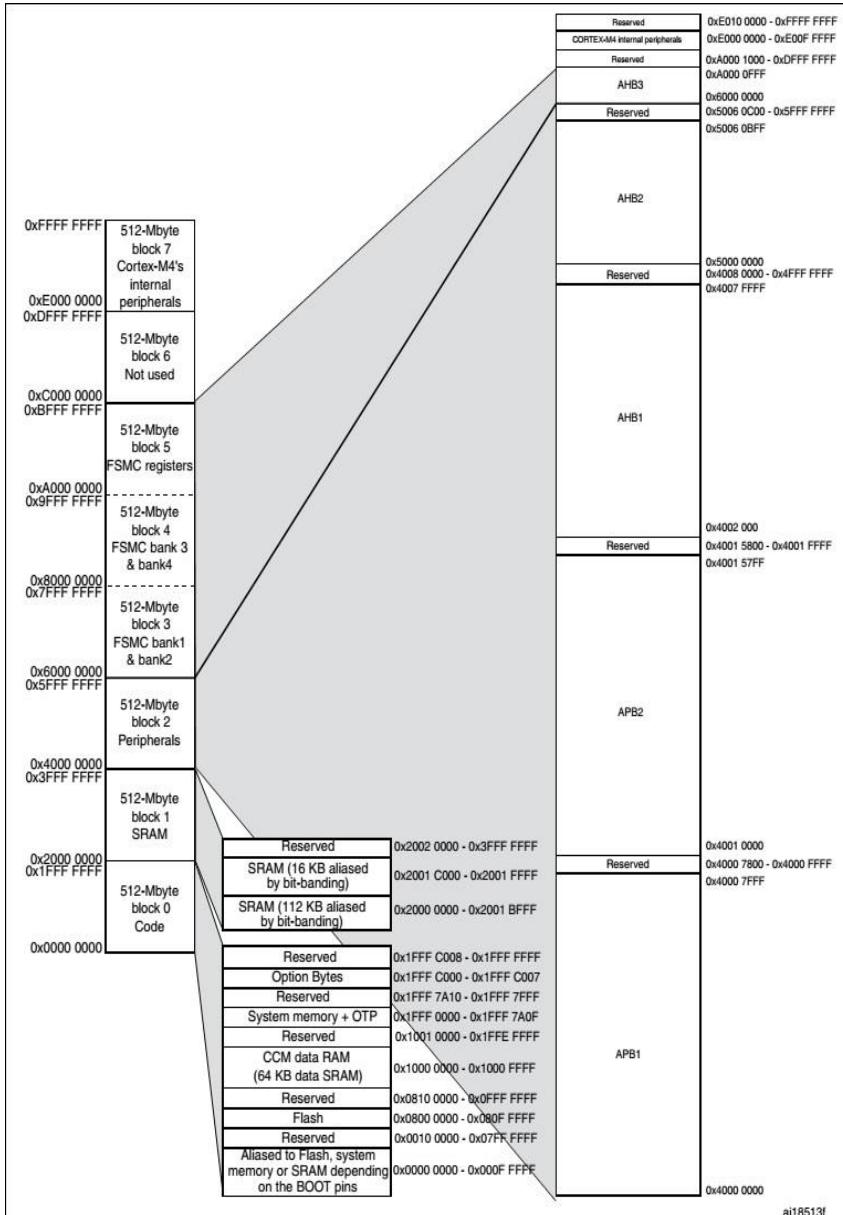
STM32F405의 기능 핀

- SPI
 - 핀 이름 : SPIx_SCK, SPIx_MISO, SPIx_MOSI, SPIx NSS
 - SPI (Serial Peripheral Interface : 직렬 주변 장치 인터페이스)를 지원하며 half / full duplex를 지원하며 8 or 16bit 포맷을 선택하여 통신, 최대 42MHz의 속도를 지원
- CAN
 - 핀 이름 : CANTX, CANRX
 - CAN(Controller area network)는 주로 자동차의 전장에 사용되는 시리얼 인터페이스이며, CAN Protocol 2.0 A, B를 모두 지원, 최대 1Mbits/s의 속도를 지원하며 Basic, FIFO, Enhanced Full CAN을 모두 지원
- USB
 - 핀 이름 : OTG_FS_ , OTG_HS_
 - USB (Universal Serial Bus)는 최근 컴퓨터의 주변 장치의 표준 인터페이스로 그 사용이 증대되고 있는 시리얼 인터페이스로 STM32F405는 USB 2.0 Full Speed를 지원

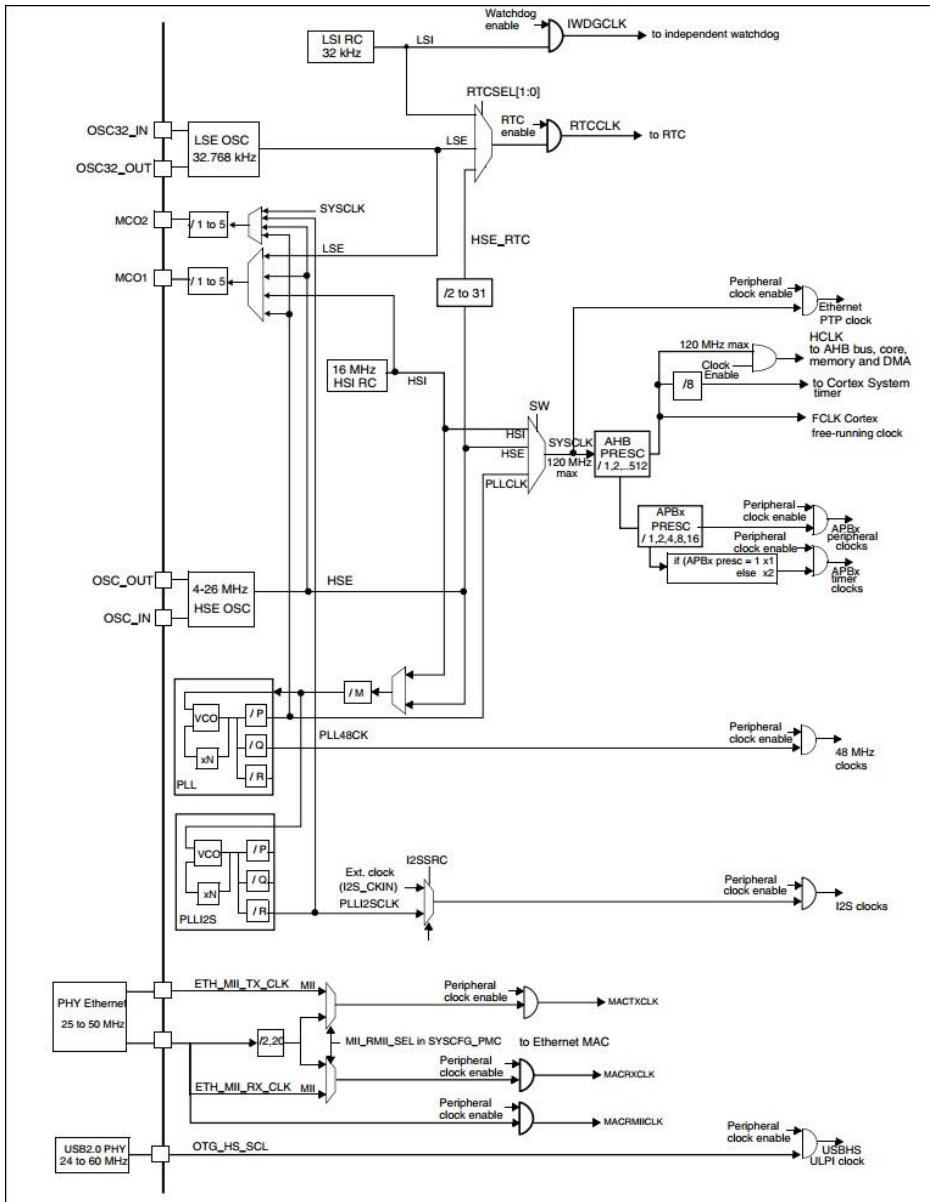
STM32F405의 기능 핀

- ETHERNET
 - 핀 이름 : ETH_
 - IEEE 1588 v2 지원 기능을 제공하는 이더넷 MAC10/100
- FSMC (Flexible static memory controller)
 - 핀 이름 : FSMC_
 - PCCard/Compact Flash, SRAM, PSRAM, NOR Flash and NAND Flash 지원
 - LCD parallel interface 지원
- SDIO (Secure digital input/output interface)
 - 핀 이름 : SDIO_D0~D7, SDIO_CK, SDIO_CMD,
 - SDIO 인터페이스를 지원하며 최대 48MHz의 속도를 지원
- DCMI (Digital Camera Interfacee)
 - 핀 이름 : DCMI_
 - 디지털 카메라 인터페이스를 지원하며 8 비트 또는 14 비트 병렬 인터페이스를 통해 비디오 데이터를 수신 받을수 있다. 최대 54M/s의 속도를 지원

STM32F405의 메모리 구조



STM32F405의 클럭



STM32F405의 클럭

- 메인 시스템 클럭인 SYSCLK는 선택하여 사용
 - HSE(High speed external clock signal)
 - 외부로부터 입력되는 클럭으로 크리스탈이나 오실레이터를 통해 입력
 - HSI(High speed internal clock signal)
 - 내부의 16MHz RC 오실레이터에서 제공하는 클럭으로 이를 직접 시스템 클럭으로 사용하거나 PLL 블록을 통해 원하는 클럭으로 증감시켜 사용
- 메인 시스템 클러과 별도로 저속의 클럭 소스를 제공
 - LSE(Low speed external clock signal)
 - 32.768kHz의 Low speed의 클럭을 외부 크리스탈이나 오실레이터를 통해 입력받으며 이는 저전력 또는 높은 정밀도를 요하는 RTC 또는 타이머의 클럭으로 사용
 - LSI(Low speed internal clock signal)
 - 저전력 설계를 위해 사용할 수 있도록 대략 32kHz의 내부 클럭을 제공

CORTEX-M4 마이크로 컨트롤러 개발환경

- 개발환경 설치
- EWARM 처음 실행



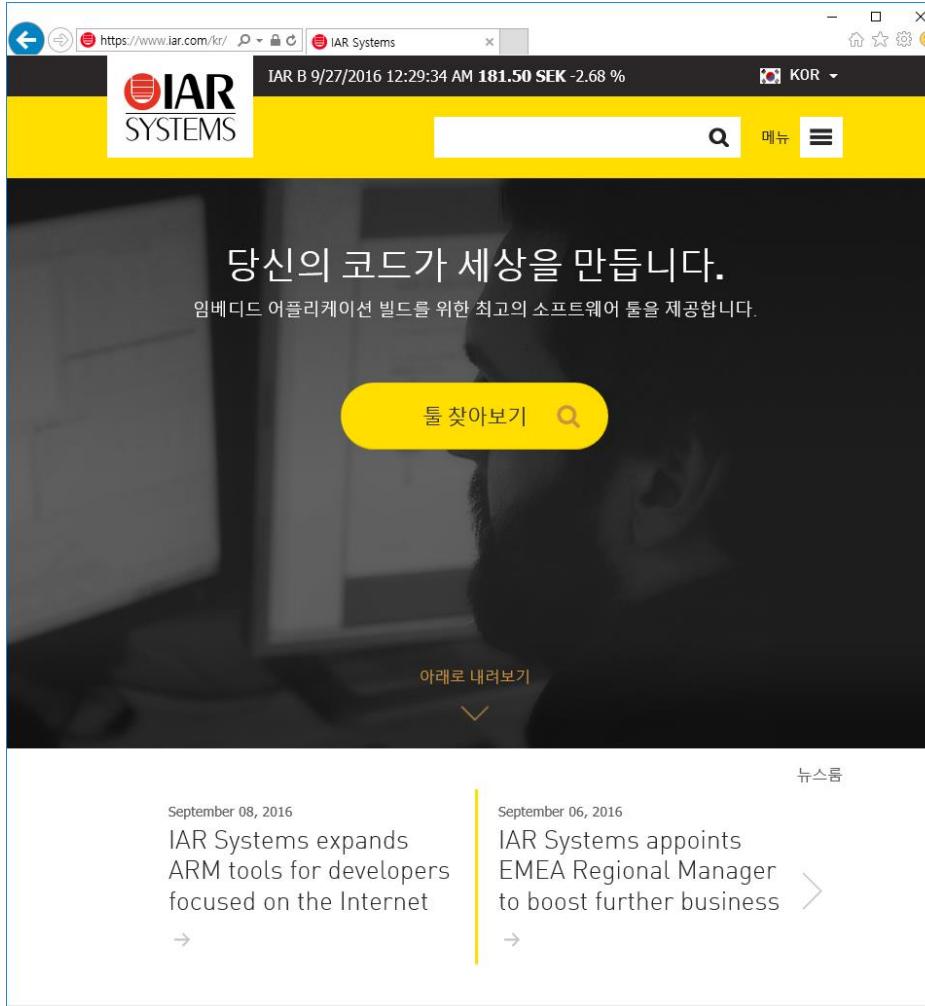
엣지아이랩

マイクロ 컨트롤러의 개발환경

- 개발 환경
 - 마이크로 컨트롤러를 사용하는 임베디드 시스템에 필요한 일련의 도구(tool)들
 - 개발 도구(tool)의 종류
 - 컴파일러 : CPU에서 동작하기 위한 프로그램을 기계어로 변환
 - 디버깅툴 : 프로그램의 오류를 찾아내기 위한 도구
 - ISP툴 : 내장된 프로그램 메모리에 프로그램을 적재시킬 도구
- Cortex-M4 마이크로 컨트롤러의 개발 환경
 - IAR Embedded Workbench for ARM(EWARM)
 - IAR 사에서 제공하는 개발환경 소프트웨어
 - Cortex-M4 설계에 필요한 코드 작성, 컴파일, 다운로드 등 모든 기능을 하나의 소프트웨어에서 제공하는 통합 개발 환경

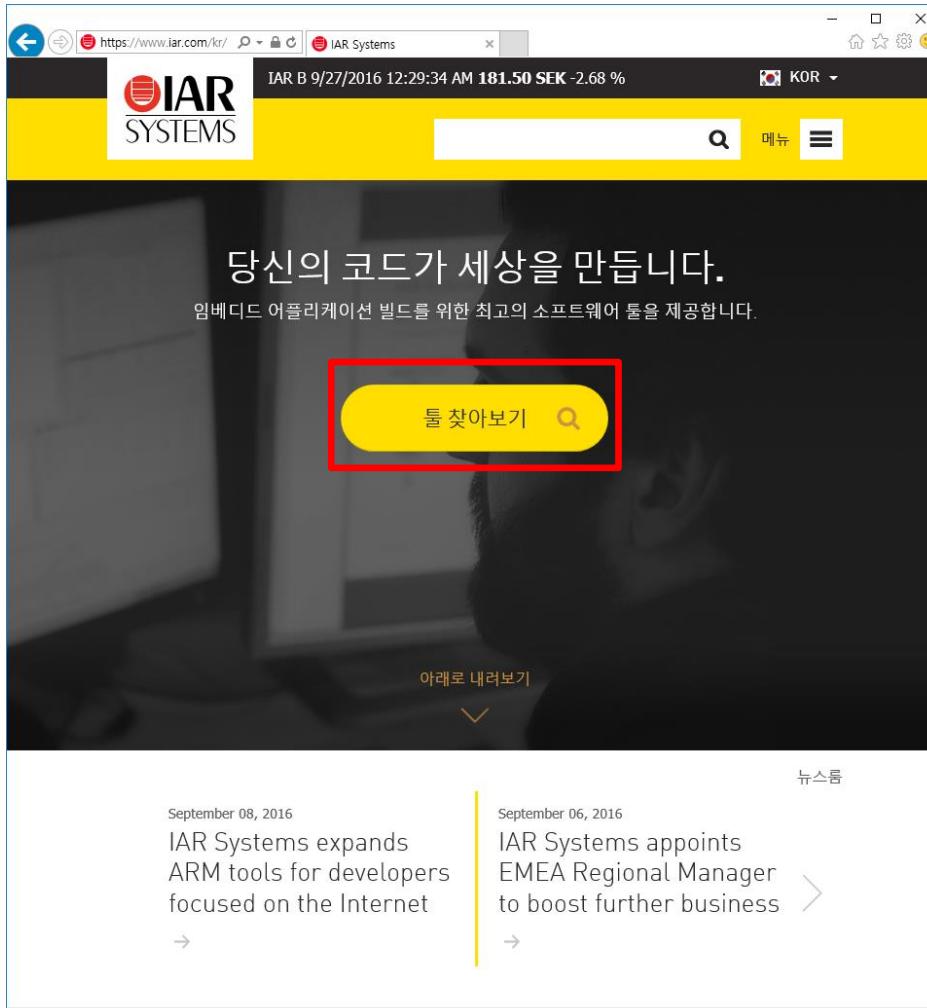
개발환경 설치

- IAR내려받기
 - IAR Systems의 홈페이지 (<http://www.iar.com>)



개발환경 설치

- “툴 찾아보기” 클릭



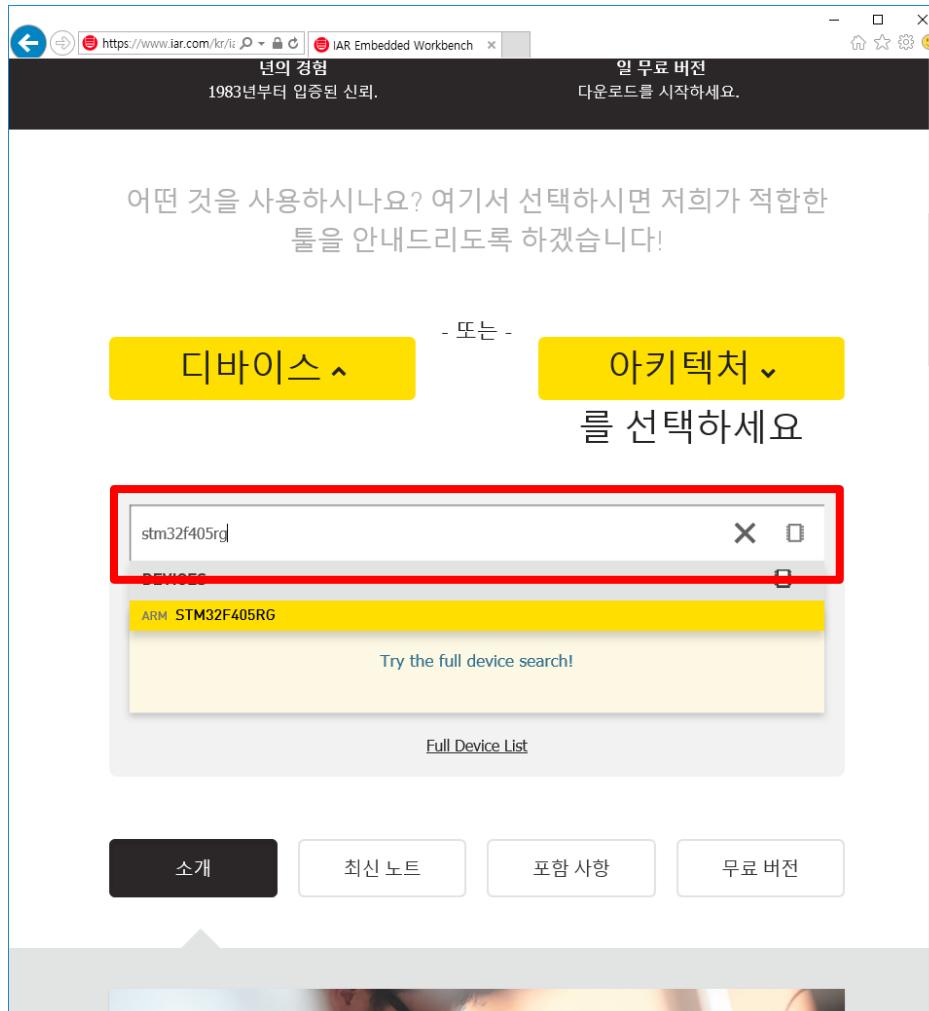
개발환경 설치

- “디바이스” 클릭



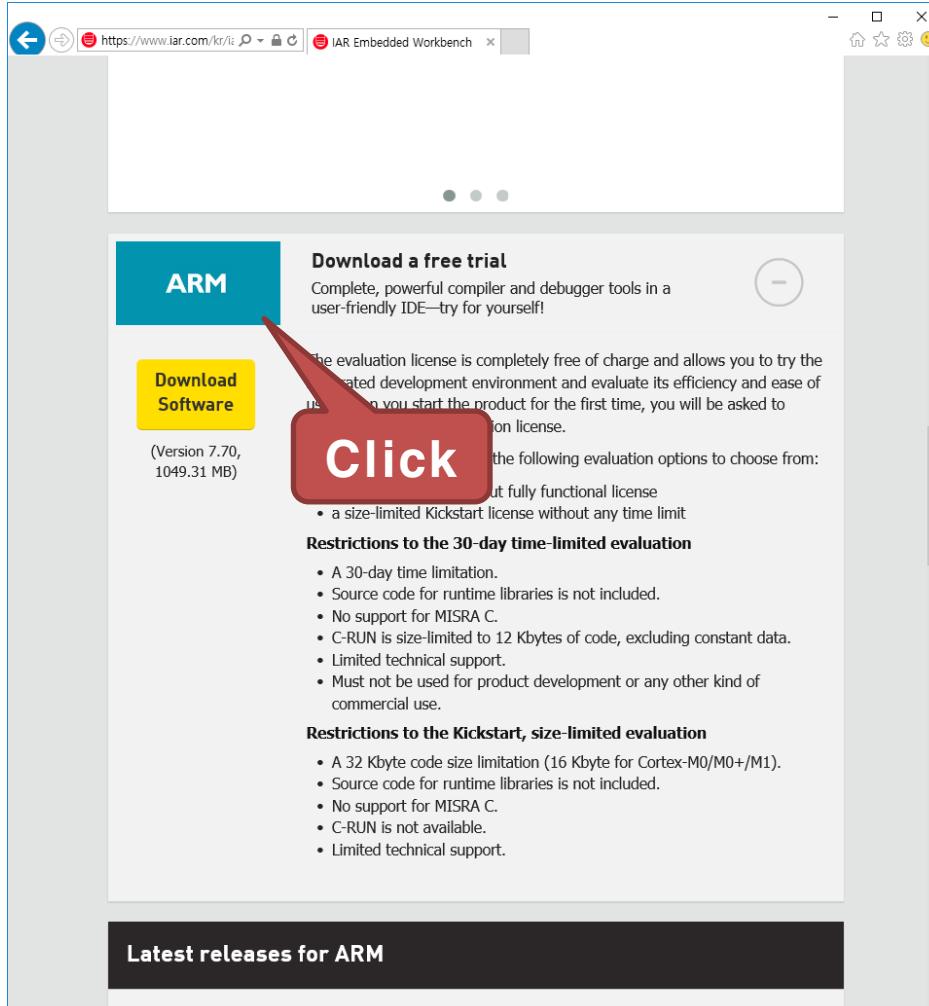
개발환경 설치

- 빙칸에 “stm32f405rg”를 입력



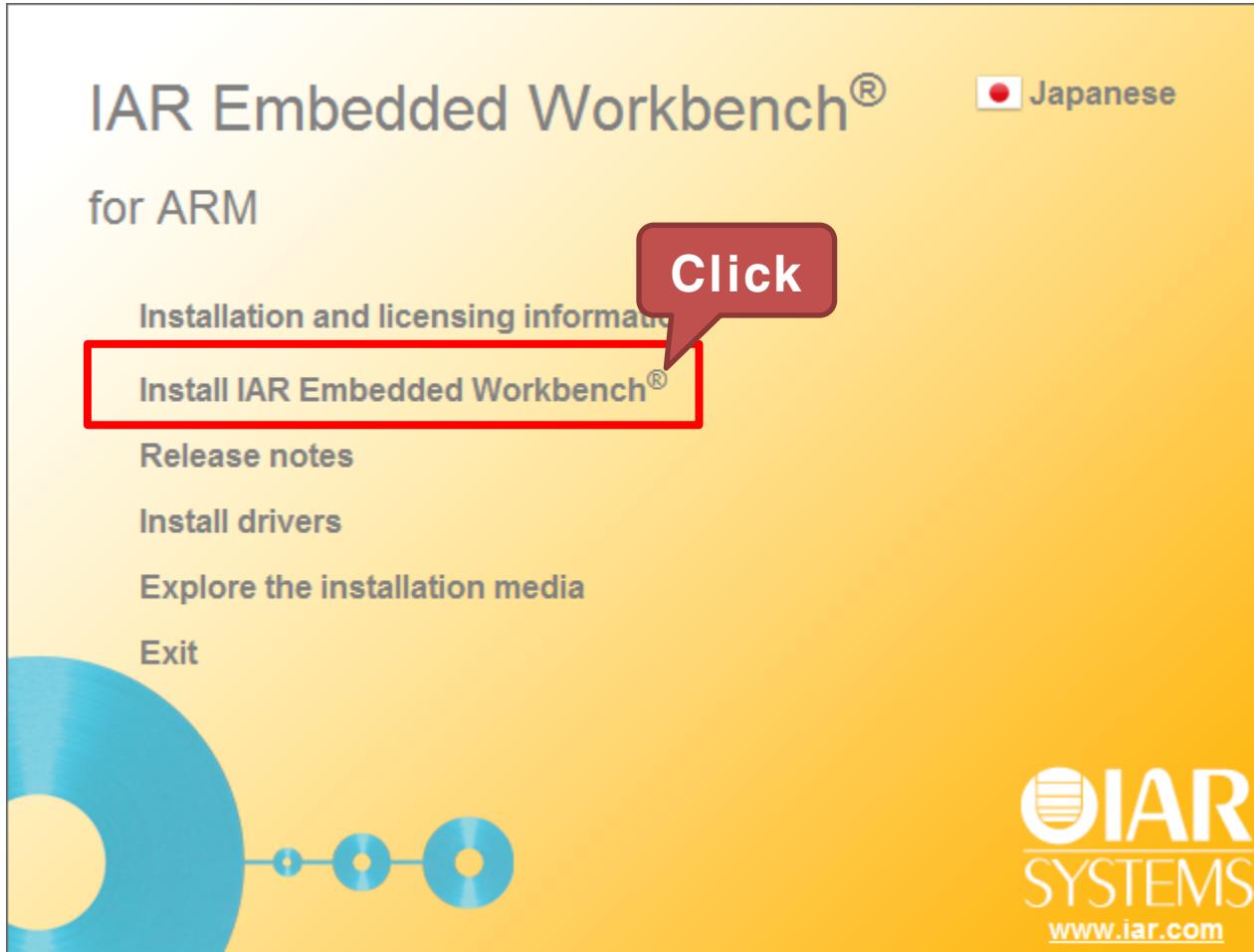
개발환경 설치

- 페이지 중간부분에서 “ARM” 아이콘을 클릭
- 그림과 같이 다운로드 아이콘이 나타나면, 설치파일 다운로드



개발환경 설치

- 다운로드한 설치파일을 실행 후, Install the IAR Embedded Workbench를 선택



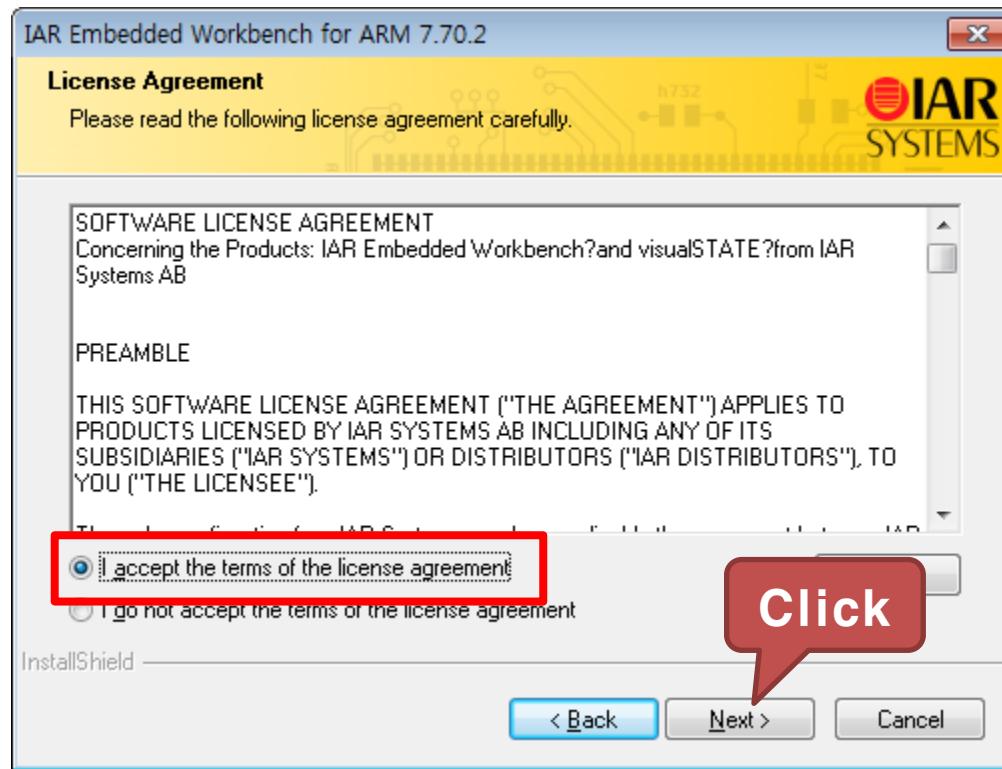
개발환경 설치

- 초기 설치화면에서 “Next”를 클릭



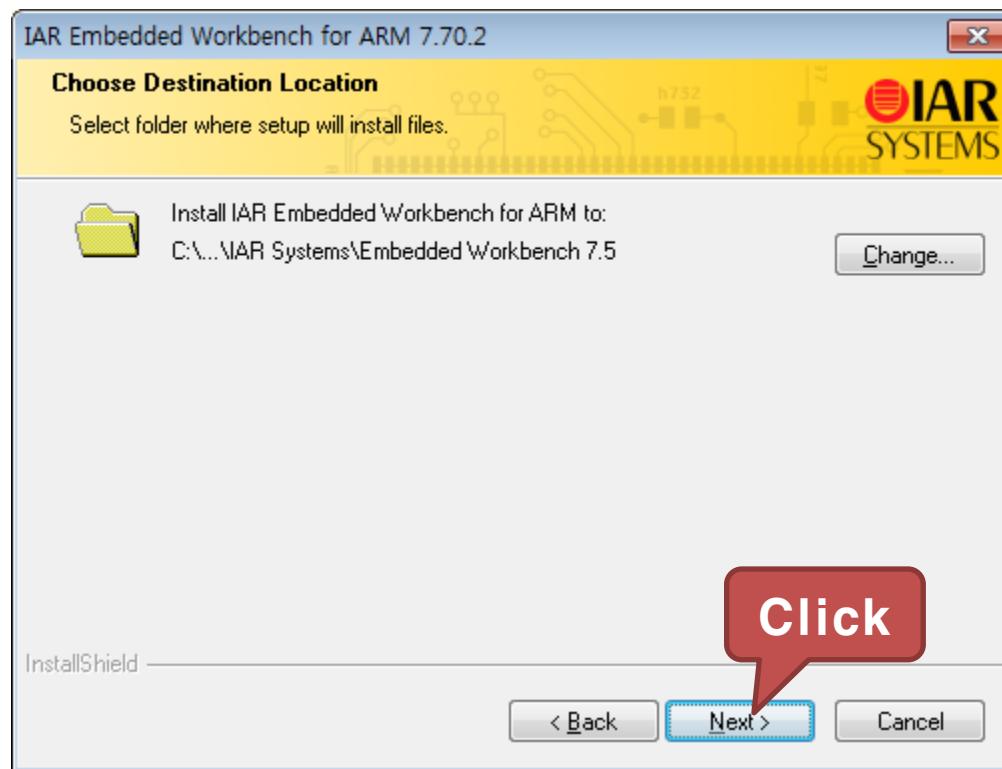
개발환경 설치

- 라이센스에 동의하고 “Next”를 클릭



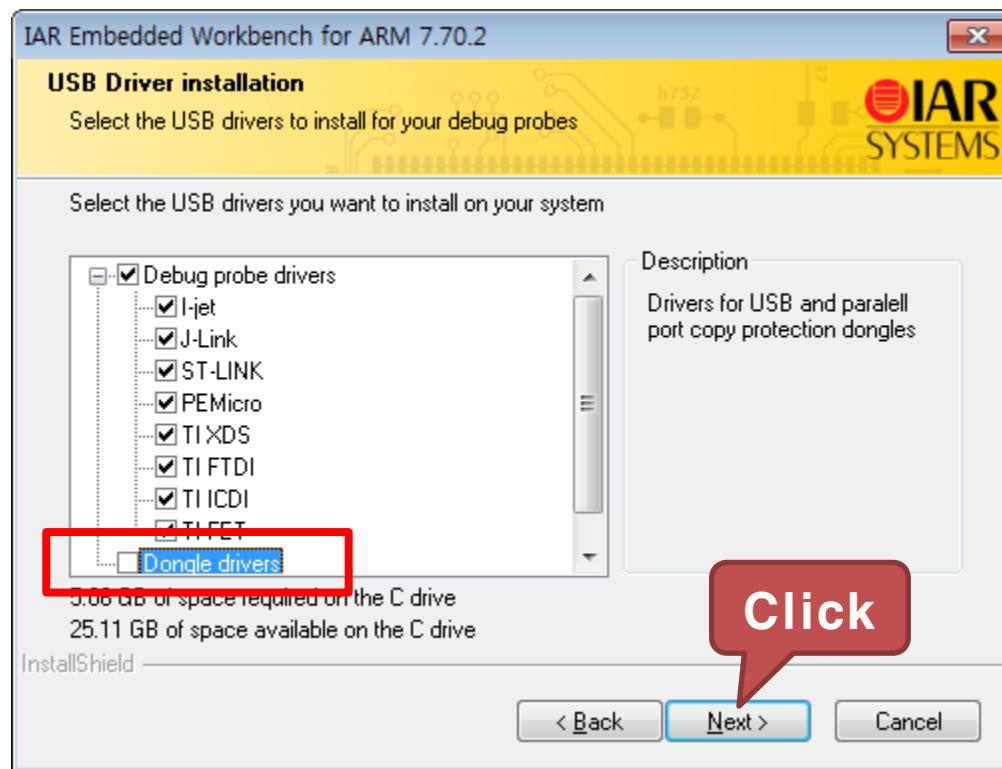
개발환경 설치

- 설치할 폴더를 설정하고 “Next” 클릭



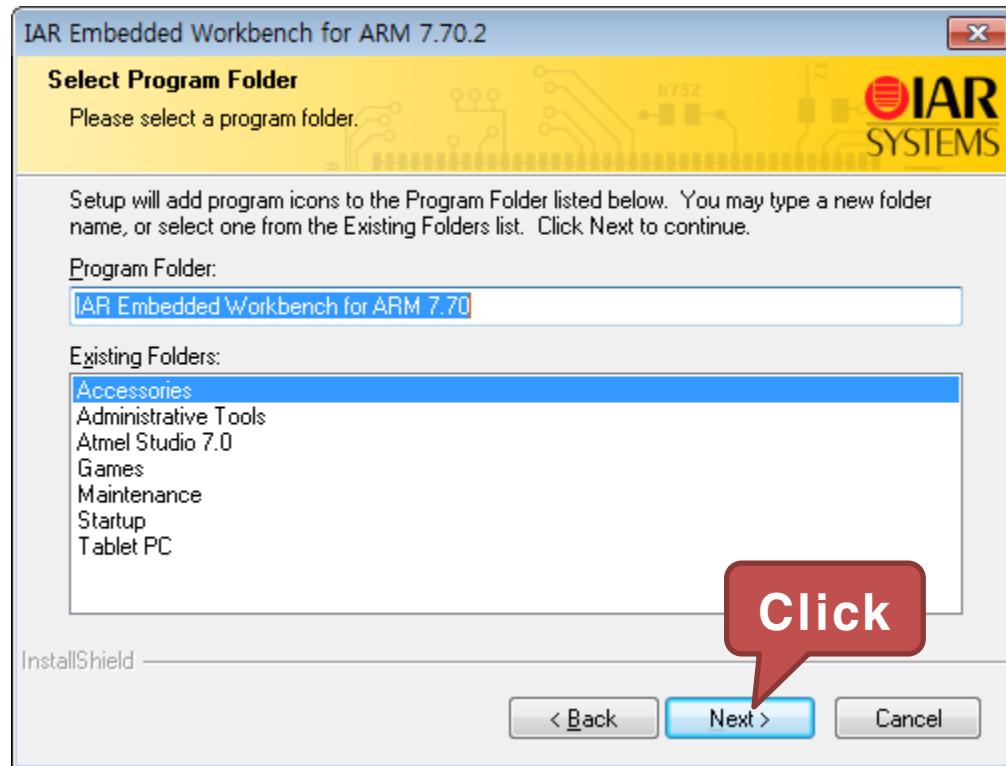
개발환경 설치

- 다음과 같이 “Dongle drivers” 는 체크 해제하고, “Next”를 선택



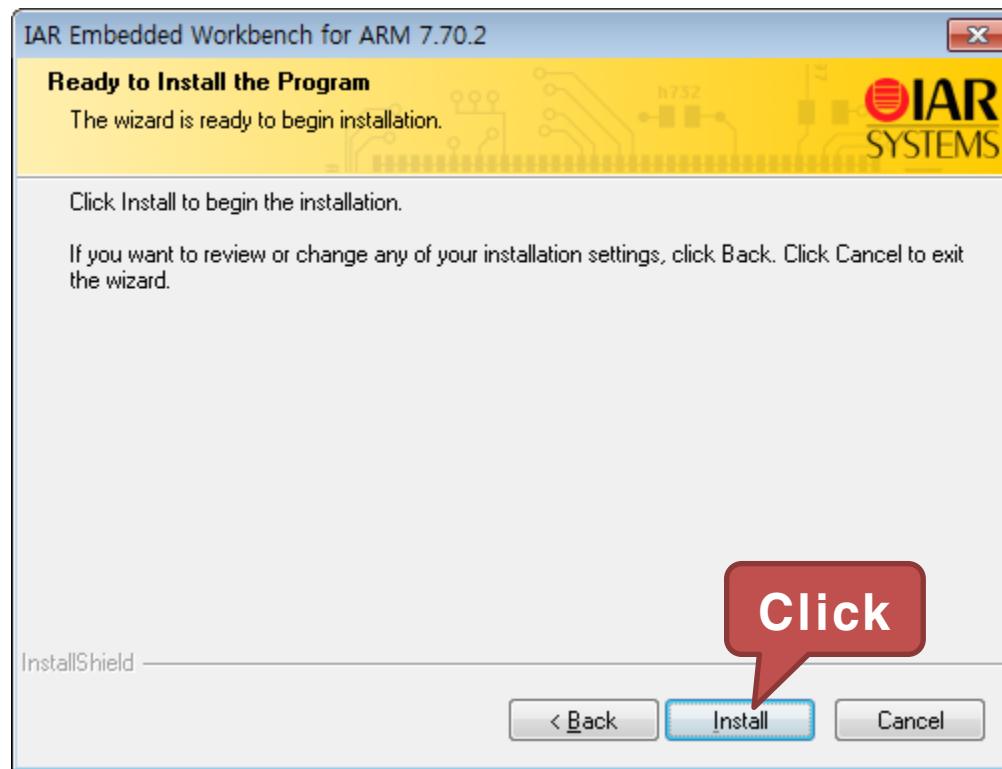
개발환경 설치

- 다음 단계는 프로그램 이름을 설정하는 단계, 초기설정 그대로 해서 "Next"를 선택



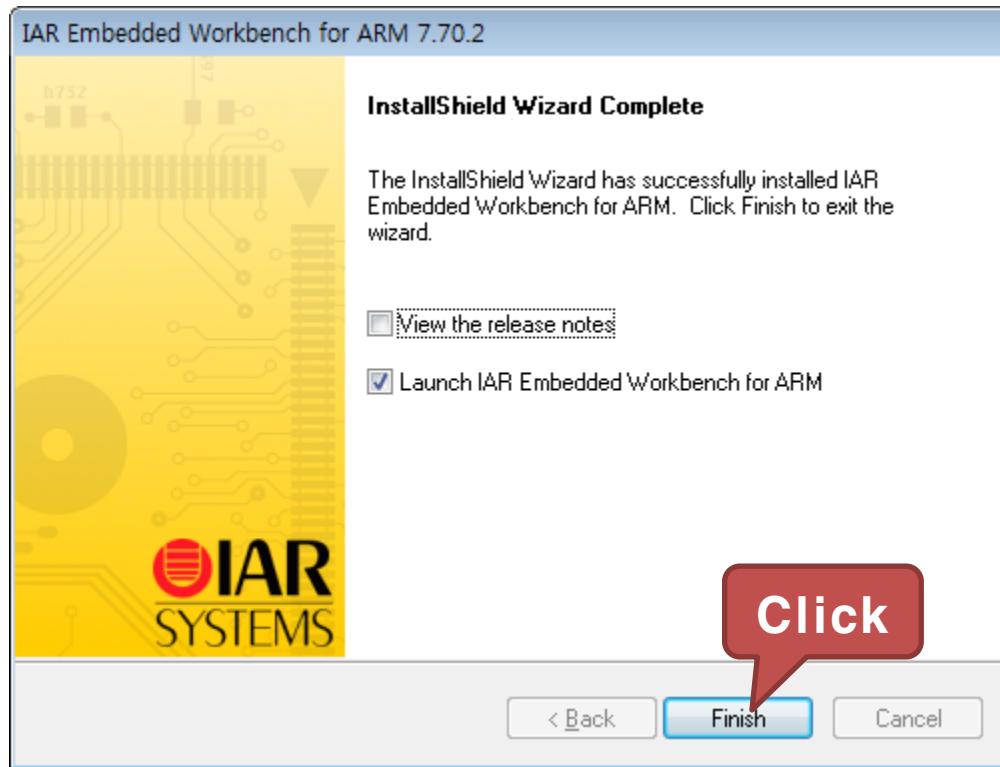
개발환경 설치

- 지금까지 설정대로 설치하려면 “Install” 클릭
- 설치 진행되며 설치에는 수 분의 시간이 소요됨



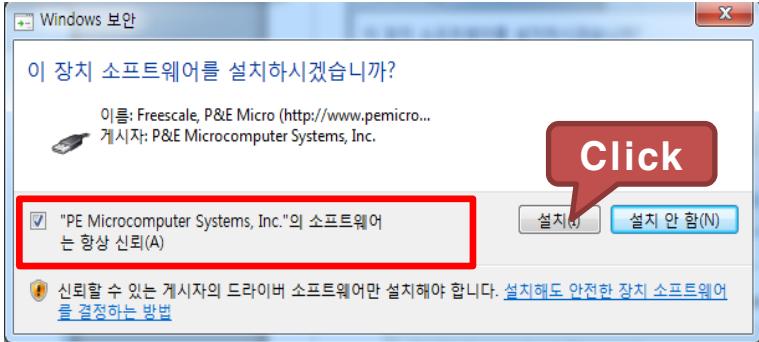
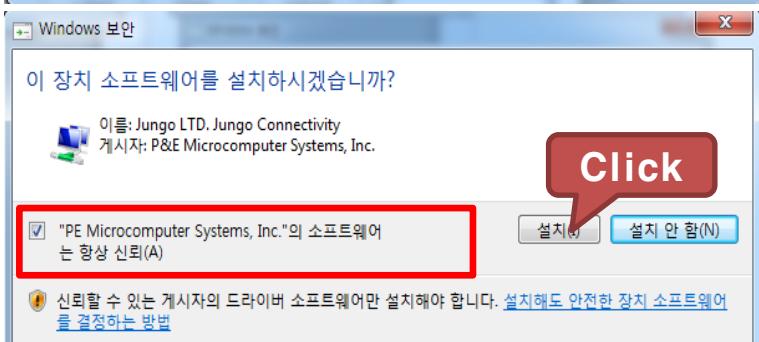
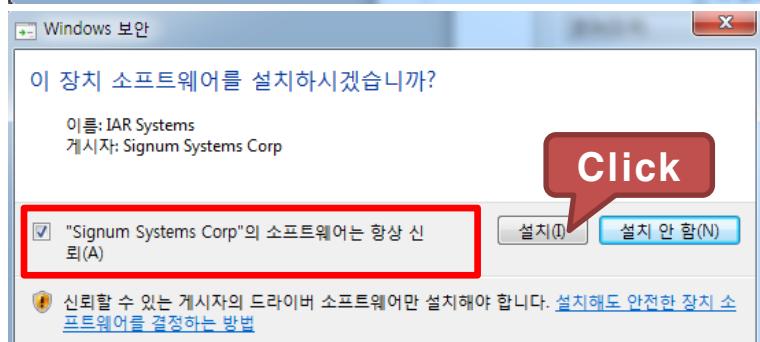
개발환경 설치

- 그림과 같이 “Launch IAR Embedded Workbench for ARM” 을 선택
- 종료 버튼을 누르면 IAR EWARM(Embedded Workbench for ARM)이 실행



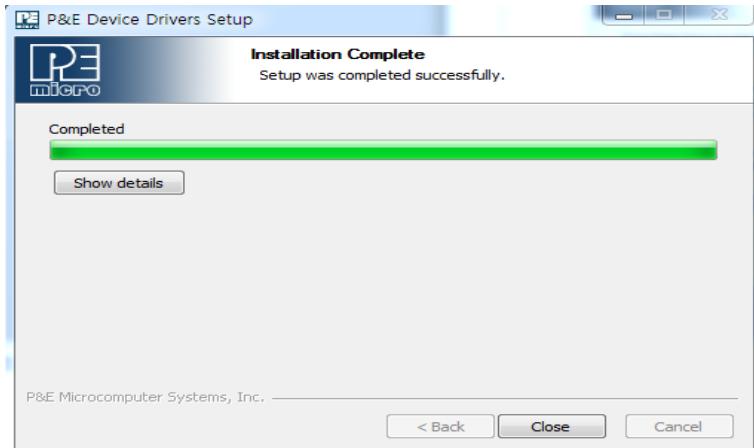
개발환경 설치

- 프로그램 중간에 드라이버 설치창이 뜨면 모두 체크하고 “설치” 클릭



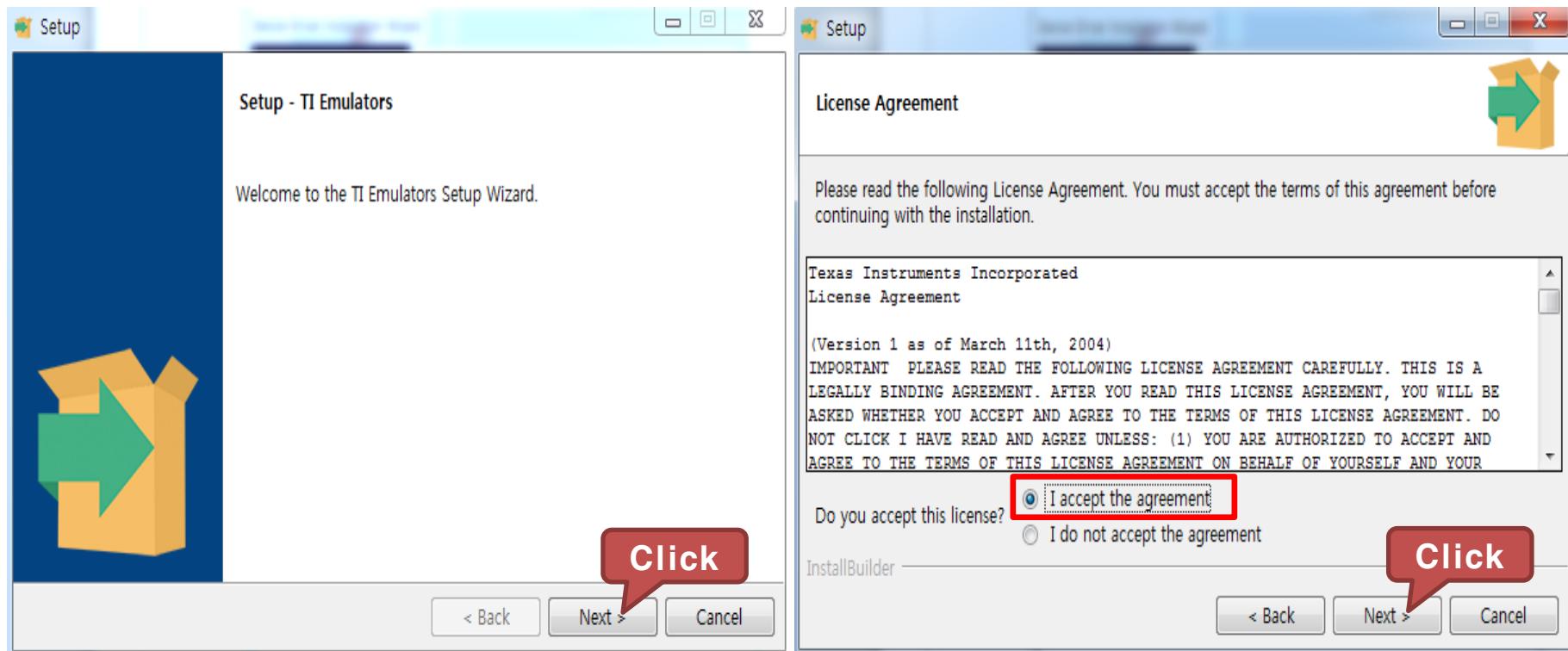
개발환경 설치

- 디바이스 드라이버 설치 창이 뜨면, 다음과 같이 클릭



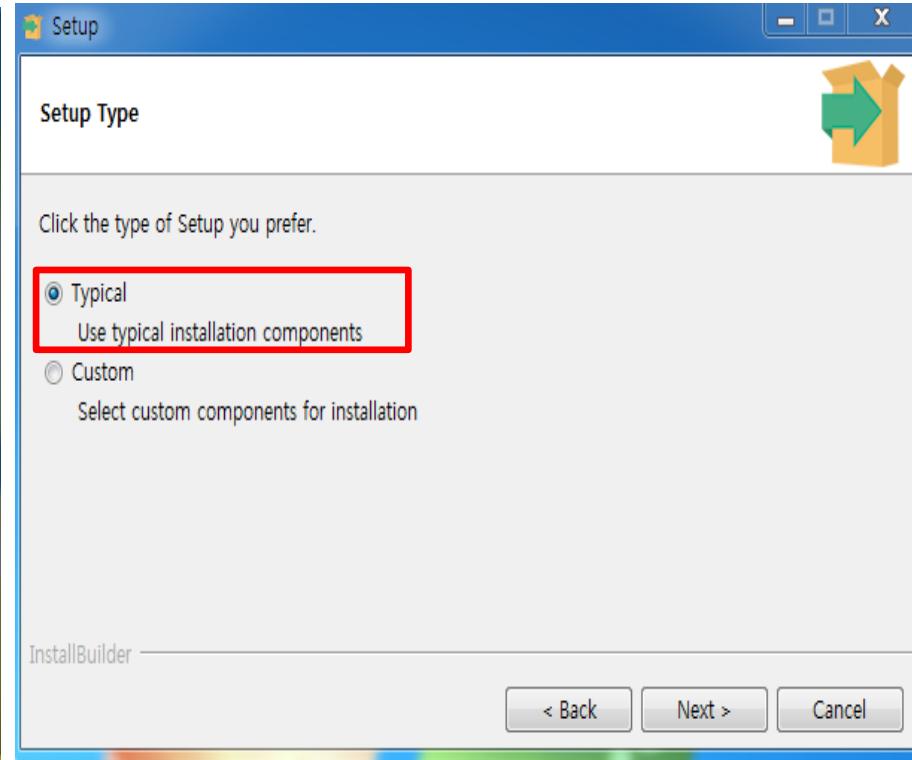
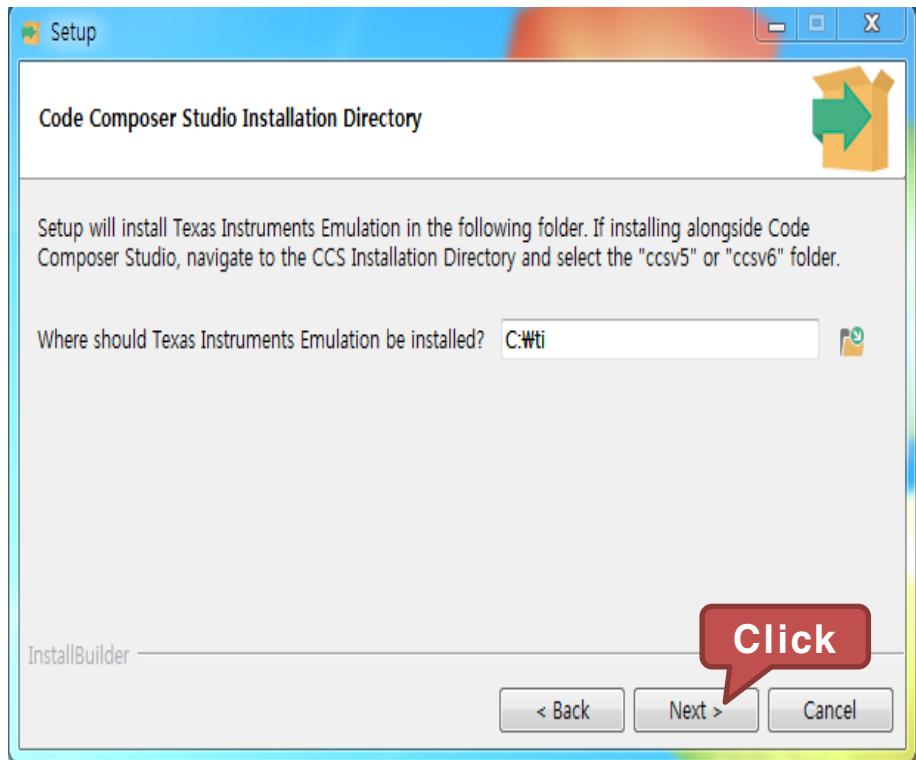
개발환경 설치

- TI Emulator 설정 창이 뜨면, 다음과 같이 “Next” 클릭
- 라이선스 동의 후, “Next” 클릭



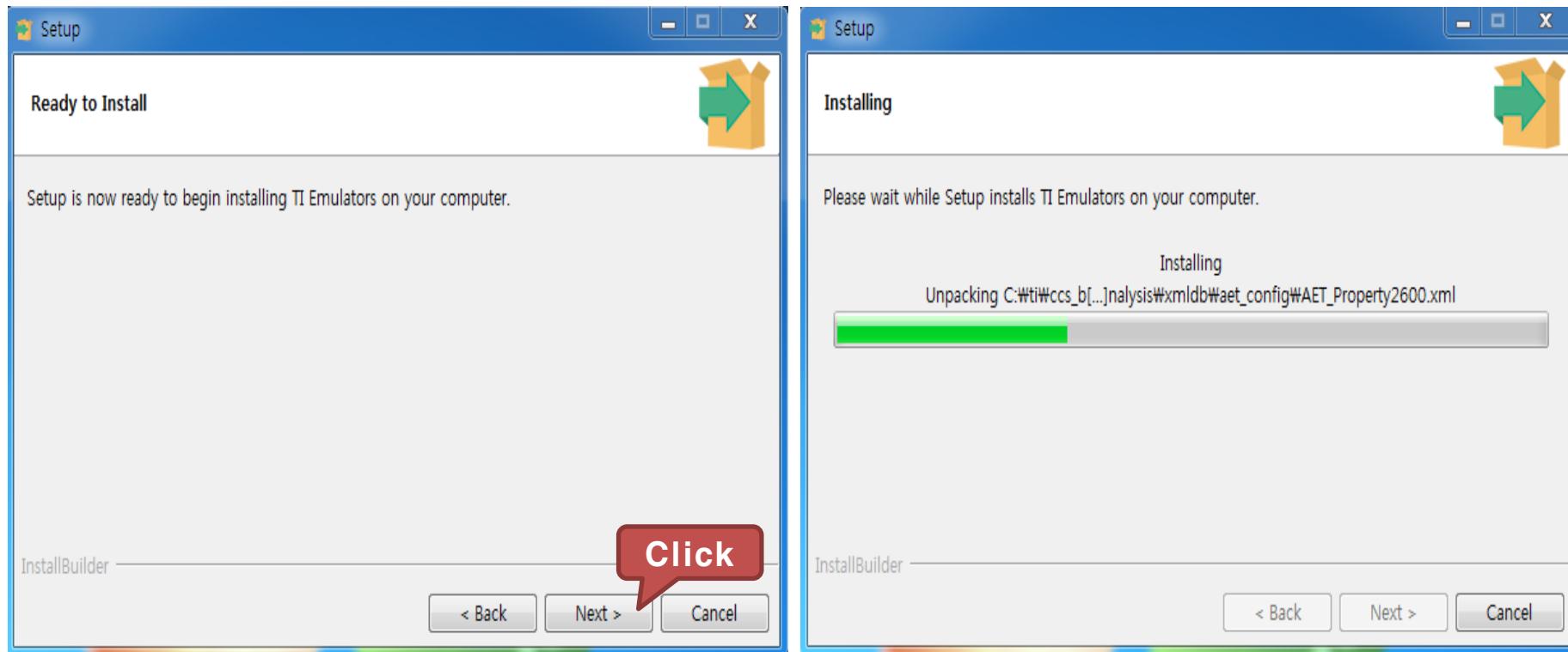
개발환경 설치

- 다음과 같이 “Typical” 선택하고 “Next” 클릭



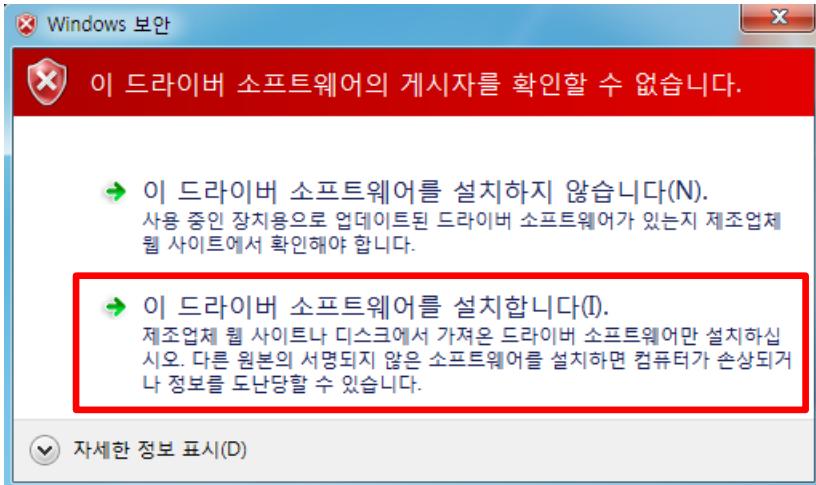
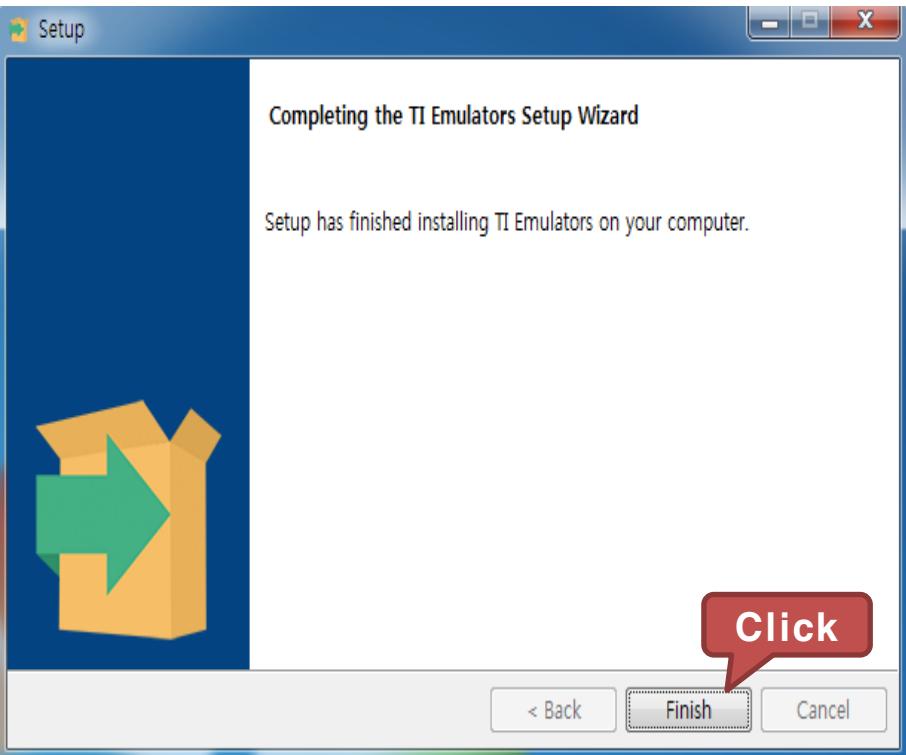
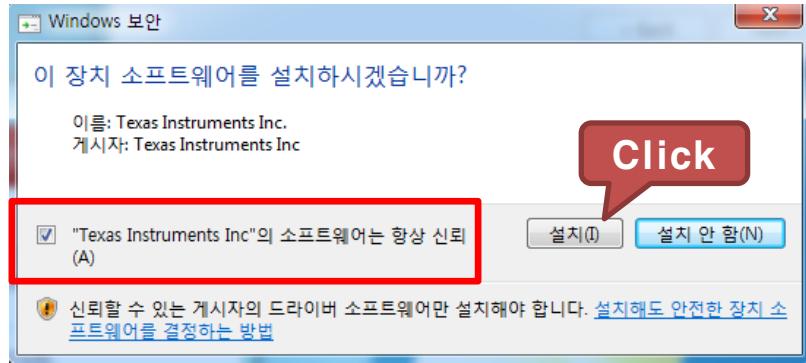
개발환경 설치

- “Next” 클릭



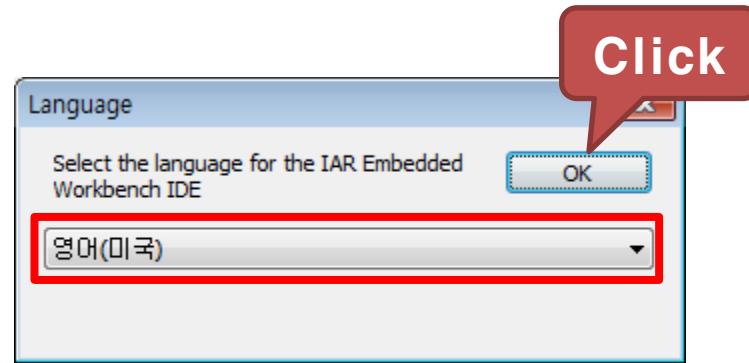
개발환경 설치

- 설치 중간에 드라이버 설치 및 경고창이 뜨면, 다음과 같이 클릭
- 설치 완료되면 “Finish” 클릭



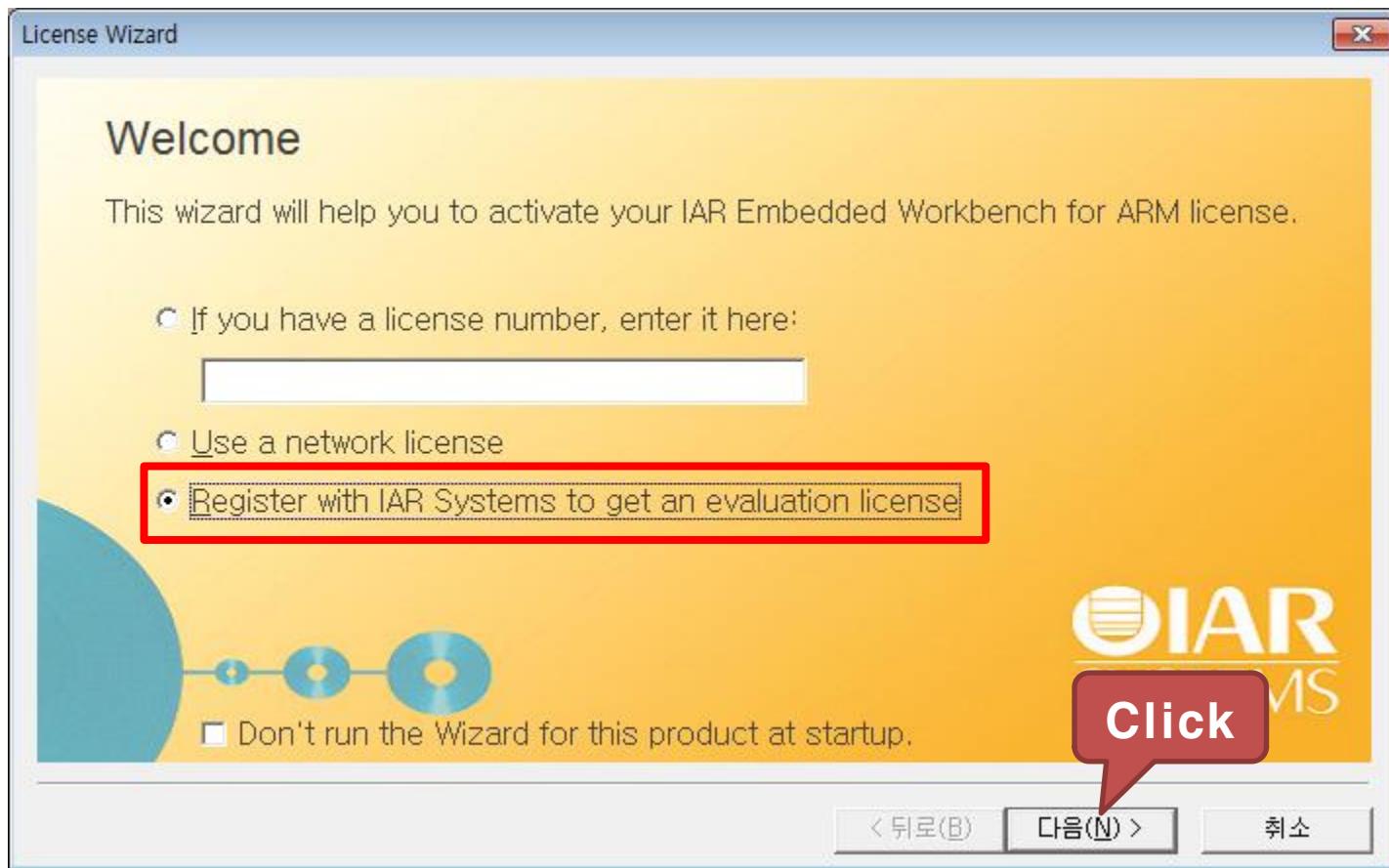
개발환경 설치

- 다음과 같이 언어 선택창이 나타나면 “영어”를 선택하고 “OK”를 클릭



개발환경 설치

- 언어 선택이 끝나면 IAR EWARM은 라이선스를 확인
- 라이선스가 있다면 입력하고 다음으로 넘어감
- 라이선스가 없다면 다음과 같이 IAR Systems 에 라이선스를 요청



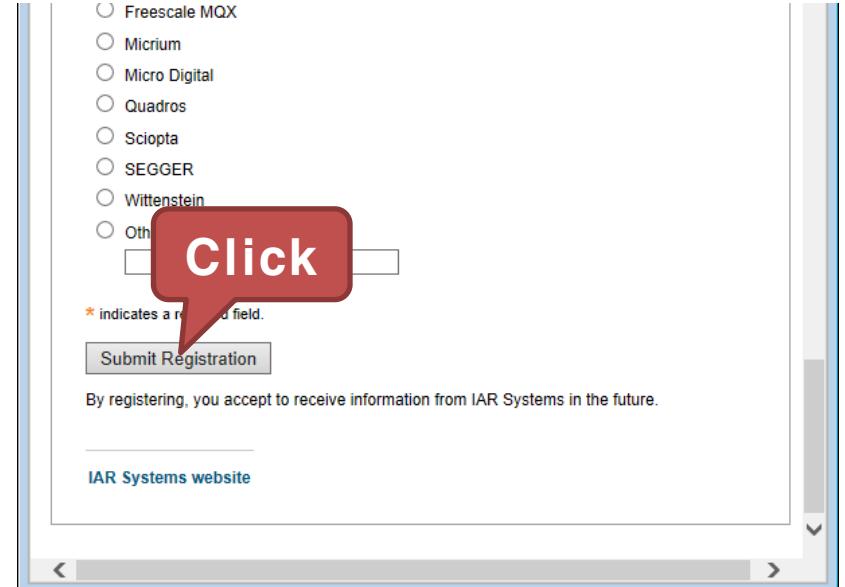
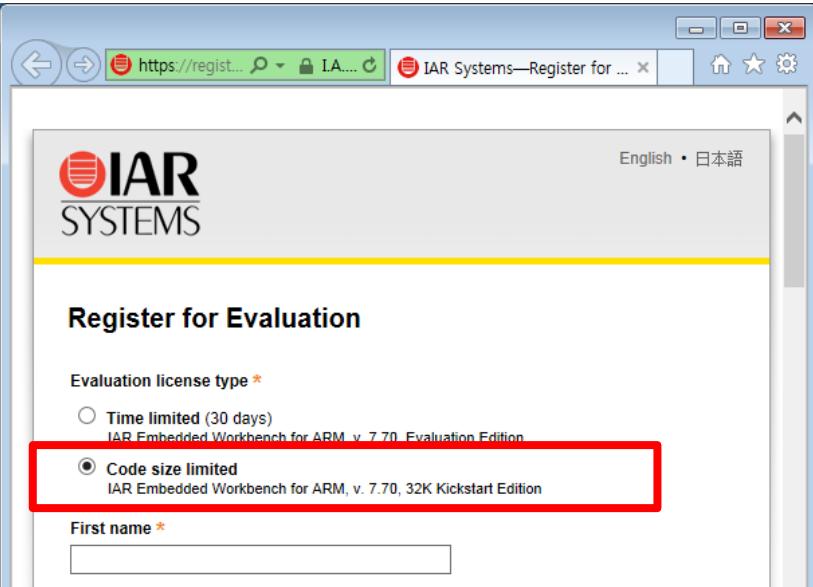
개발환경 설치

- 우선 라이선스를 요청하기 위해 다음과 같이 “Register”를 클릭



개발환경 설치

- 라이선스 요청을 위한 등록 페이지에서 다음과 같이 “Code size limited” 을 선택
- 입력 칸을 채운 후 “Submit Registration” 클릭하면 라이선스 요청이 완료
- 등록 요청에 문제가 없으면 다음과 같이 앞서 입력한 자신의 메일 주소로 “Confirm Registration” 메일이 발송



The image shows two screenshots of the IAR Systems registration page. The left screenshot shows the 'Register for Evaluation' form with the 'Evaluation license type' section. The 'Code size limited' option is selected and highlighted with a red box. The right screenshot shows the same form with a red callout bubble pointing to the 'Submit Registration' button, which is highlighted with a red box.

Register for Evaluation

Evaluation license type *

Time limited (30 days)
IAR Embedded Workbench for ARM, v. 7.70, Evaluation Edition

Code size limited
IAR Embedded Workbench for ARM, v. 7.70, 32K Kickstart Edition

First name *

English • 日本語

Freescale MQX
Micrium
Micro Digital
Quadros
Sciopta
SEGGER
Wittenstein
Other

* indicates a required field.

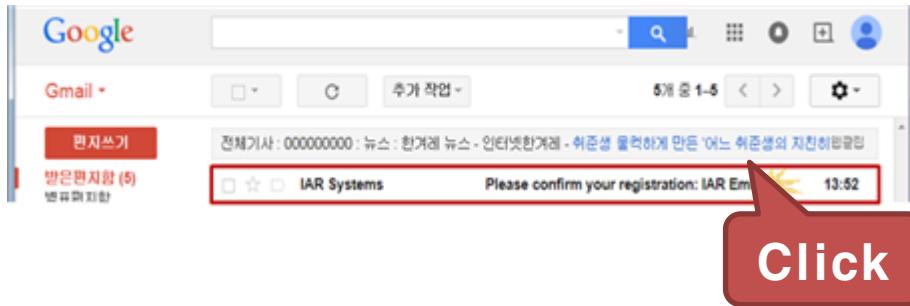
Submit Registration

By registering, you accept to receive information from IAR Systems in the future.

IAR Systems website

개발환경 설치

- 앞에서 입력한 메일로 로그인을 하여 그림과 같이 IAR Systems가 보낸 메일을 확인



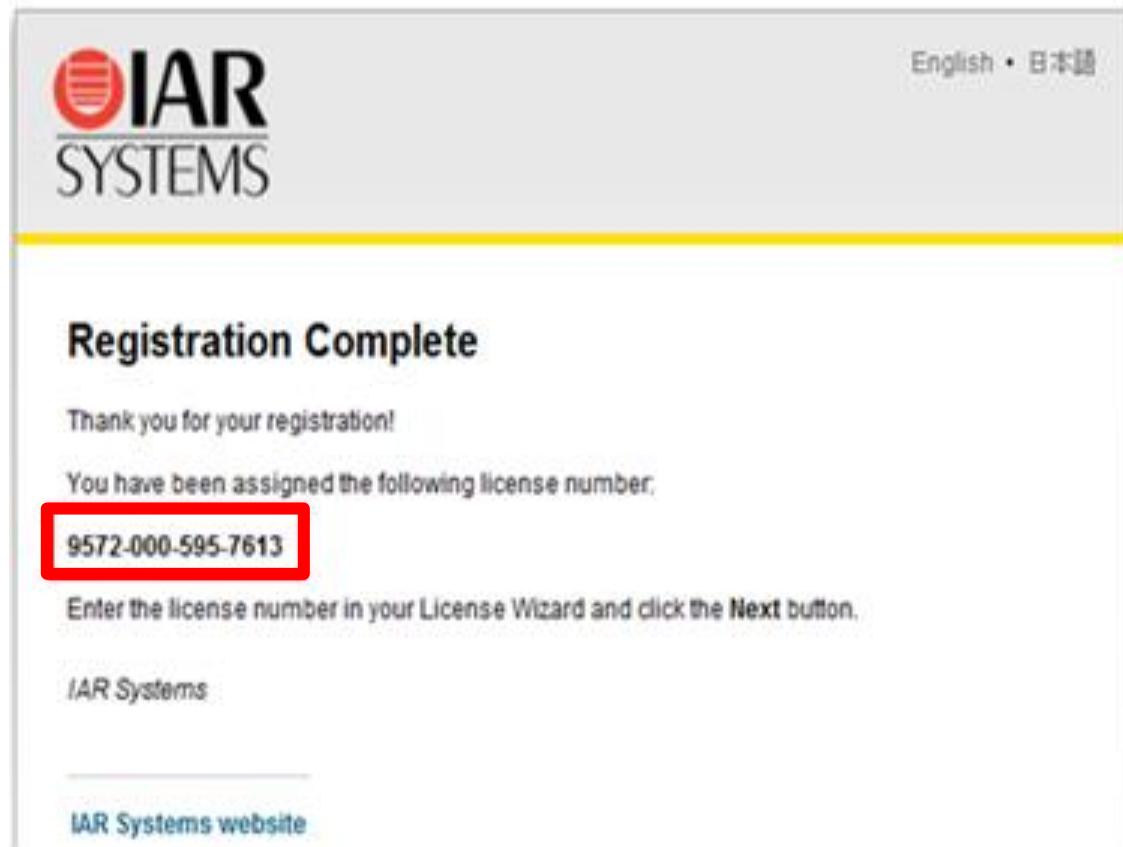
- 메일 내용 중 다음과 같이 Link된 주소를 클릭한다.

Dear Developer,
Please confirm your web registration for the product
IAR Embedded Workbench for ARM, v. 7.70, 32K Kickstart Edition
using this link
<https://register.iar.com/confirm?lang=en&key=14b6b-9492-6689977acc87>
Unconfirmed registrations are erased from our system after 14 days.

A red speech bubble labeled 'Click' points to the URL in the email body.

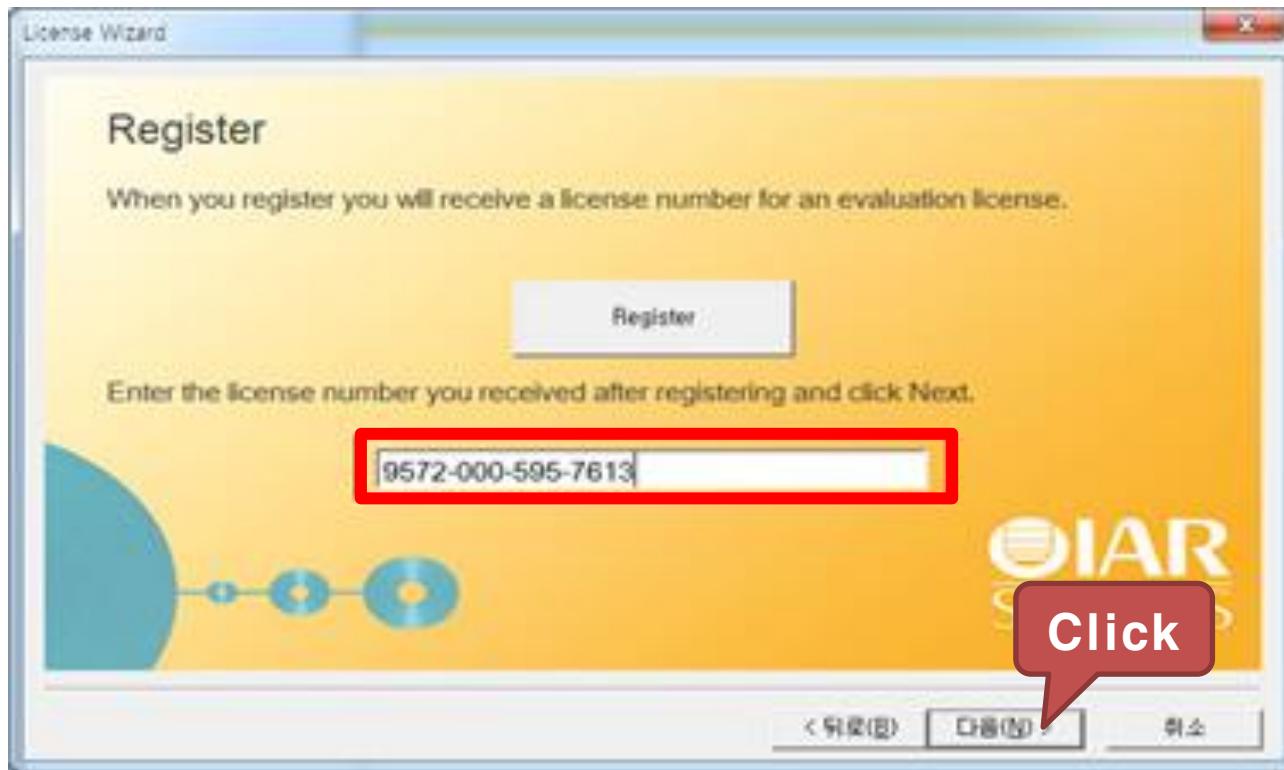
개발환경 설치

- 링크 주소로 이동하면 할당된 라이선스 번호가 있으며, 라이선스 번호는 한 번만 입력하면 되므로 복사



개발환경 설치

- 라이선스 등록창의 아래쪽 빈칸에 할당 받은 라이선스 번호를 붙여 입력



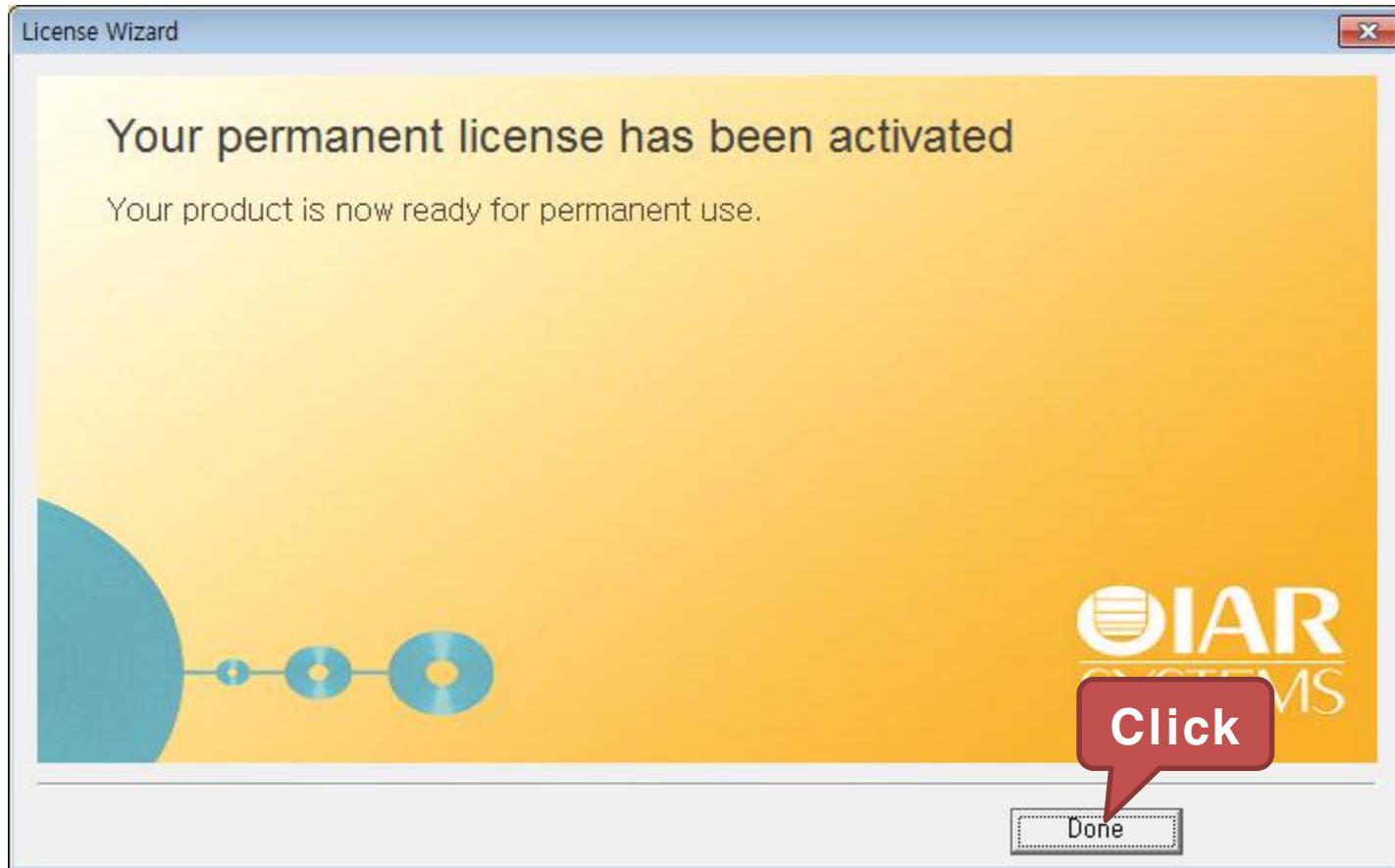
개발환경 설치

- 라이선스의 정보가 표시, “다음”을 클릭



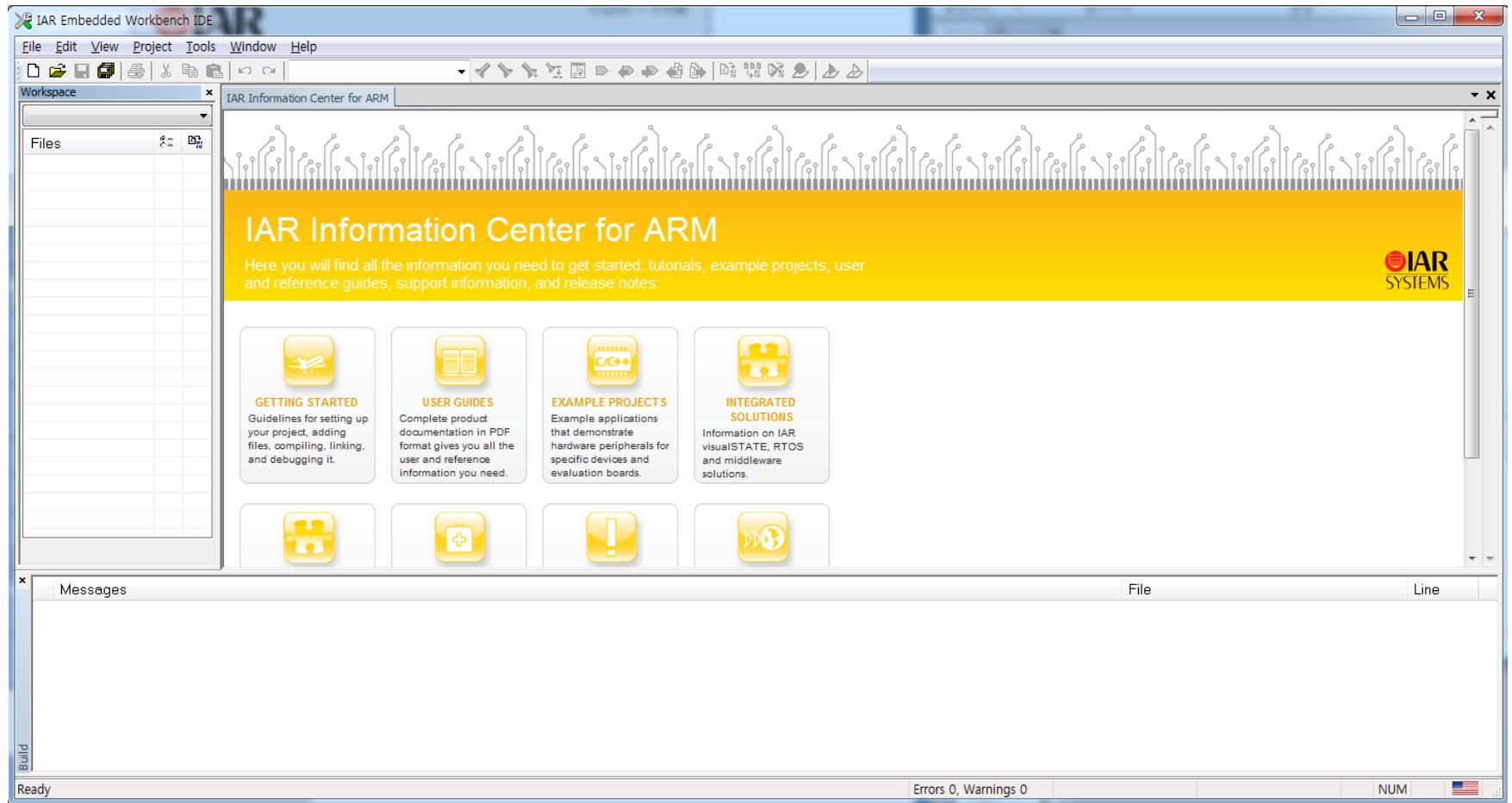
개발환경 설치

- 이제 라이선스가 활성화 되었고, "Done" 클릭



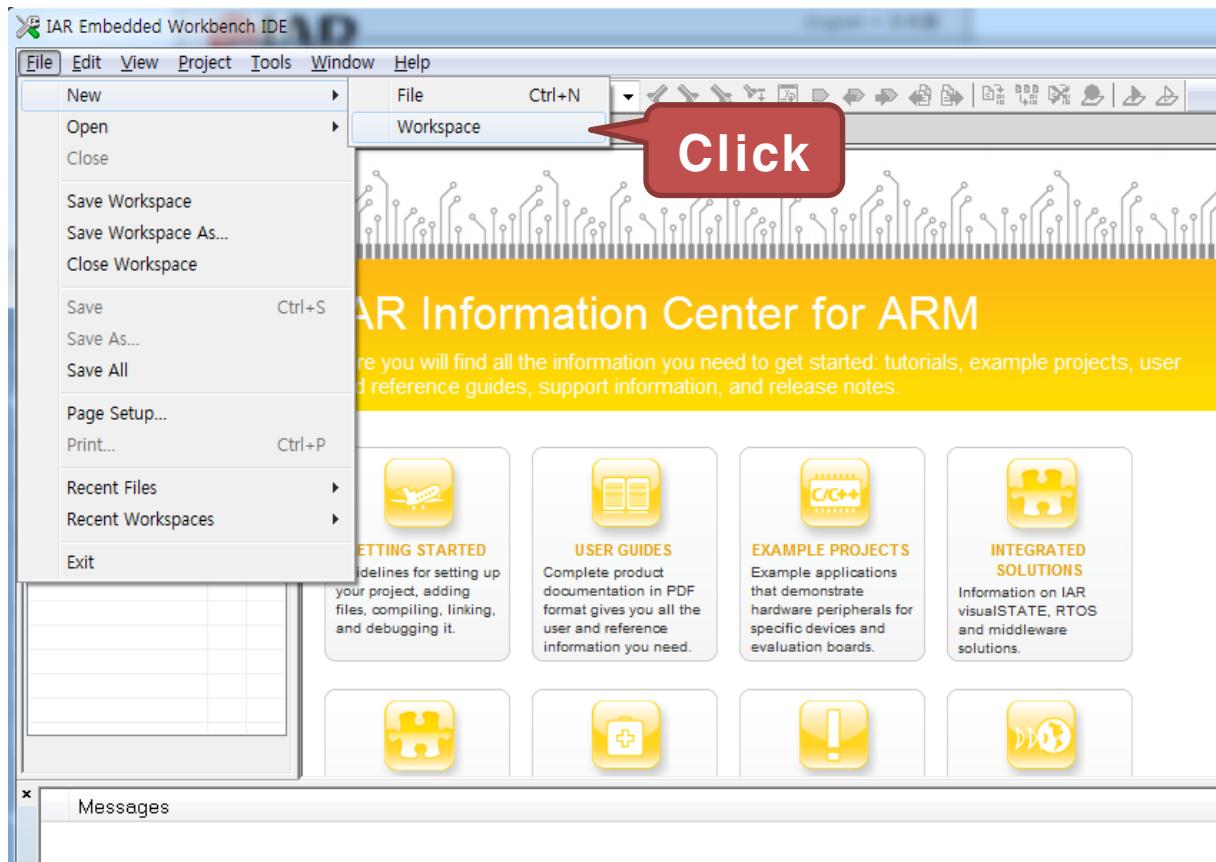
개발환경 설치

- 다음과 같은 IAR Embedded Workbench IDE 실행 화면 확인

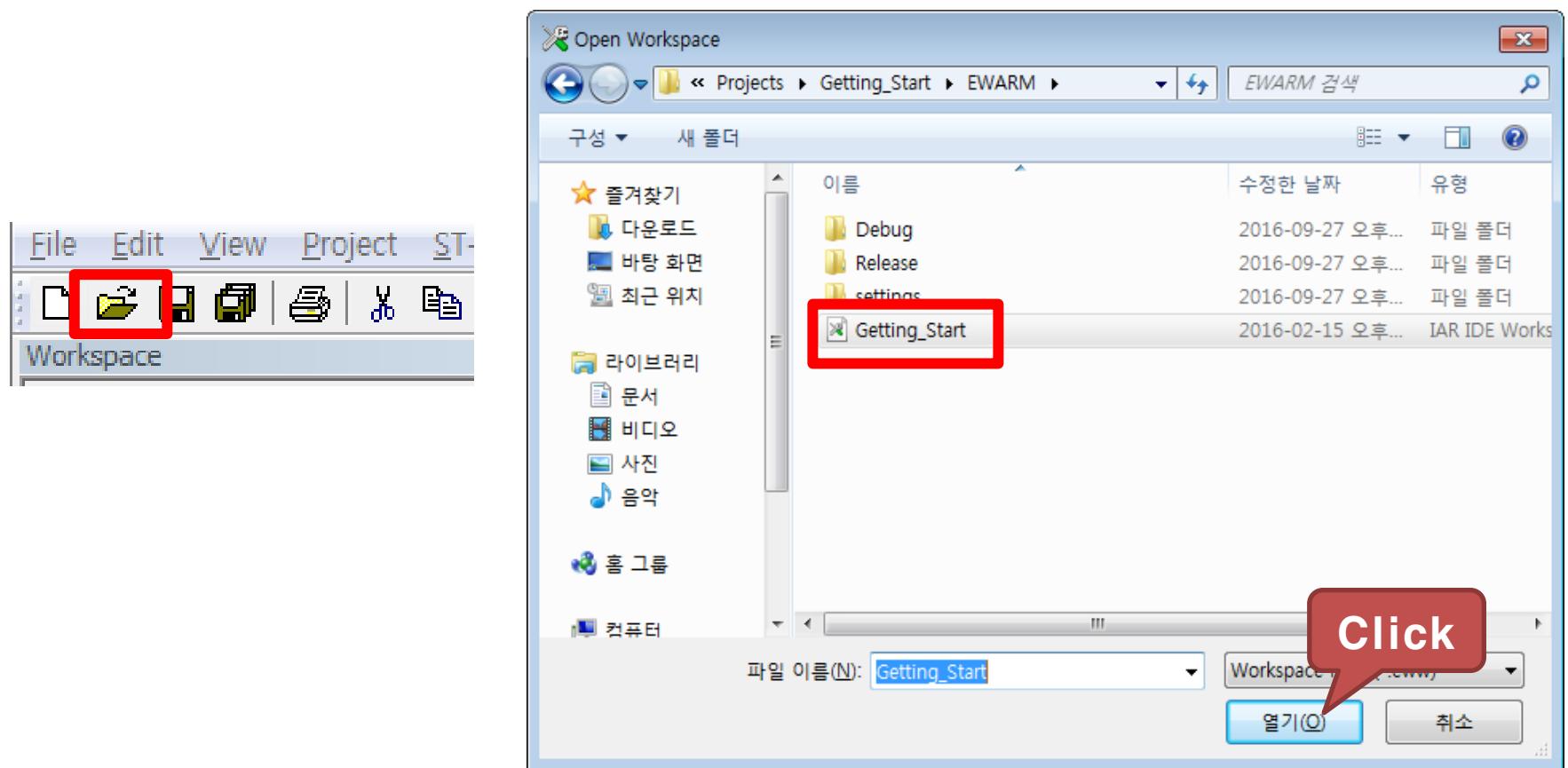


EWARM 실행

- 교재 Example 폴더를 전체 경로에 한글이 없는 폴더로 하드디스크에 복사
- IAR Embedded Workbench 프로그램 실행
- 프로그램 좌측 상단의 “File” 탭에서 Open → “Workspace...”를 선택

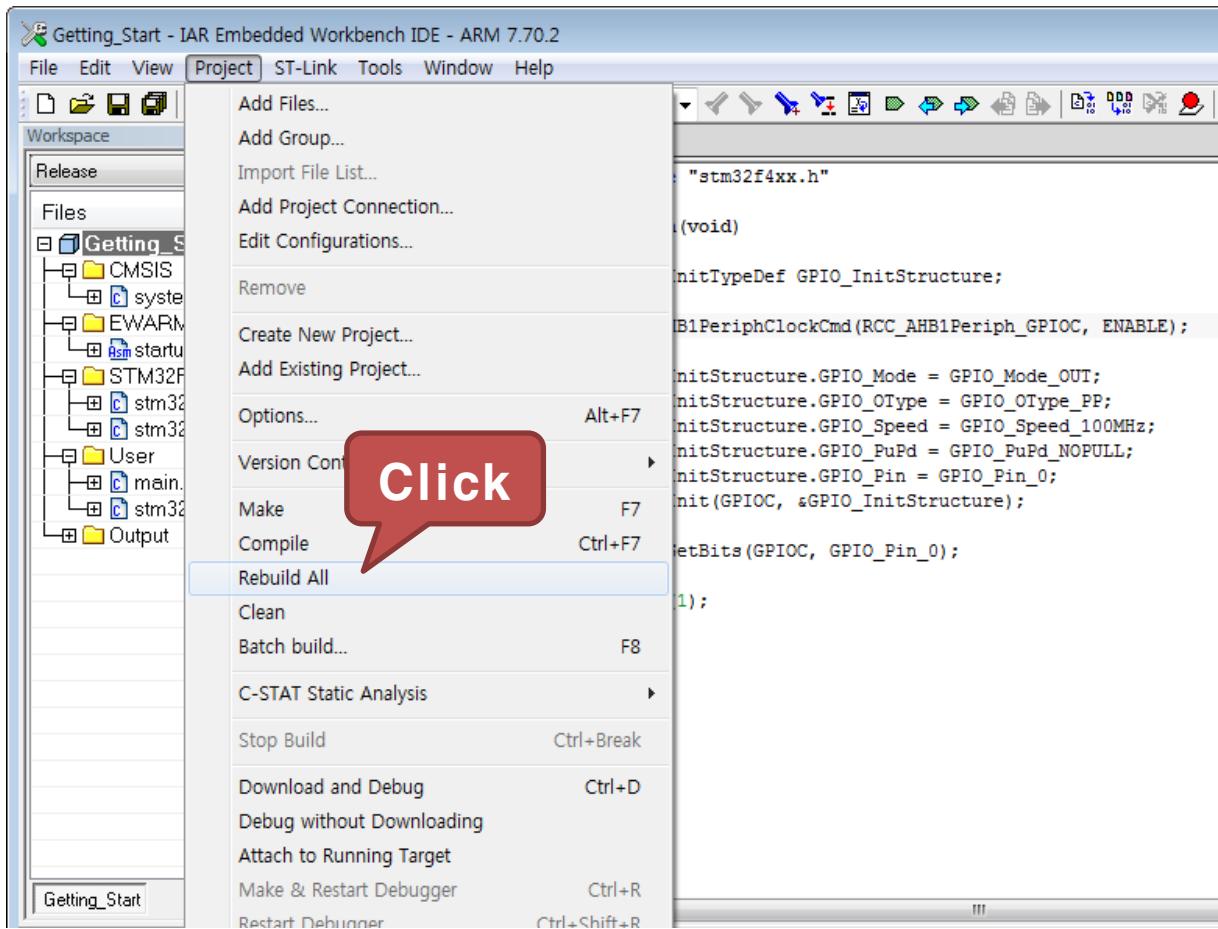


- Workspace 선택창이 나타나면 "Open" 클릭
- 앞에서 복사한 Example 폴더에서 "Projects\Getting_Start\EWARM" 폴더 안에 "Getting_Start.eww"를 선택
 - 파일이 안 보일 시, 파일 이름 옆 목록에서 "Workspace Files(*.eww)"를 선택

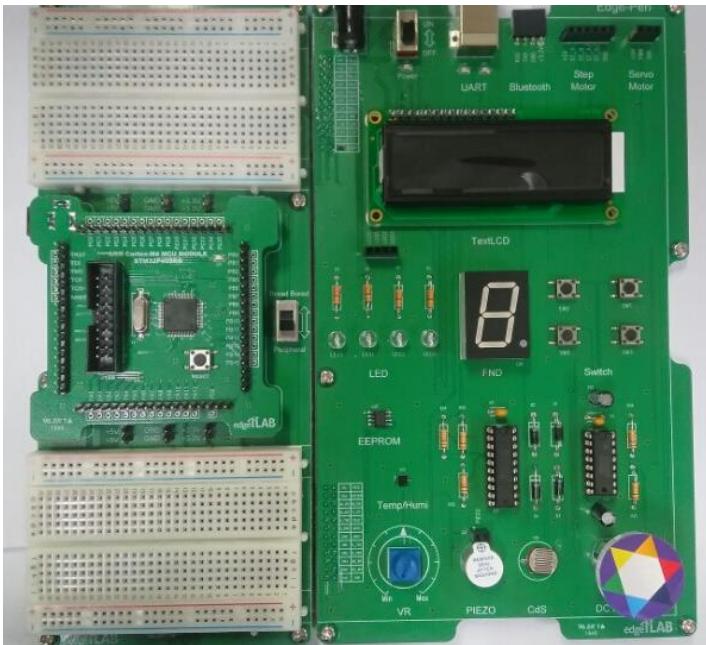


EWARM 실행

- 프로젝트가 열리면 main.c 파일을 편집하는 에디터 창과 Workspace 창이 나타남
- “Project” 탭에서 “Rebuild All”을 선택하여 프로젝트를 빌드



- Edge-MCU 보드, ST-LINK/V2, USB cable, 플랫 cable 준비



- ST-Link의 다운로드 케이블(USB)을 PC와 연결

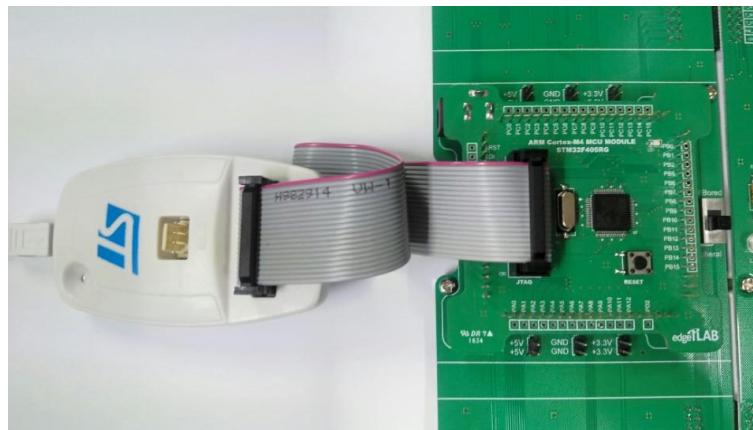


PC 연결



ST-LINK 연결

- JTAG 커넥터에 플랫케이블을 결합하여, ST-Link와 MCU 보드 연결



ST-Link와 MCU 보드 연결

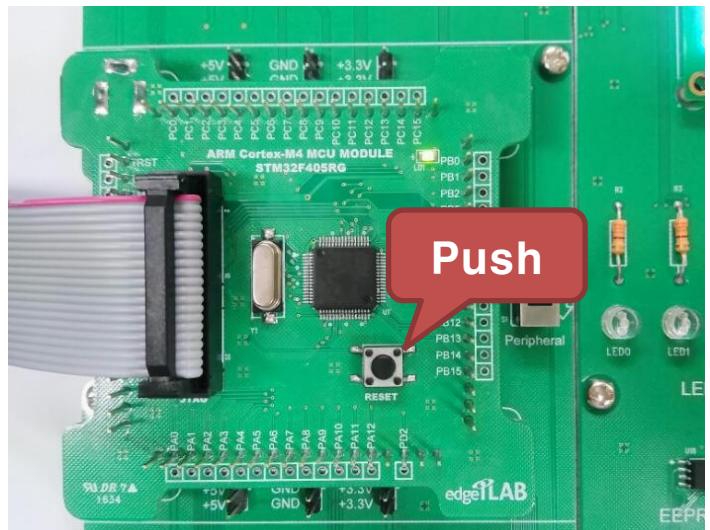
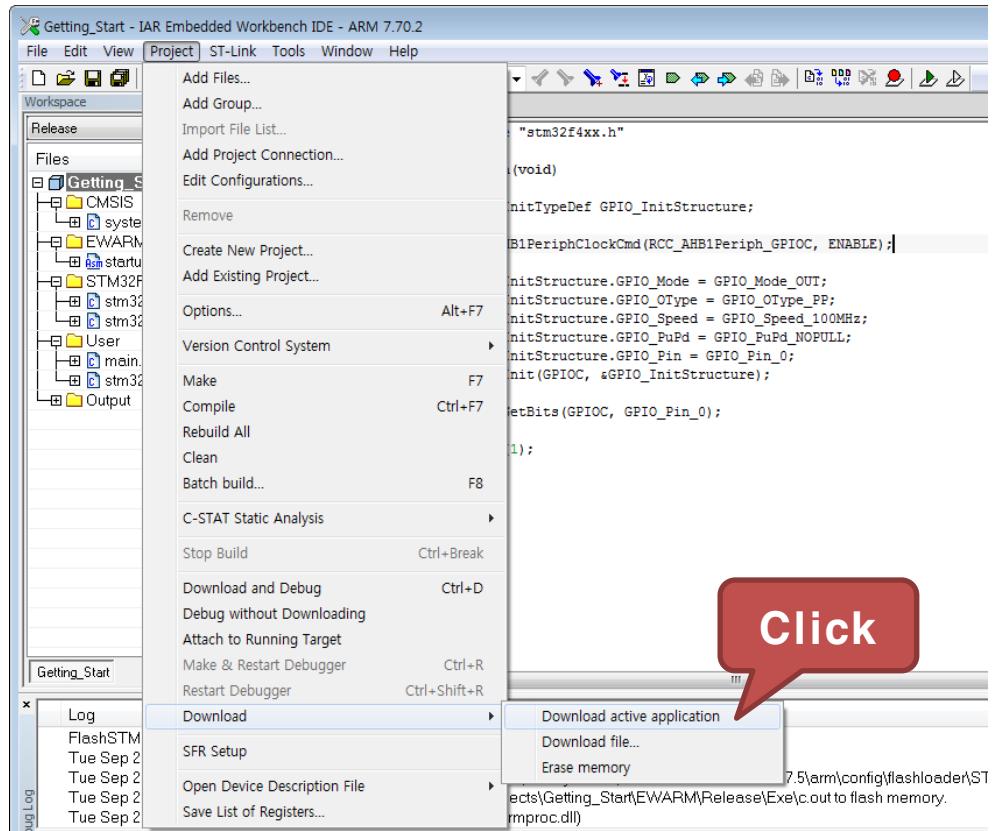
- DC 전원 준비, Edge-Peri 보드에 연결 후, 스위치 ON



DC 전원 연결, 스위치 On

EWARM 실행

- 빌드 한 프로젝트를 다운로드
 - "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
 - 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 누름



EWARM 실행

결과 확인

- 프로젝트 다운로드가 완료되면, LED0에 불이 점등
 - 보드 옆 스위치를 Peripheral(주변장치)로 이동





GPIO 입출력 제어

- 마이크로 컨트롤러와 GPIO
- STM32F405의 입출력 포트
- GPIO를 이용하여 LED 켜기
- GPIO를 이용하여 스위치 입력 받기
- GPIO를 이용하여 FND LED 켜기



엣지아이랩

マイクロ 컨트롤러와 GPIO

- GPIO(General Purpose Input Output)

- **범용으로 사용되는 입출력 포트**

- 설계자가 마음대로 변형하면서 제어할 수 있도록 제공해 주는 입출력 포트
 - 입력과 출력을 선택할 수 있고, 0과 1의 출력 신호를 임의로 만들수 있는 구조
 - 입력으로 사용할 때는 외부 인터럽트를 처리할 수 있도록 하는 경우가 많음
 - 입출력 방향 전환용 레지스터와 출력용/입력용 데이터 레지스터등이 필요
 - 마이크로 컨트롤러에서는 대부분의 핀들을 GPIO로 설정하는 경우가 많음

STM32F405의 입출력 포트

- STM32F405의 입출력 포트 특징

- 최대 51개의 I/O포트를 사용할 수 있도록 구성
- 출력 포트의 버퍼는 많은 유입 전류와 유출 전류를 사용(최대 25mA)
- 모든 포트 핀은 개별적으로 내부 풀업/풀다운/플로팅/오픈드레인 저항을 사용
- 모든 I/O핀은 VCC와 GND사이에 다이오드를 접속하여 포트를 보호
- Read-Modify-Write기능을 가지고 있어, 비트 단위의 포트 설정 자유
- 각 포트에 대한 데이터 출력 레지스터(GPIOx_ODR, GPIOx_BSRR, GPIOx_BRR)와 데이터 입출력 제어 레지스터(GPIOx_CRL, CRH), 데이터 입력 레지스터 (GPIOx_IDR), 그리고 포트의 각 핀의 입출력 값을 고정하는 용도의 레지스터 (GPIOx_LCKR)를 보유

STM32F405의 입출력 포트

- 입출력 포트 제어용 레지스터
 - GPIOx_MODER(GPIO port mode register)
 - GPIO 포트 모드 레지스터
 - MODERy[1:0] : 포트 입출력 설정 비트 ($y = 0 \sim 15$)
입 출력 방향 및 모드를 설정하는 비트
 - 00 : Input (reset state)
 - 01 : General purpose output mode
 - 10 : Alternate function mode
 - 11 : Analog mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15 [1:0]		MODER14 [1:0]		MODER13 [1:0]		MODER12 [1:0]		MODER11 [1:0]		MODER10 [1:0]		MODER9 [1:0]		MODER8 [1:0]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7 [1:0]		MODER6 [1:0]		MODER5 [1:0]		MODER4 [1:0]		MODER3 [1:0]		MODER2 [1:0]		MODER1 [1:0]		MODER0 [1:0]	

STM32F405의 입출력 포트

- 입출력 포트 제어용 레지스터
 - GPIOx_OTYPER(GPIO port output type register)
 - GPIO 포트 출력 타입 레지스터
 - OTy : 포트 출력 타입 설정 비트 ($y = 0 \sim 15$)
 출력 포트의 타입을 설정하는 비트
 - 0 : Output push-pull(reset state)
 - 1 : Output open-drain

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0

STM32F405의 입출력 포트

- 입출력 포트 제어용 레지스터
 - GPIOx_OSPEEDR(GPIO port output speed register)
 - GPIO 출력 포트 속도 레지스터
 - OSPEEDR[y][1:0] : 포트 속도 설정 비트 ($y = 0 \sim 15$)
 - 출력 포트의 속도를 설정하는 비트
 - 00 : Low speed
 - 01 : Medium speed
 - 10 : High speed
 - 11 : Very high speed

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	

STM32F405의 입출력 포트

- 입출력 포트 제어용 레지스터
 - GPIOx_PUPDR (GPIO port pull-up/pull-down register)
 - GPIO 포트 풀업/풀다운 레지스터
 - PUPDRy[1:0] : 포트 풀업/풀다운 설정 비트 ($y = 0 \sim 15$)
출력 포트의 풀업/풀다운을 설정하는 비트
 - 00 : No pull-up, pull-down
 - 01 : Pull-up
 - 10 : Pull-down
 - 11 : Reserved

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15 [1:0]		PUPDR14 [1:0]		PUPDR13 [1:0]		PUPDR12 [1:0]		PUPDR11 [1:0]		PUPDR10 [1:0]		PUPDR9 [1:0]		PUPDR8 [1:0]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7 [1:0]		PUPDR6 [1:0]		PUPDR5 [1:0]		PUPDR4 [1:0]		PUPDR3 [1:0]		PUPDR2 [1:0]		PUPDR1 [1:0]		PUPDR0 [1:0]	

STM32F405의 입출력 포트

- 입출력 포트 제어용 레지스터
 - GPIOx_IDR (GPIO port input data register)
 - GPIO 포트 입력 데이터 레지스터
 - IDRy[1:0] : 포트 입력 데이터 비트 ($y = 0 \sim 15$)
읽기 전용 레지스터이며, 포트로부터 입력되는 데이터 저장

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0

STM32F405의 입출력 포트

- 입출력 포트 제어용 레지스터
 - GPIOx_ODR (GPIO port output data register)
 - GPIO 포트 출력 데이터 레지스터
 - ODRy[1:0] : 포트 출력 데이터 비트 ($y = 0 \sim 15$)
읽기와 쓰기 모두 가능한 레지스터이며, 데이터를 해당 포트 전체에 출력하기 위한 레지스터

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR 15	ODR 14	ODR 13	ODR 12	ODR 11	ODR 10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0

STM32F405의 입출력 포트

- 입출력 포트 제어용 레지스터

- GPIOx_BSRR (GPIO port bit set/reset register)

- GPIO 포트 비트 셋/리셋 레지스터
- BRy[1:0] : 포트 리셋 설정 비트 ($y = 0 \sim 15$)

쓰기 전용 레지스터이며, 해당 비트의 출력을 리셋하기 위한 레지스터

- 0 : No action on the corresponding ODRx bit
- 1 : Resets the corresponding ODRx bit

- BSy[1:0] : 포트 셋 설정 비트 ($y = 0 \sim 15$)

쓰기 전용 레지스터이며, 해당 비트의 출력을 셋하기 위한 레지스터

- 0 : No action on the corresponding ODRx bit
- 1 : Sets the corresponding ODRx bit

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0

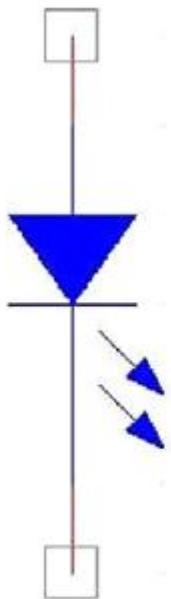
실습 1 : GPIO를 이용하여 LED 켜기

- 실습 개요
 - STM32F405 마이크로컨트롤러의 GPIO를 이용하여 LED를 켜기
 - 입출력 포트를 출력으로 설정하고, 그 포트를 이용하여 LED에 신호를 보내 점등
 - 프로그램이 시작하면 1초마다 LED에 불이 점등
- 실습 목표
 - GPIO 입출력 포트의 방향 제어 및 출력 제어 방법 습득
 - LED 동작 원리 습득
 - 프로그램에서 시간지연 방법 습득

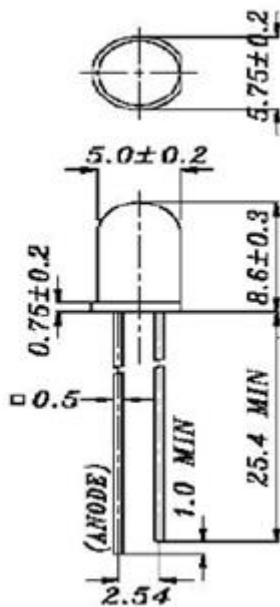
실습 1 : GPIO를 이용하여 LED 켜기

● LED 구조

- LED(Light-emitting diode) : 빛을 발산하는 반도체 소자(발광 다이오드)
- 순방향에 전류를 흘리는 것에 따라 전자와 정공이 재결합하여 발광
- 다리가 긴 부분이 양극 (Anode), 짧은 부분이 음극 (Cathode)



심볼

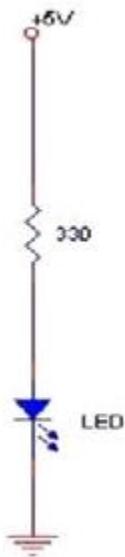


패키지 형상

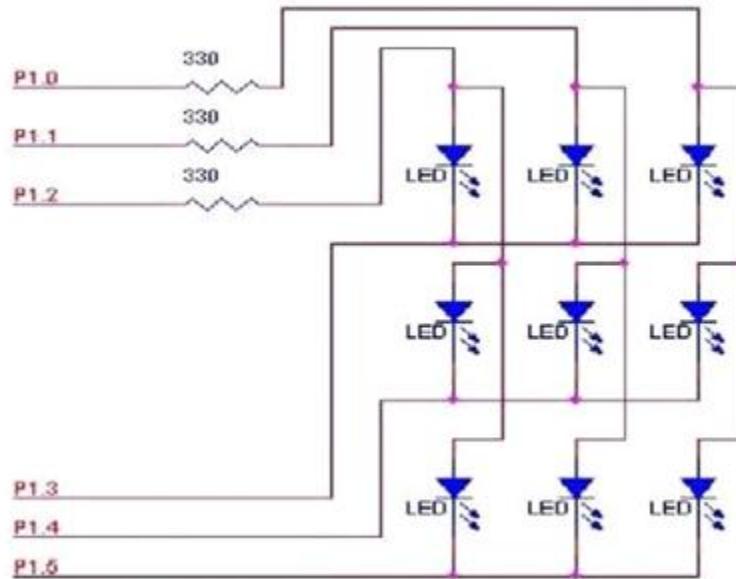
실습 1 : GPIO를 이용하여 LED 켜기

- LED 구동방법

- 정적 구동방식 : 각각의 LED를 독립해서 구동
- 동적 구동방식 : 여러 개의 LED를 매트릭스 구조로 엮어서 함께 구동



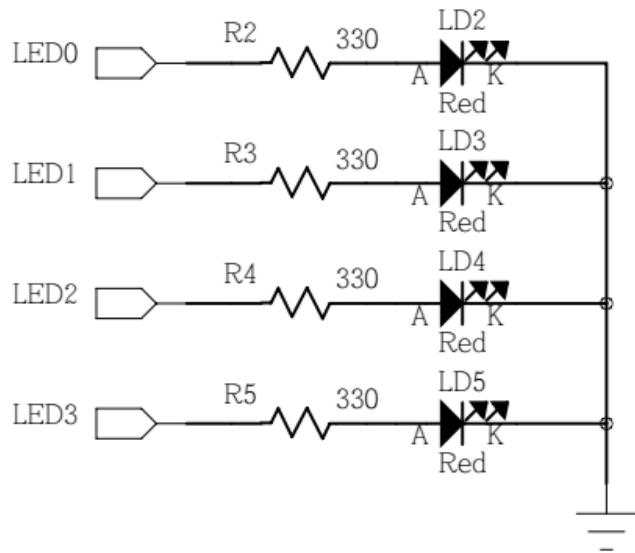
정적 구동



동적 구동

실습 1 : GPIO를 이용하여 LED 켜기

- 사용 모듈의 회로
 - LED 모듈



실습 1 : GPIO를 이용하여 LED 켜기

- 구동 프로그램 : 사전 지식
 - LED를 점등하기 위해서는 LED 신호에 '1'을 인가해야 함
 - 즉, MCU C포트에서 '1'을 출력하도록 해야 함
 - MCU C 포트에 '1'을 출력하기 위해선
 - 입출력 포트 C의 GPIO 방향을 출력으로 설정
 - 입출력 포트를 출력으로 선언하려면 GPIOx_MODER, GPIOx_OTYPER 레지스터를 설정해줘야 함
 - GPIOx_ODR레지스터(여기서는 GPIOC_ODR레지스터)에 '1'로 설정

실습 1 : GPIO를 이용하여 LED 켜기

- GPIO 라이브러리

- STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
- GPIO_InitTypeDef 구조체

GPIO_InitTypeDef구조체	설 명
uint32_t GPIO_Pin	사용할 핀을 선택 GPIO_Pin_x (x = 0 ~ 15 ,All)
GPIOMode_TypeDef GPIO_Mode	해당 핀의 동작 모드를 설정 GPIO_Mode_IN, GPIO_Mode_OUT, GPIO_Mode_AF, GPIO_Mode_AN
GPIOMode_TypeDef GPIO_Speed	해당 핀의 동작 속도를 설정
GPIOMode_TypeDef GPIO_OType	해당 핀이 출력 핀일 경우 타입을 설정 GPIO_OType_PP, GPIO_OType_OD
GPIOMode_TypeDef GPIO_PuPd	해당핀의 풀업/풀다운을 설정 GPIO_PuPd_NOPULL, GPIO_PuPd_UP, GPIO_PuPd_DOWN

실습 1 : GPIO를 이용하여 LED 켜기

- マイクロ 컨트롤러 구동시 시간지연 방법
 - 반복문에 의한 시간 지연
 - for-loop나 while-loop를 사용하여 시간을 지연

```
void delay(unsigned char i){  
    while(i--);  
}  
혹은  
void delay(unsigned char i){  
    int k;  
    for(k=0;k<=i;k++);  
}  
  
void main(void){  
    delay(0x0100);  
}
```

- 매우 부정확한 방법임(MCU상태, 클럭속도에 따라 달라짐)
- 그러나 가장 손쉬운 방법

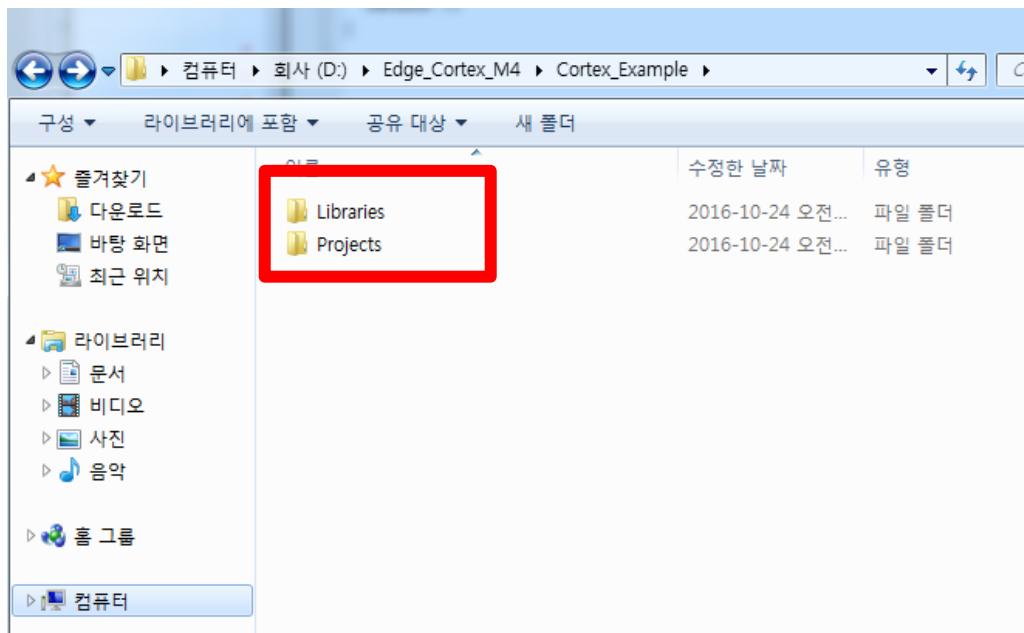
실습 1 : GPIO를 이용하여 LED 켜기

- 마이크로 컨트롤러 구동시 시간지연 방법
 - 하드웨어에 의한 시간 지연
 - 마이크로 컨트롤러에서 하드웨어로 제공하는 내부 타이머/카운터를 사용하는 방법
 - 가장 정확한 방법

실습 1 : GPIO를 이용하여 LED 켜기

● 새 프로젝트 만들기

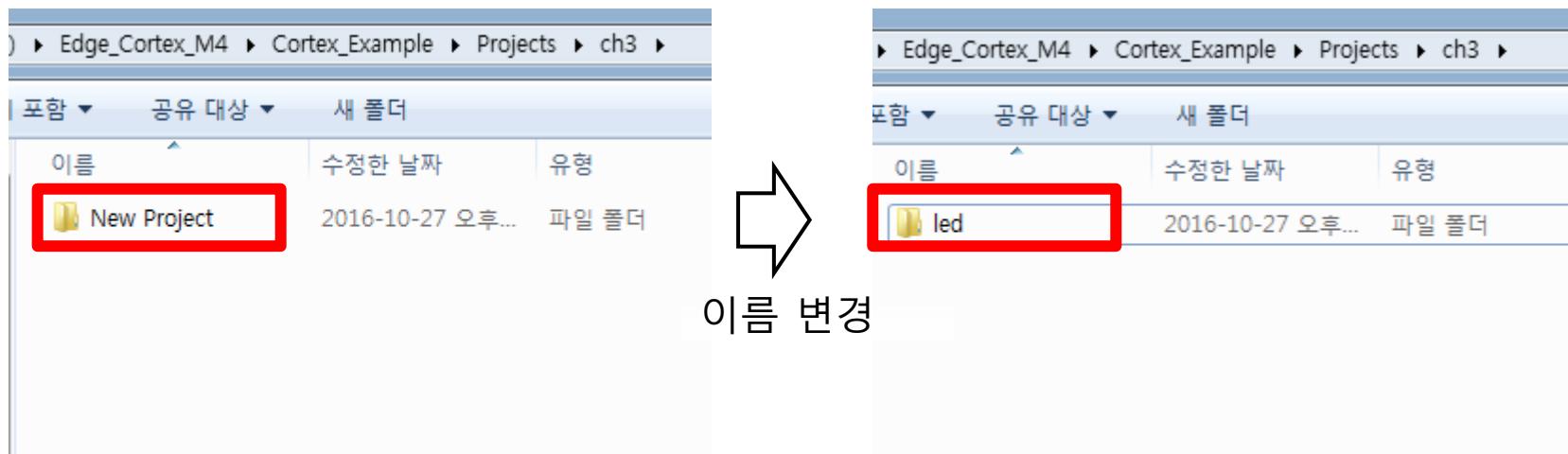
- “Cortex_Example” 폴더를 열기
- “Cortex_Example” 폴더를 살펴보면 “Libraries” 폴더와 “Projects” 폴더 존재
- “Libraries” 폴더는 프로젝트에 필요한 라이브러리 파일들이 들어 있는 폴더이며, “Projects” 폴더는 실제 예제들이 들어있는 폴더



실습 1 : GPIO를 이용하여 LED 켜기

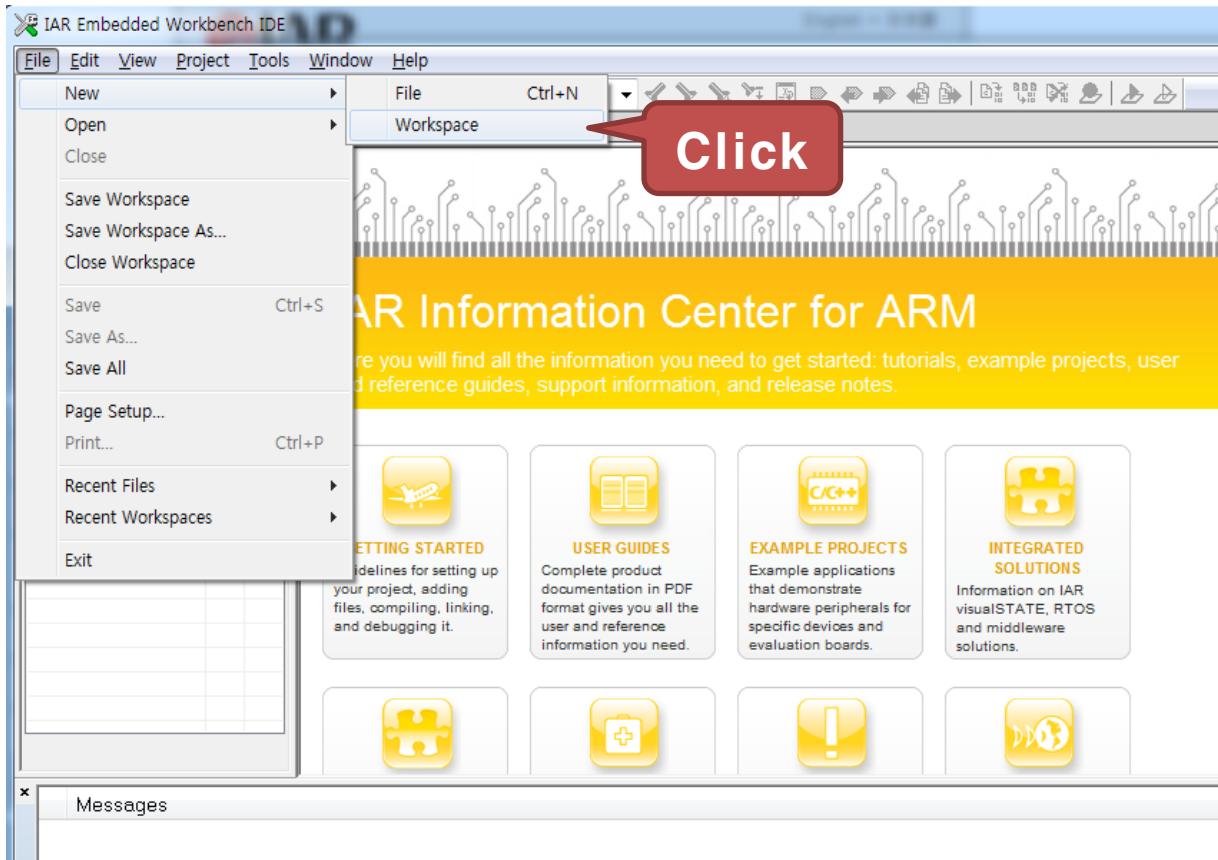
● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch3” 폴더에 붙여 넣기
- 다음과 같이 “New Project” 폴더가 복사된 뒤, 이 폴더의 이름을 변경하면 되는데 여기서는 “led”라고 변경



실습 1 : GPIO를 이용하여 LED 켜기

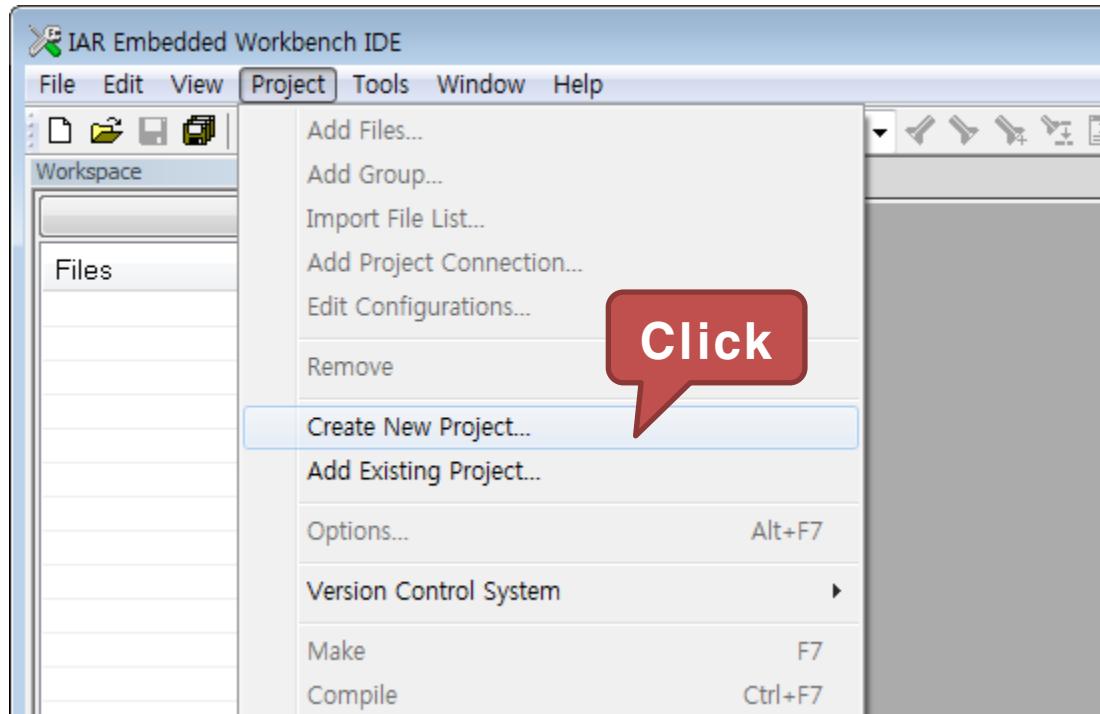
- IAR Embedded Workbench 프로그램 실행
- 프로그램 좌측 상단의 “File” 탭에서 Open → “Workspace...”를 선택



실습 1 : GPIO를 이용하여 LED 켜기

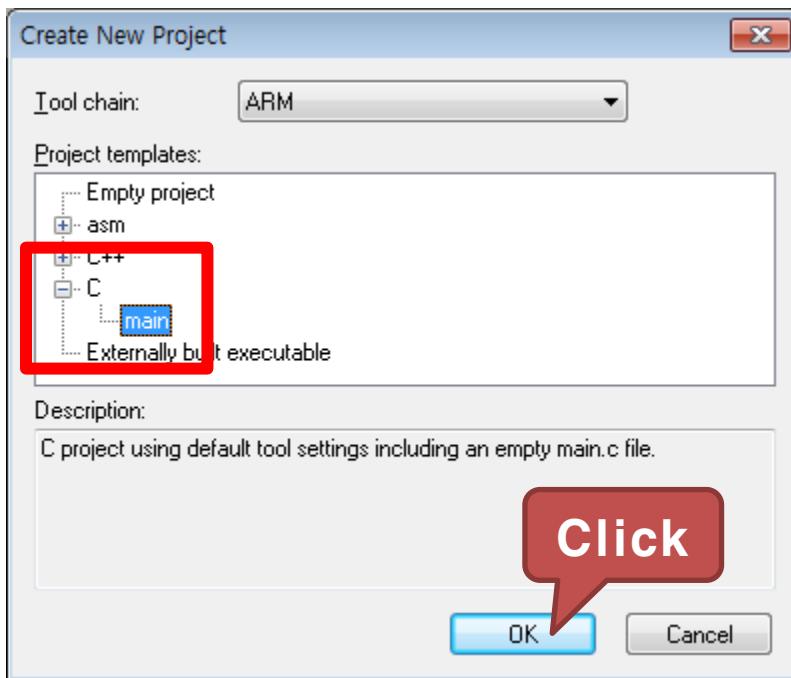
- 새 프로젝트 만들기

- IAR 사의 IAR Embedded Workbench IDE를 실행
- IAR Embedded Workbench IDE 중 Cortx-M4 개발환경은 ARM 계열이기 때문에 줄여서 보통 EWARM 개발환경이라고 부름
- EWARM 실행 화면에서 다음과 같이 새 프로젝트를 생성



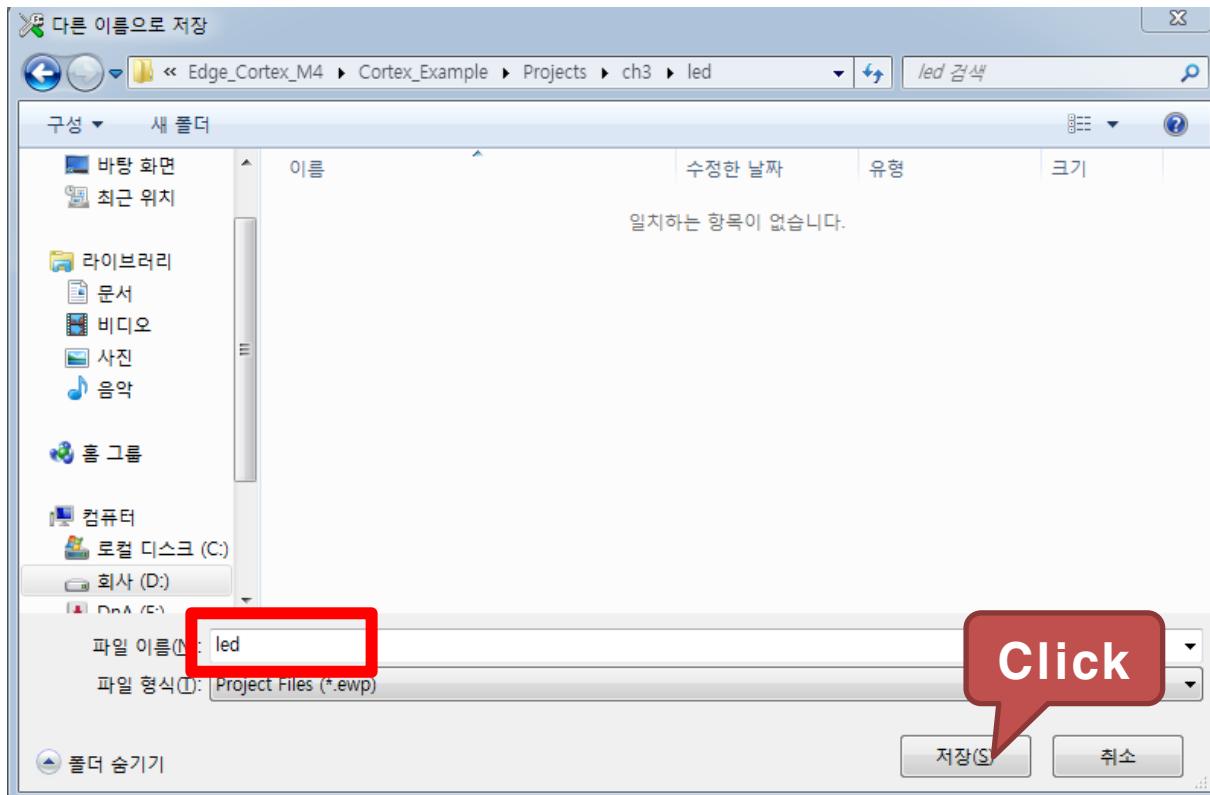
실습 1 : GPIO를 이용하여 LED 켜기

- 새 프로젝트 만들기
 - 새 프로젝트 생성 창에서 다음과 같이 설정하고 “OK”를 클릭



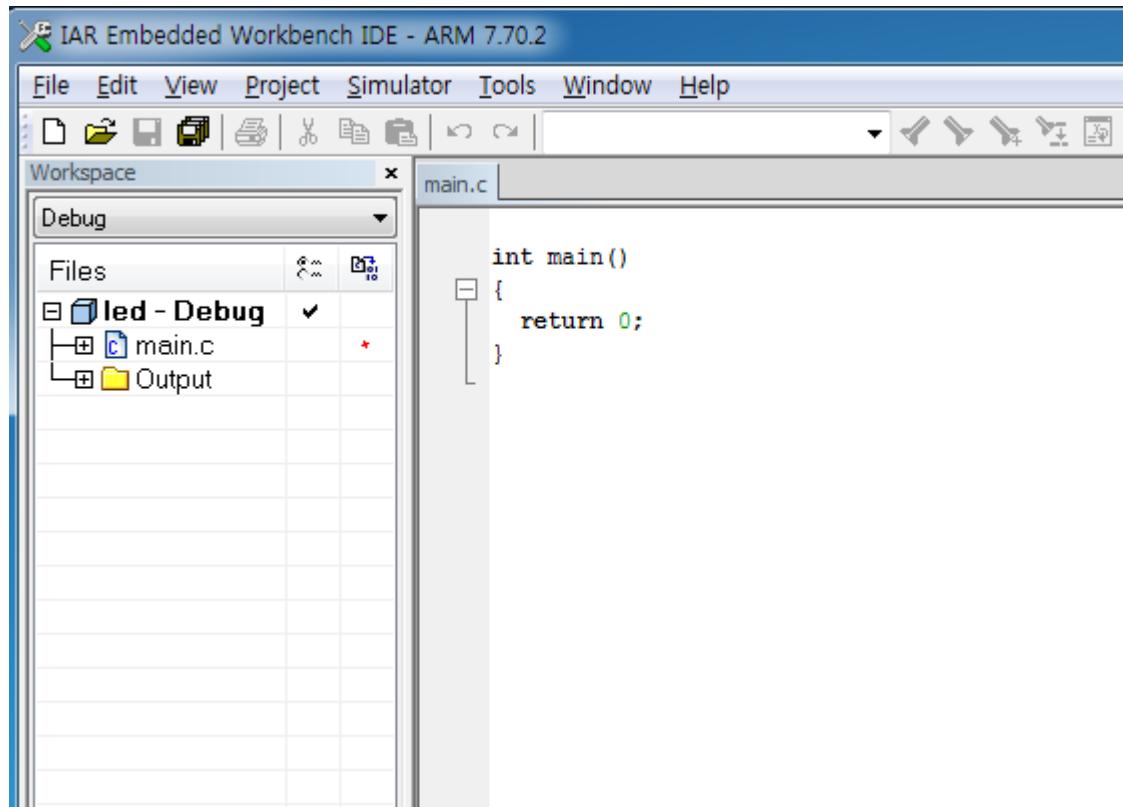
실습 1 : GPIO를 이용하여 LED 켜기

- 새 프로젝트 만들기
 - 새 프로젝트가 위치할 경로는 아까 복사해둔 폴더인 "led" 폴더로 선택
 - 그리고 프로젝트 이름은 "led"라고 쓰고 "저장" 클릭



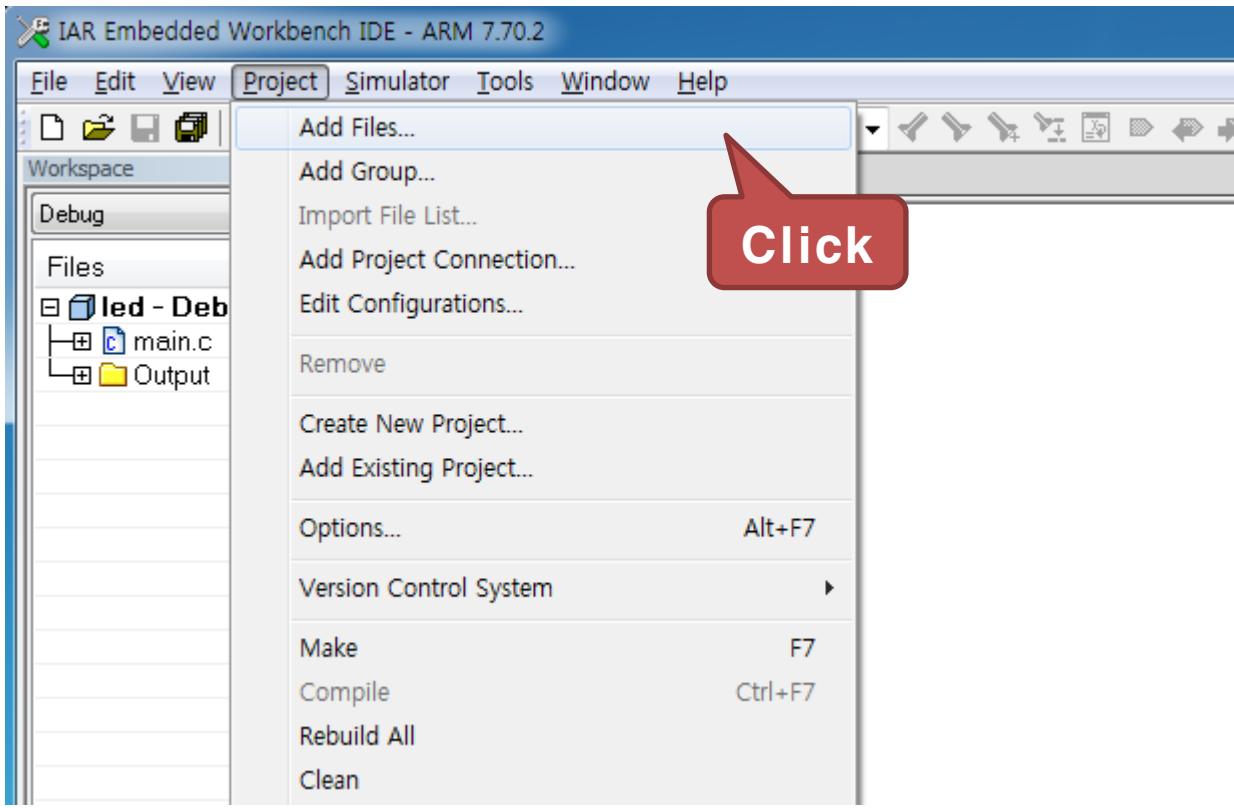
실습 1 : GPIO를 이용하여 LED 켜기

- 새 프로젝트 만들기
 - 다음과 같이 “led” 프로젝트가 생성



실습 1 : GPIO를 이용하여 LED 켜기

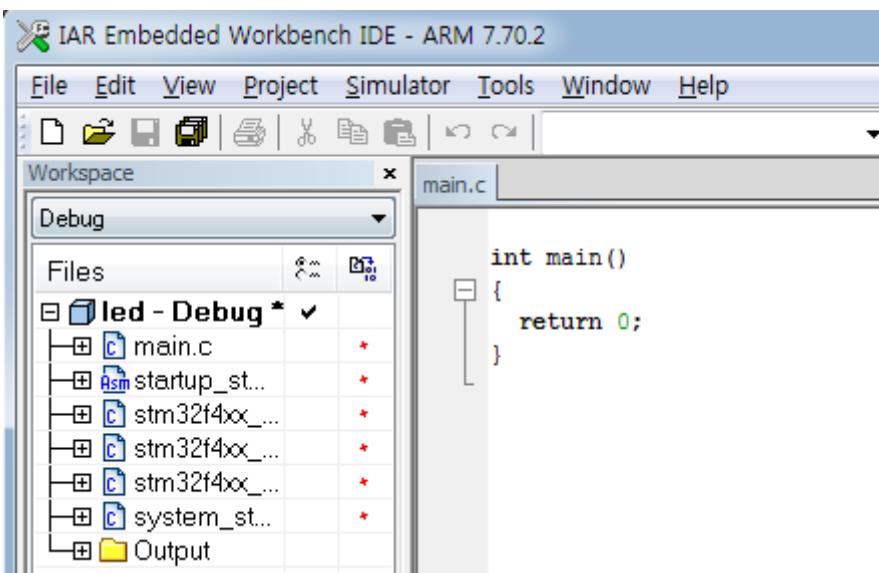
- 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록



실습 1 : GPIO를 이용하여 LED 켜기

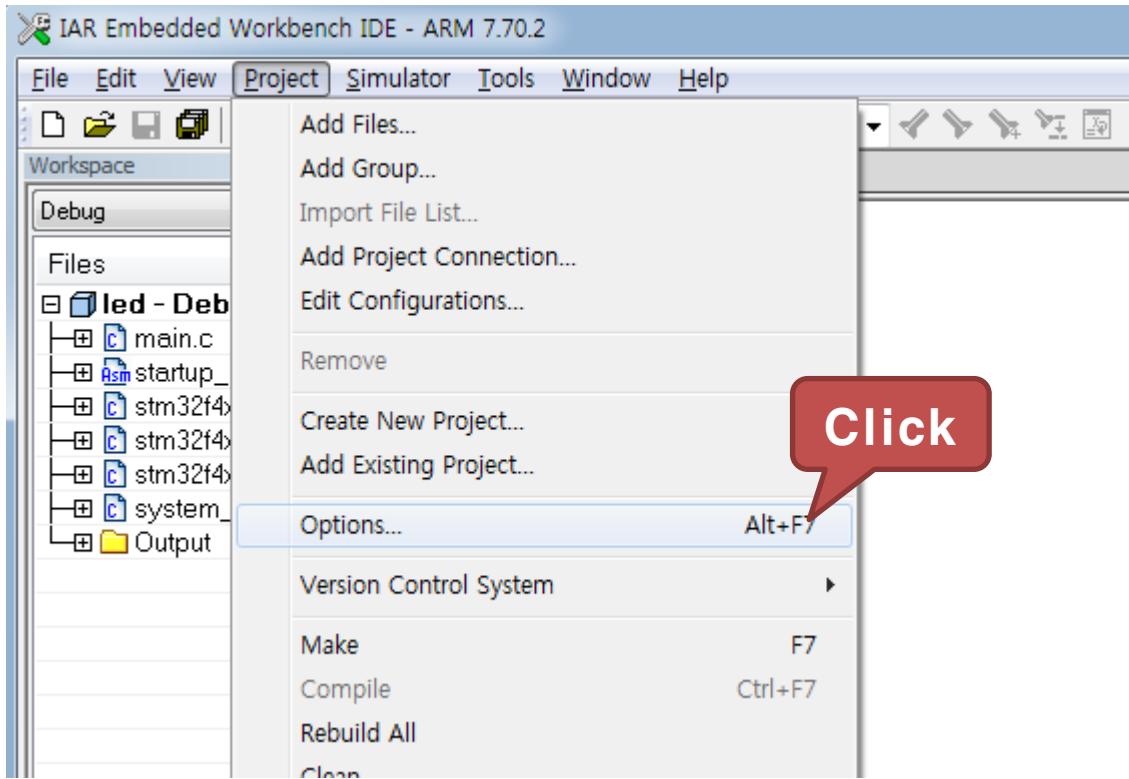
- 라이브러리 파일 추가
 - 추가할 파일

경로	파일
Cortex_Example\Projects\ch3\led	system_stm32f4xx.c
Cortex_Example\Projects\ch3\led	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c



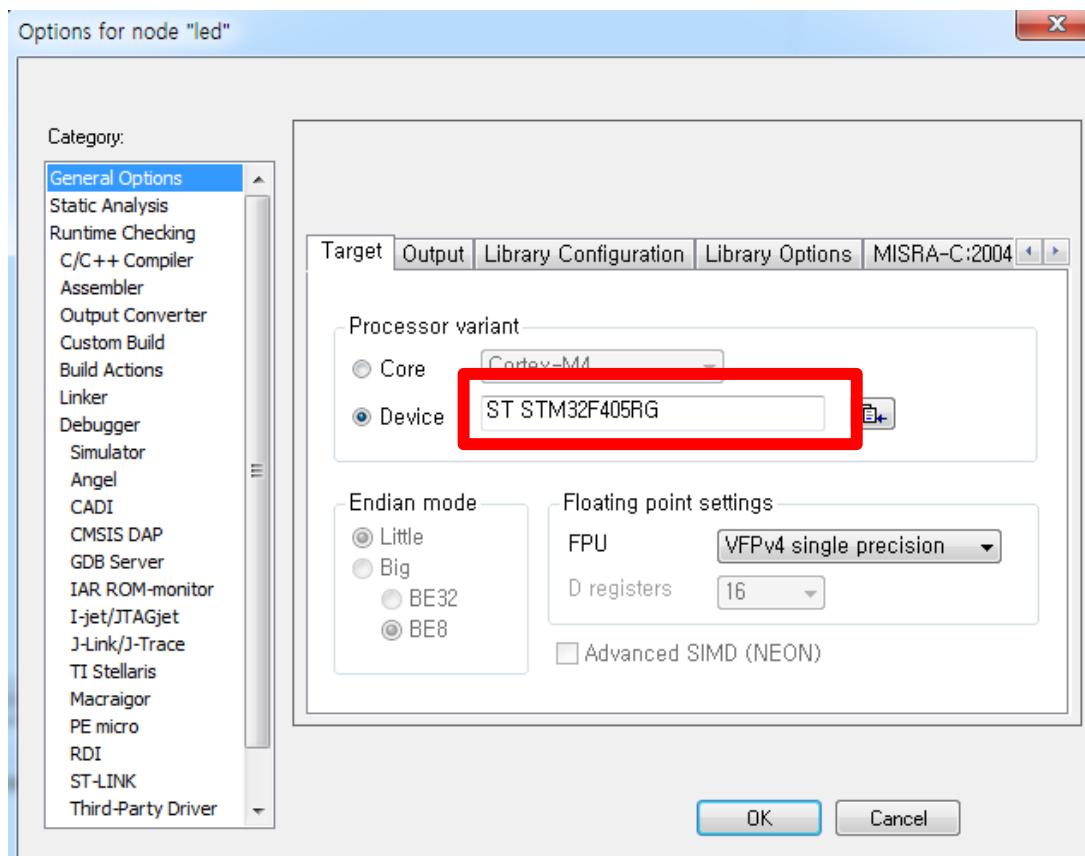
실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - 다음 그림과 같이 프로젝트 설정을 실행
 - 프로젝트 옵션 설정은 복잡하지만 중요하니 정확하게 설정해야 함
 - 다음 그림들을 참고하여 프로젝트 설정



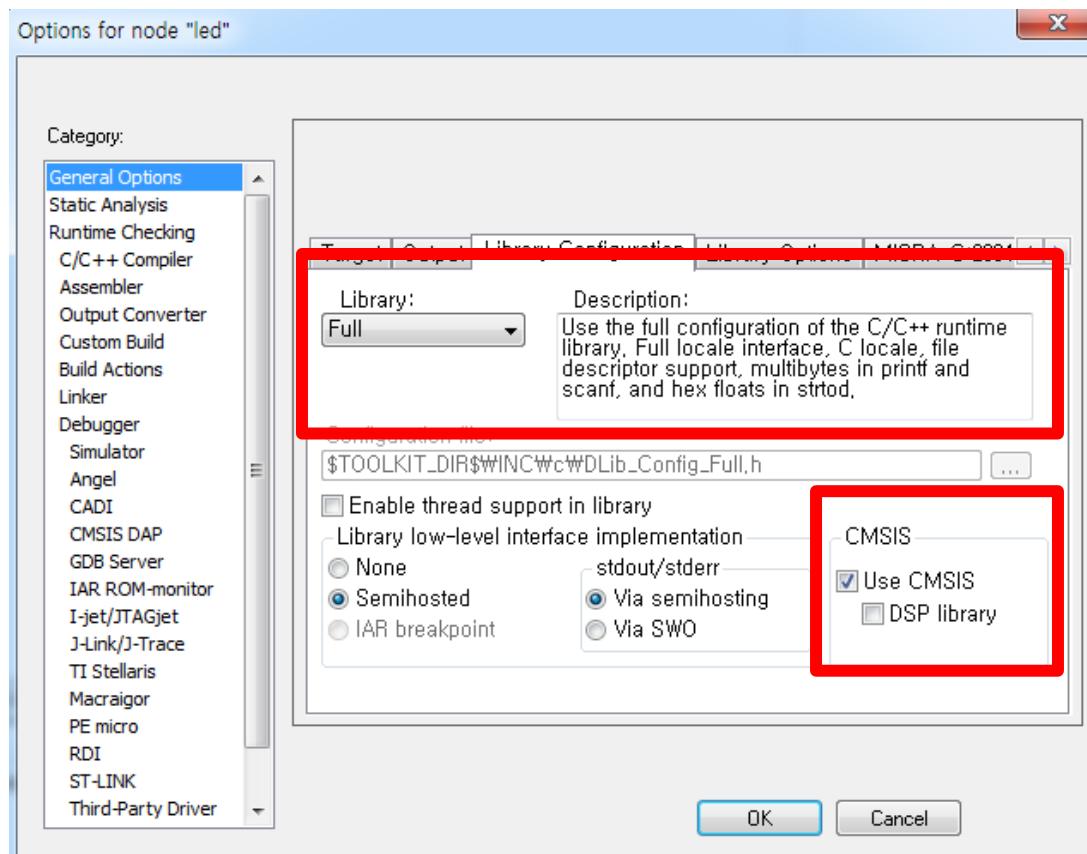
실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - General Options
 - Target Tab 옵션
 - ST STM32F405RG 선택



실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - General Options
 - Library Configuration Tab 옵션
 - full 선택
 - Use CMSIS 체크

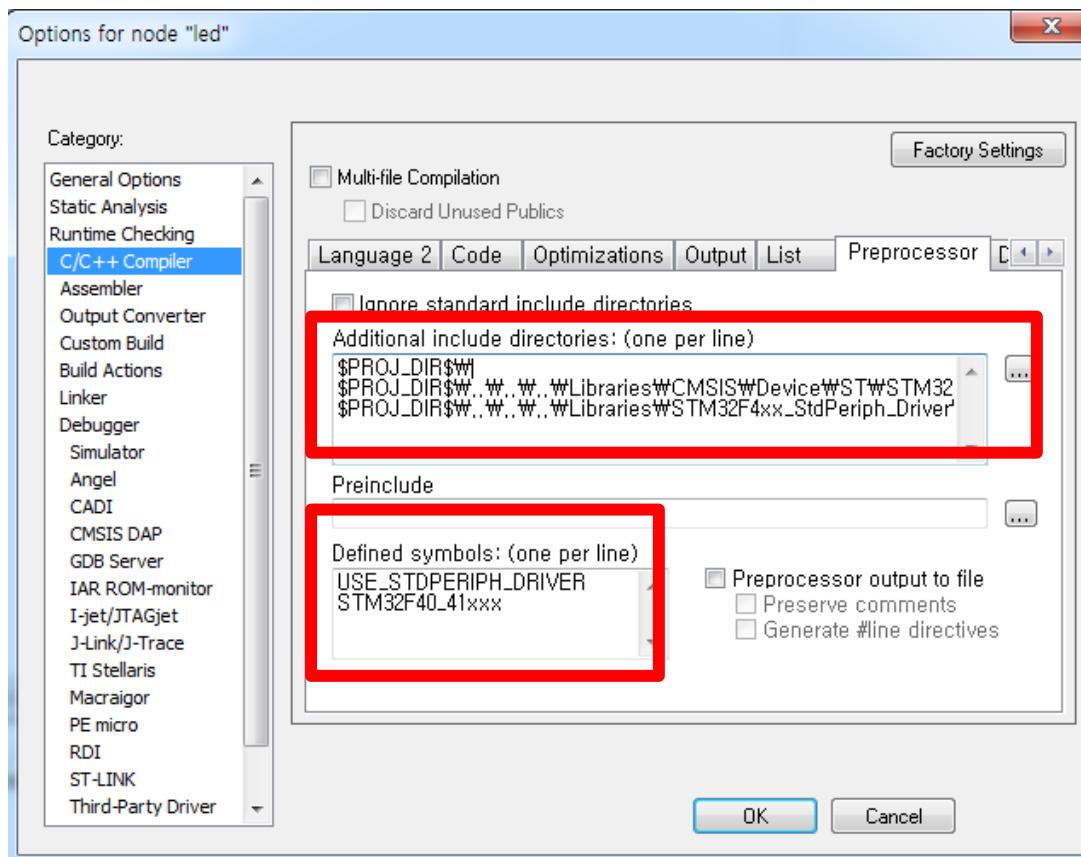


실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - C/C++ Compiler
 - Preprocessor Tab 옵션을 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx

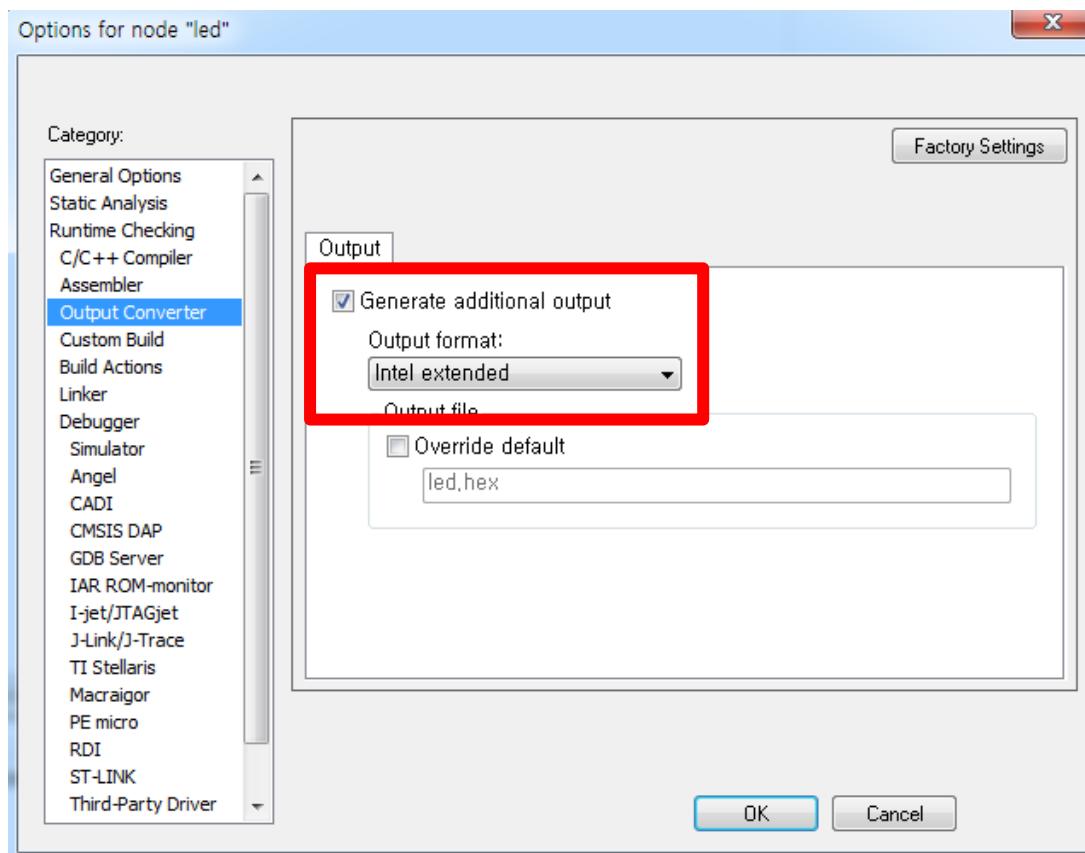
실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - C/C++ Compiler
 - Preprocessor Tab 옵션



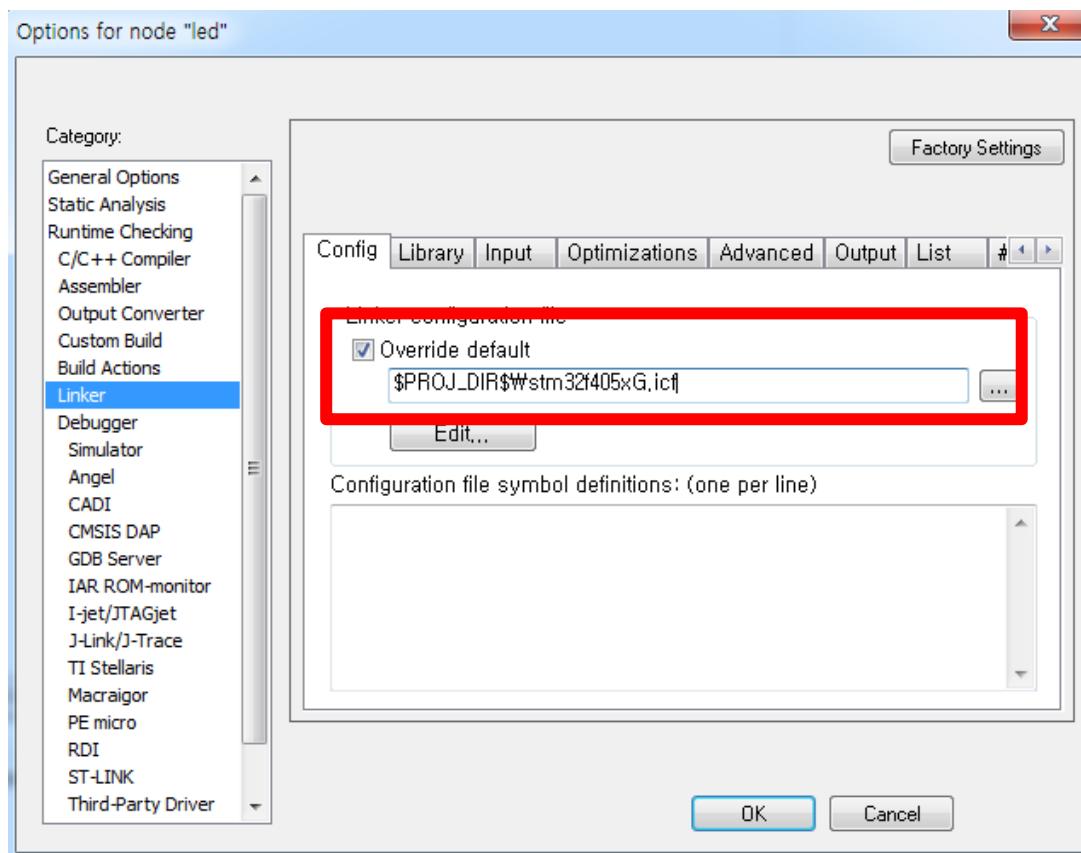
실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - Output Converter
 - Output Tab 옵션
 - Generate additional output 체크
 - Intel extended 선택



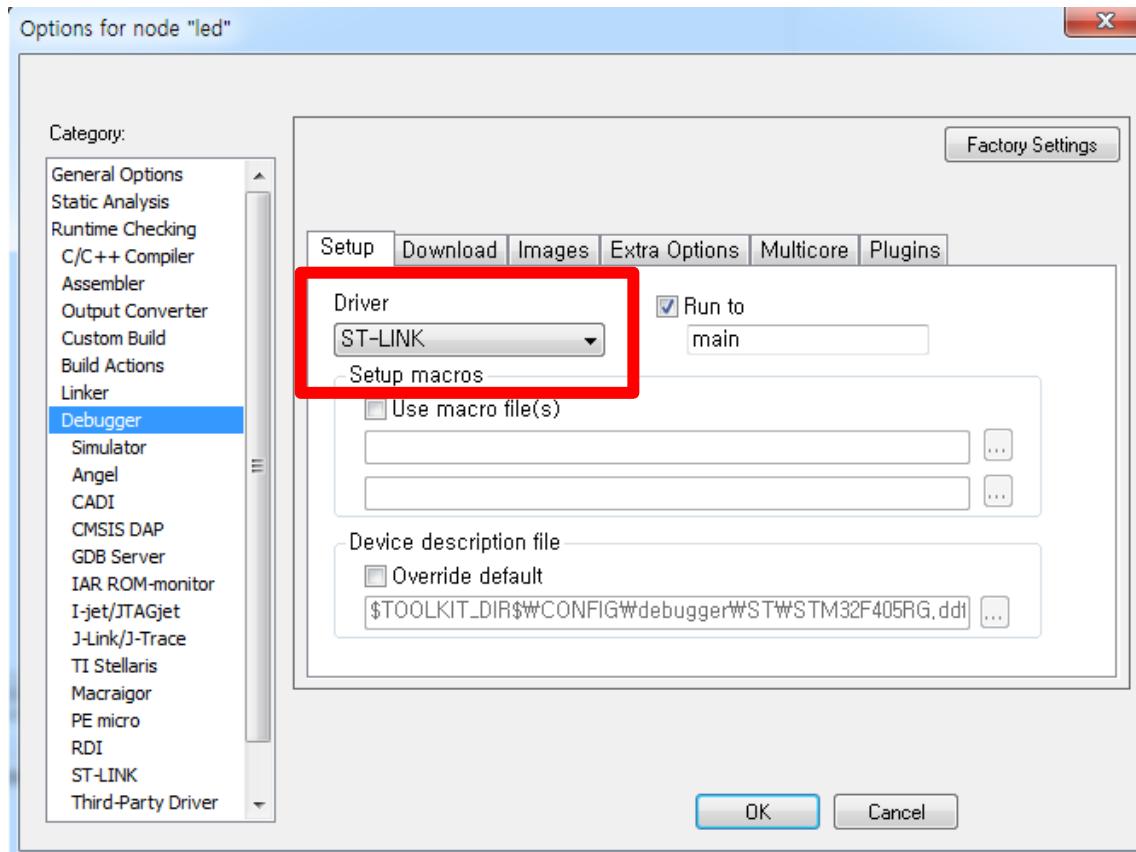
실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - Linker
 - Config Tab 옵션
 - Override default 체크
 - \$PROJ_DIR\$\stm32f405xG.icf 입력



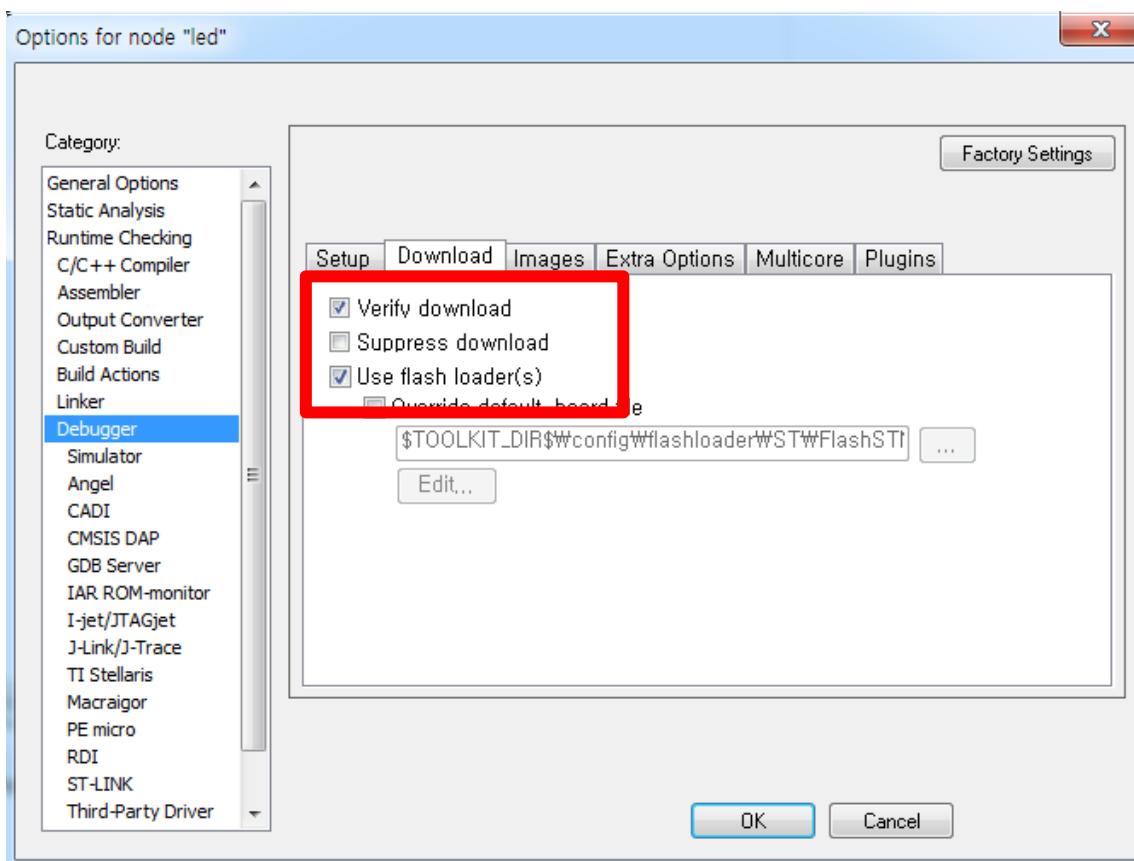
실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - Debugger
 - Setup Tab 옵션
 - ST-LINK 선택



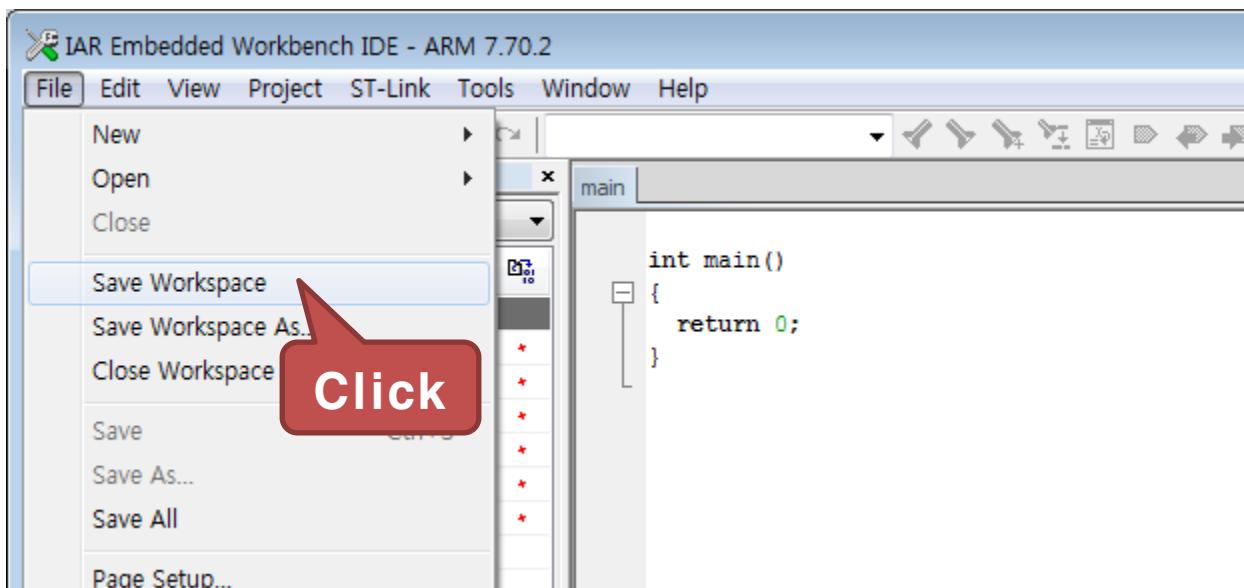
실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 옵션 설정
 - Debugger
 - Download Tab 옵션
 - Verify download, Use flash loader(s) 체크



실습 1 : GPIO를 이용하여 LED 켜기

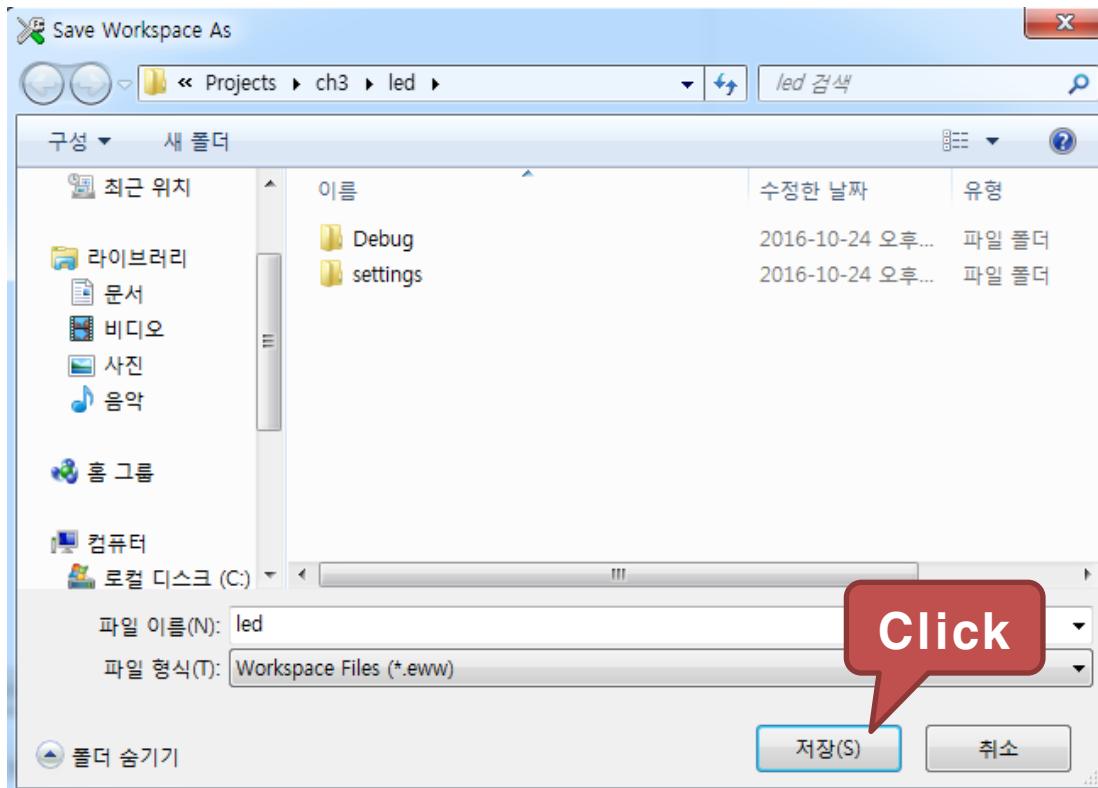
- 프로젝트 저장
 - 본 장비에서 사용되는 모든 예제는 프로젝트 옵션의 설정이 같음
 - 프로젝트 옵션의 설정이 끝나면 이제 프로젝트의 Workspace 를 저장
 - 다음 그림과 같이 “Save Workspace”를 클릭



실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 저장

- “Save Workspace”를 누르면 프로젝트의 Workspace를 저장할 위치가 나타남
- 다음과 같이 저장할 프로젝트의 Workspace 이름을 설정하고 저장



실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 코드
 - main.c 파일에 실제 코드 작성

```
main
// stm32f4xx의 각 레지스터들을 정의한 헤더파일
#include "stm32f4xx.h"

// delay 함수
static void Delay(const uint32_t Count)
{
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);

}

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    unsigned int LED_Data=0x0000;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    while(1)
    {
        GPIO_Write(GPIOC, LED_Data);

        LED_Data++;
        if(LED_Data > 0x0F) LED_Data = 0; //LED_Data값을 1씩 증가한다.
        //LED_Data값이 0x0F보다 크면 0으로 초기화
        Delay(1000);
    }
}
```

실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 코드
 - main.c 코드 작성

```
// stm32f4xx의 각 레지스터들을 정의한 헤더파일
#include "stm32f4xx.h"

// delay 함수
static void Delay(const uint32_t Count) {
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);

}

int main(void){
    GPIO_InitTypeDef GPIO_InitStructure;
    unsigned int LED_Data=0x0000;
    // AHB1(Advanced High-performance Bus 1)버스에 연결되어 있는
    // 주변기기의 동작 클럭을 제어
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
```

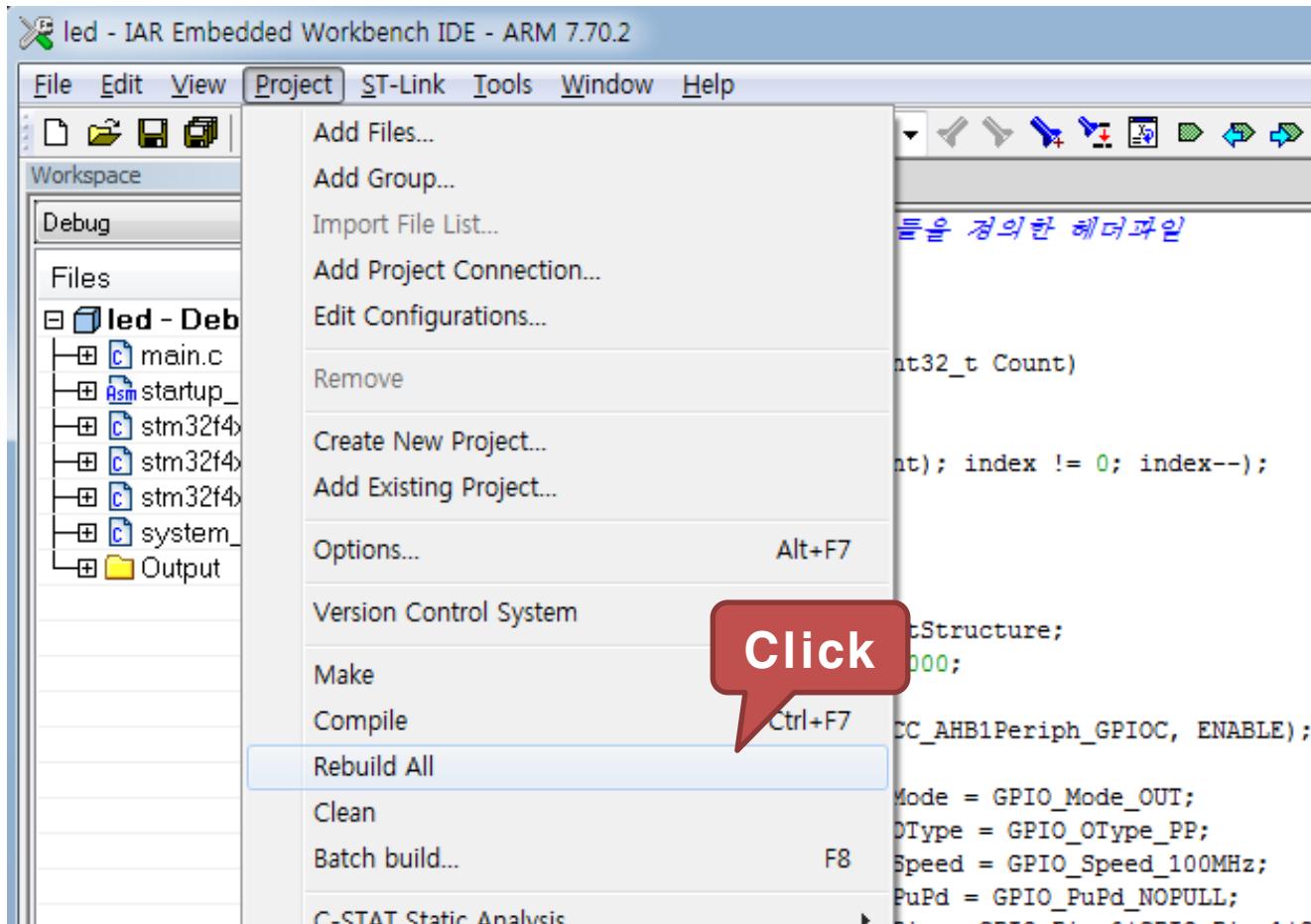
실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 코드
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;  
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  
GPIO_InitStructure.GPIO_Pin =  
    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;  
GPIO_Init(GPIOC, &GPIO_InitStructure);  
while(1) {  
    GPIO_Write(GPIOC, LED_Data);  
    LED_Data++;          // LED_Data값을 1씩 증가  
    // LED_Data값이 0x0F 보다 크면 0으로 초기화  
    if(LED_Data > 0x0F) LED_Data = 0;  
    Delay(1000);  
}  
}
```

실습 1 : GPIO를 이용하여 LED 켜기

- 프로젝트 빌드
 - 코드 작성이 완료되면 led 프로젝트 빌드



실습 1 : GPIO를 이용하여 LED 켜기

- 새 프로젝트 만들기
 - 프로젝트의 설정이 정확하게 되었다면 다음과 같이 빌드 결과가 나타남

The screenshot shows a software interface for building a project. The top section is labeled "Messages" and lists several files: main.c, stm32f4xx_gpio.c, stm32f4xx_it.c, stm32f4xx_rcc.c, and system_stm32f4xx.c. Below this, the process is shown as "Linking". The bottom section shows the output of the build command, which includes "GPIO.out" and "Converting". A red box highlights the message area, which displays "Total number of errors: 0" and "Total number of warnings: 0". At the bottom of the window, there are tabs for "Build" and "Debug Log", with "Build" being the active tab. The status bar at the bottom shows the word "ady".

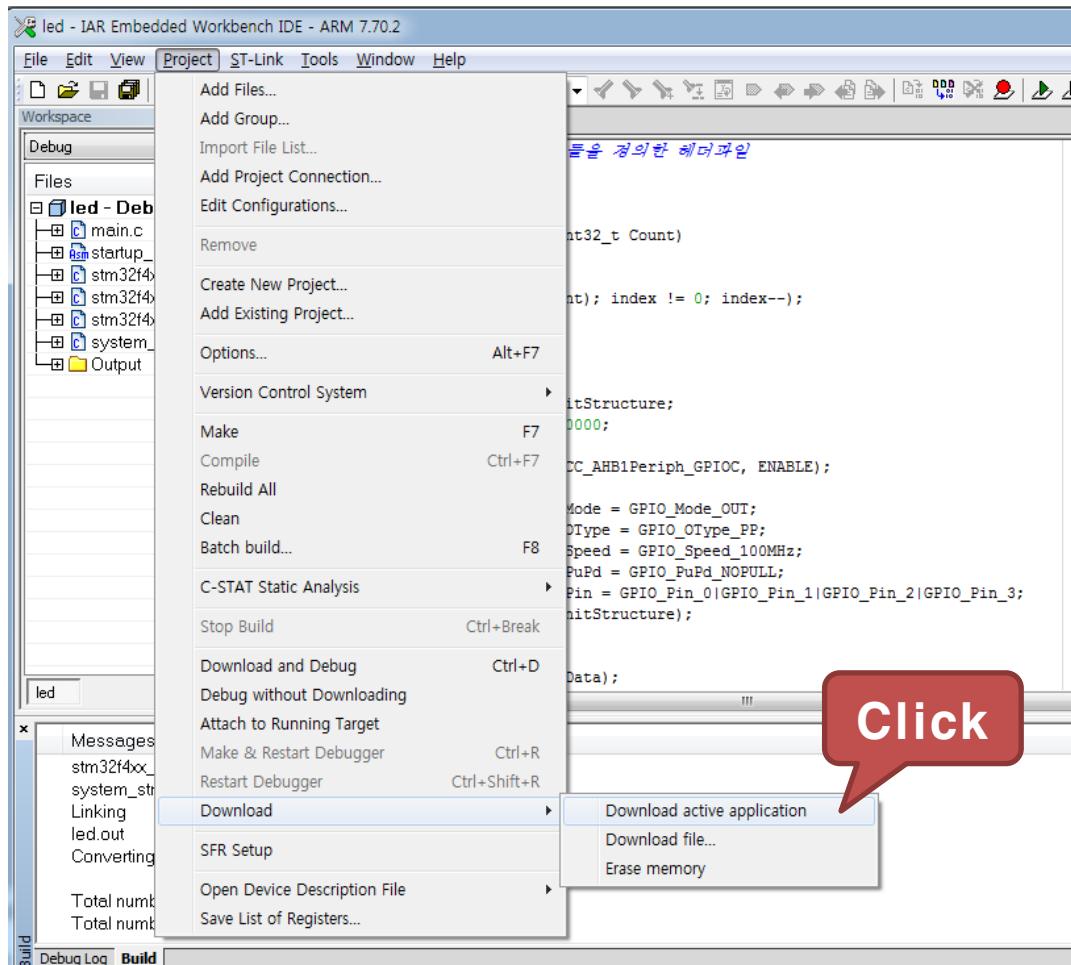
```
Messages
main.c
stm32f4xx_gpio.c
stm32f4xx_it.c
stm32f4xx_rcc.c
system_stm32f4xx.c
Linking
GPIO.out
Converting

Total number of errors: 0
Total number of warnings: 0

Build Debug Log
ady
```

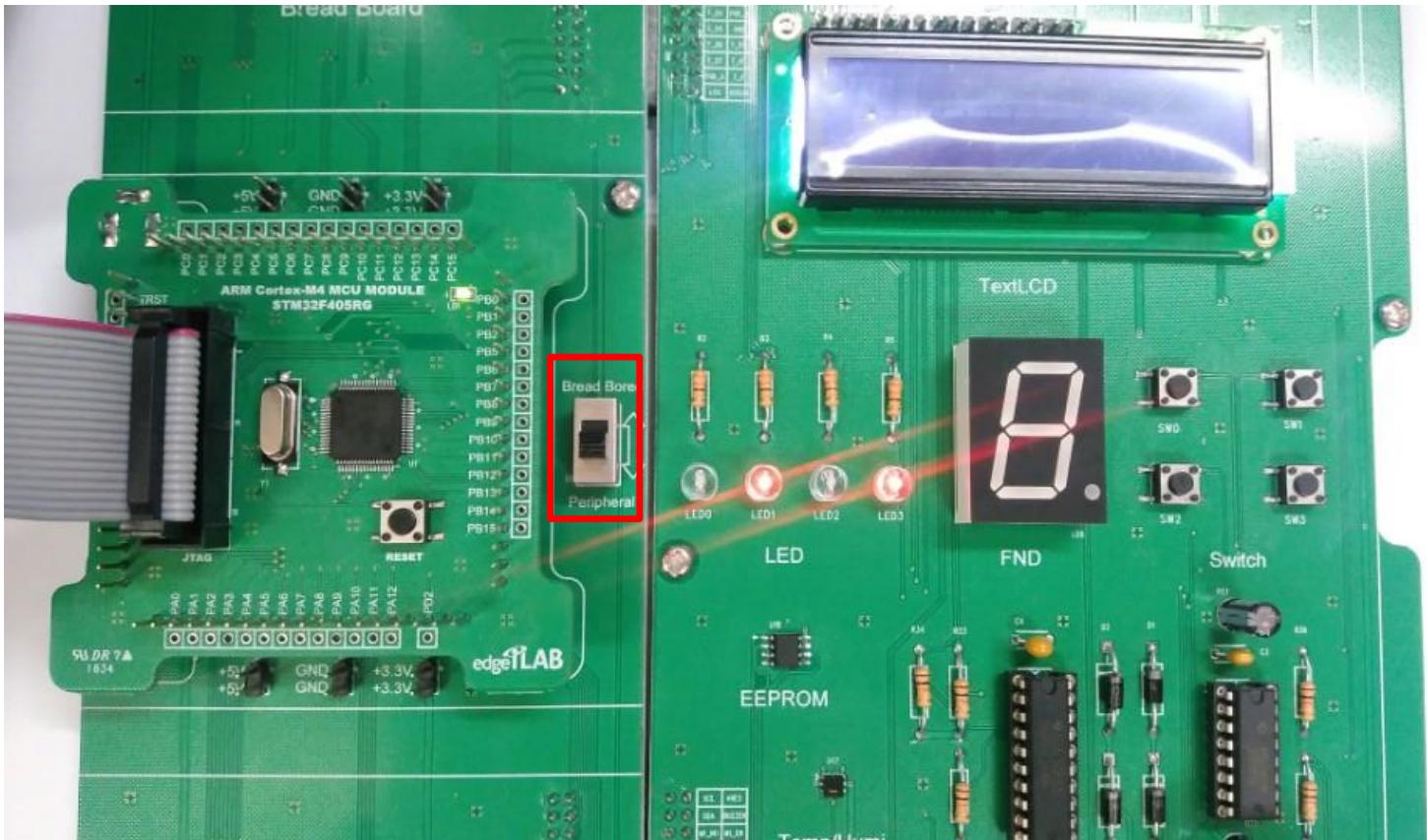
실습 1 : GPIO를 이용하여 LED 켜기

- “Project” 탭에서 “Download”의 “Download active application”을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 “Reset” 버튼 클릭



실습 1 : GPIO를 이용하여 LED 켜기

- 실행 결과
 - LED의 불이 순차적으로 점멸
 - 보드 옆 스위치를 Peripheral(주변장치)로 이동

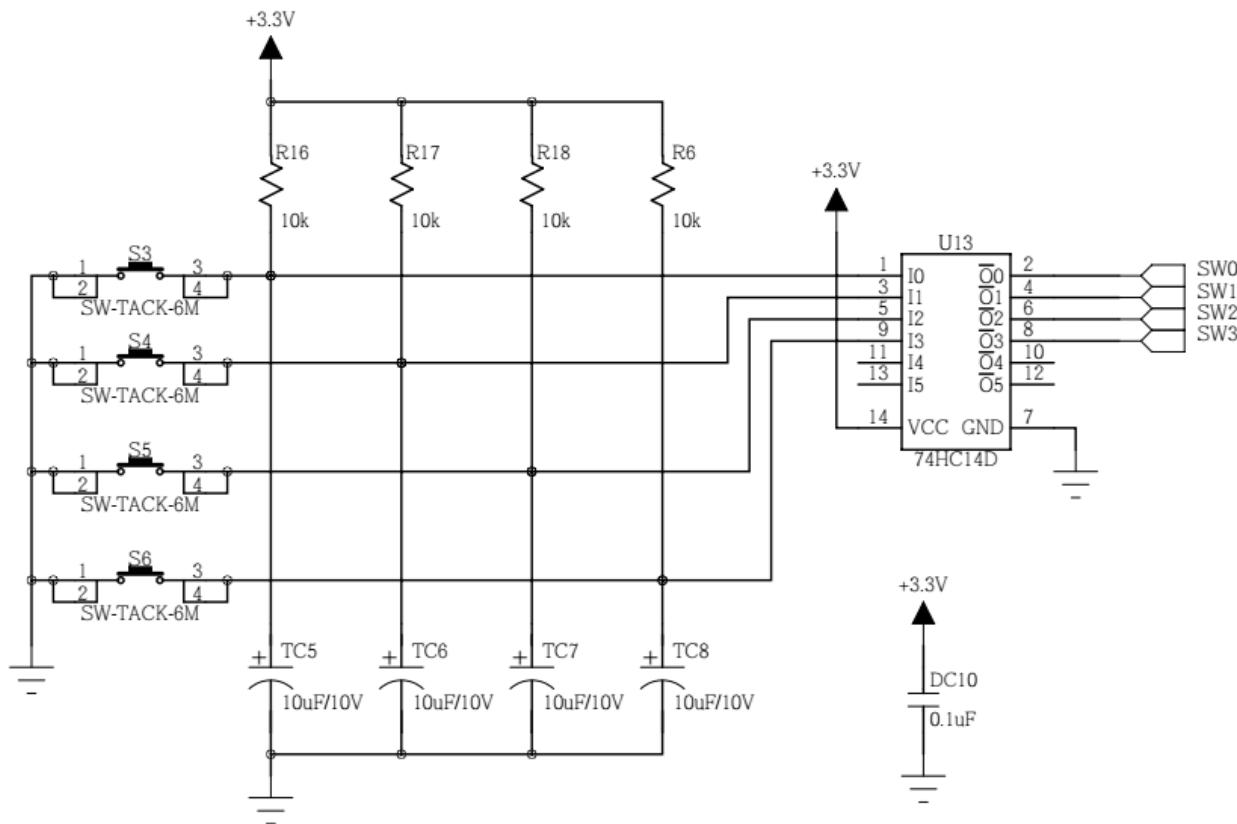


실습 2 : GPIO를 이용하여 스위치 입력 받기

- 실습 개요
 - 단순 출력이 아니고, GPIO포트를 통해 신호를 입력하여 그 신호에 따라 LED켜기
 - 스위치 모듈의 스위치를 누르면 해당되는 LED 모듈의 LED가 점등되도록 함
 - 입출력 포트를 스위치쪽은 입력으로 LED쪽은 출력으로 설정하도록 함
- 실습 목표
 - GPIO 입출력 포트의 방향 제어 및 입력 제어 방법 습득
 - 스위치 동작원리 습득

실습 2 : GPIO를 이용하여 스위치 입력 받기

- 사용 모듈의 회로
— Push Button 모듈



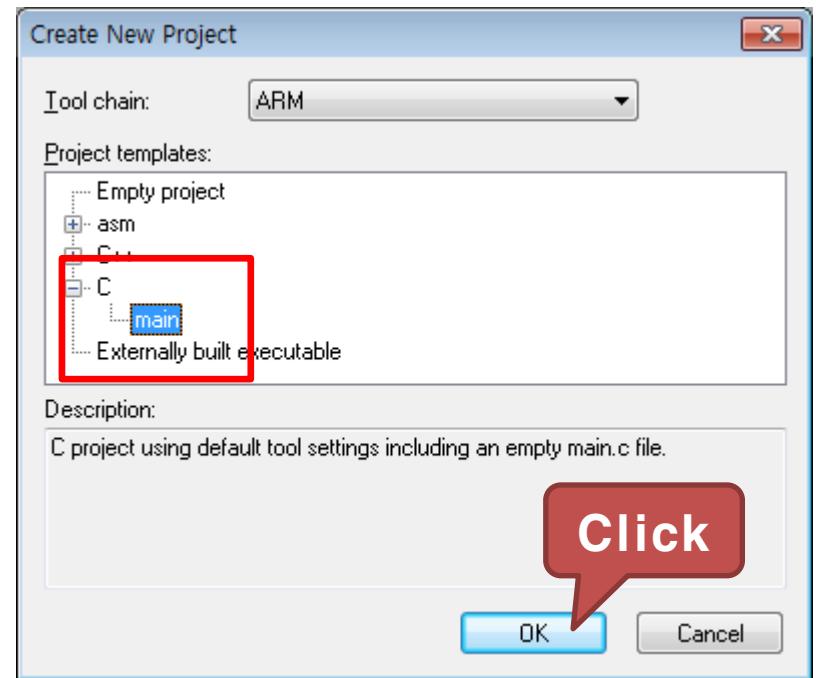
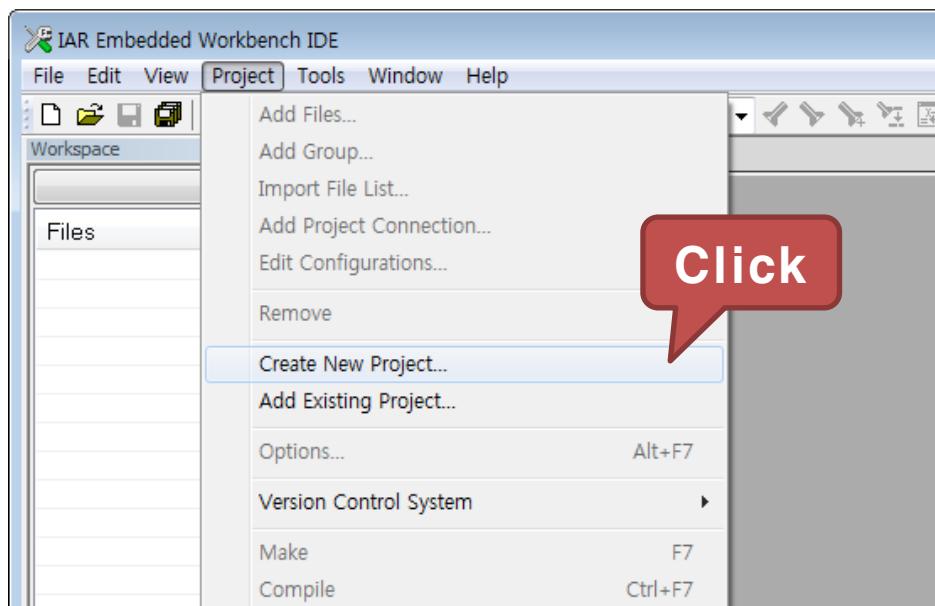
실습 2 : GPIO를 이용하여 스위치 입력 받기

- 구동 프로그램 : 사전 지식
 - 스위치를 누르면 "1"신호가 나오고 놓으면 "0"신호가 나옴
 - 이 신호를 입력 받기 위해서는
 - MCU의 입출력 포트를 입력으로 선언해야 함
 - 즉, 입력으로 사용하기로 한 MCU B 포트를 입력으로 선언해야 함
 - 입출력 포트를 입력으로 선언하려면 GPIOx_MODER 레지스터를 설정
 - 스위치 모듈의 버튼을 누른다면 GPIOx_IDR 레지스터(여기서는 B 포트를 사용하므로 GPIOB_IDR 레지스터)에 '1'이라는 값이 입력되어 들어옴
 - LED 출력 방법은 앞의 예제와 동일

실습 2 : GPIO를 이용하여 스위치 입력 받기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch3” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “switch”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : GPIO를 이용하여 스위치 입력 받기

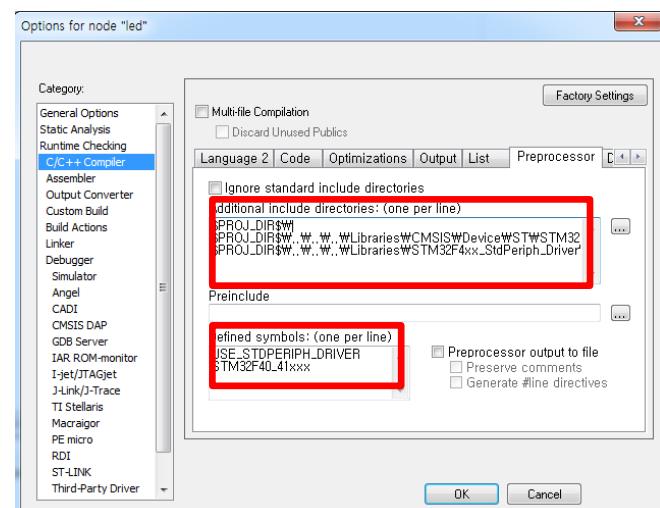
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch3\switch	system_stm32f4xx.c
Cortex_Example\Projects\ch3\switch	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 2 : GPIO를 이용하여 스위치 입력 받기

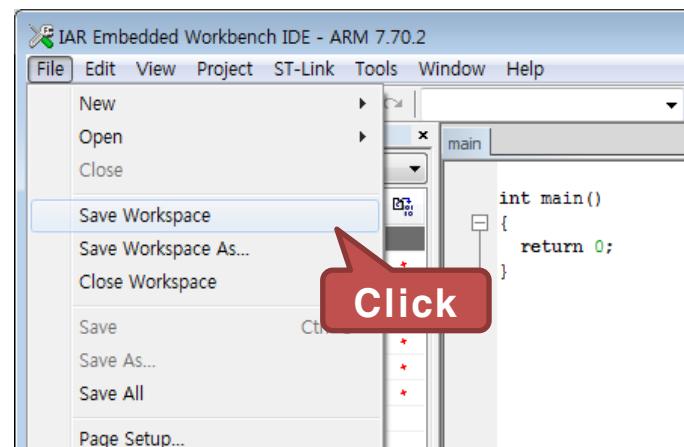
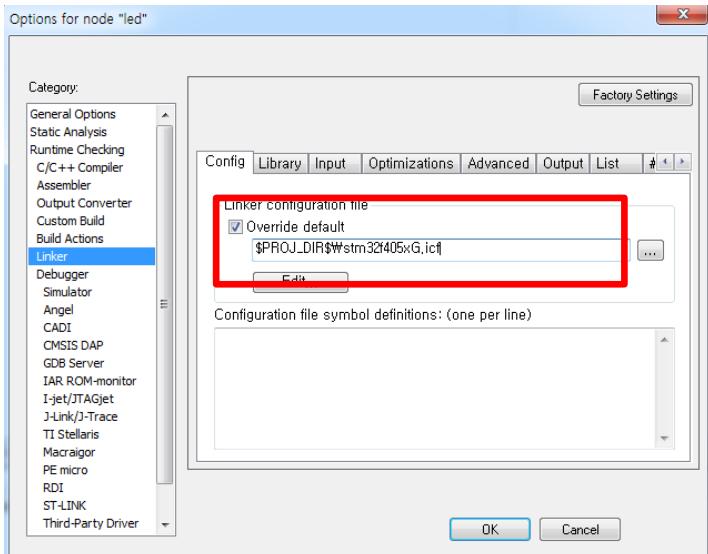
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : GPIO를 이용하여 스위치 입력 받기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : GPIO를 이용하여 스위치 입력 받기

- 구동 프로그램

- main.c 코드 작성

```
// stm32f4xx의 각 레지스터들을 정의한 헤더파일
#include "stm32f4xx.h"

int main(void){
    GPIO_InitTypeDef     GPIO_InitStructure;

    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOC,ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    // C 포트 하위 4비트를 출력으로 선언
    GPIO_InitStructure.GPIO_Pin =
                    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
```

실습 2 : GPIO를 이용하여 스위치 입력 받기

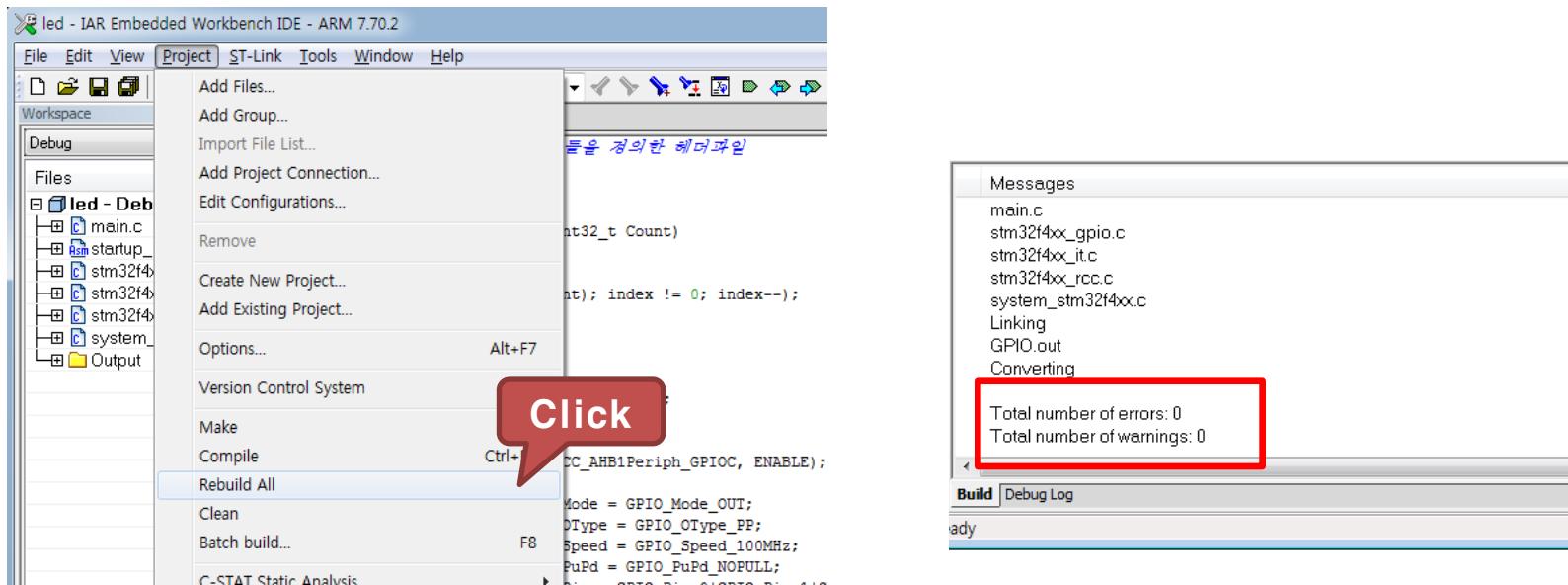
- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;  
// B 포트 상위 4비트를 입력으로 선언  
GPIO_InitStructure.GPIO_Pin =  
    GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;  
GPIO_Init(GPIOB, &GPIO_InitStructure);  
  
while(1)  
{  
    // B 포트 상위 4비트에 입력된 값을 포트 C의 하위 4비트로 바로 출력  
    GPIO_Write(GPIOC, (GPIO_ReadInputData(GPIOB)&0xF000)>>12);  
}  
}
```

실습 2 : GPIO를 이용하여 스위치 입력 받기

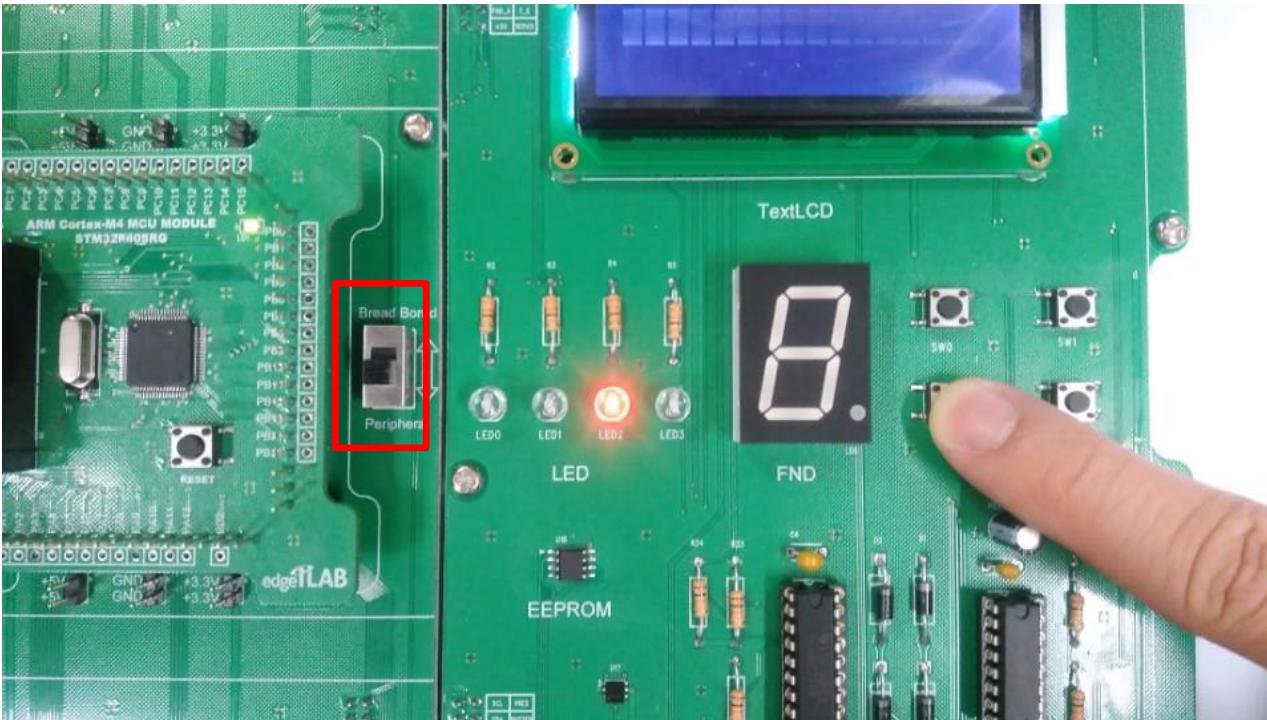
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 switch 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 2 : GPIO를 이용하여 스위치 입력 받기

- 실행 결과
 - Switch 모듈의 눌러진 버튼과 같은 LED의 불이 점등
 - 각 버튼을 누르면서 확인

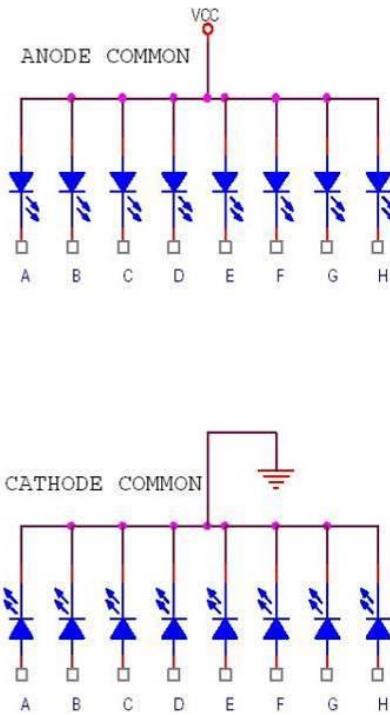
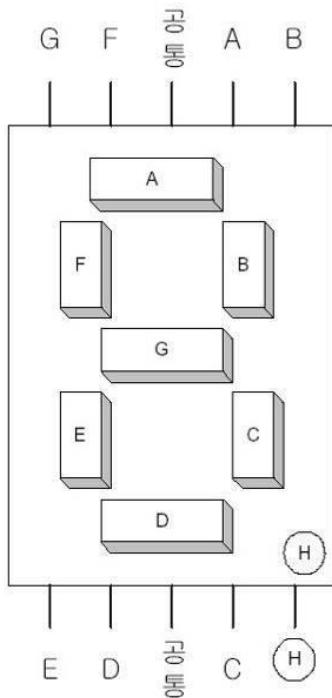


실습 3 : GPIO를 이용하여 FND LED 켜기

- 실습 개요
 - 단순 LED가 아닌 FND(Flexible Numeric Display: 7-Segment LED)를 이용하여 숫자를 표시하기
 - 마이크로컨트롤러의 포트를 출력으로 선언하고, 이 포트를 Array FND 모듈의 신호선 포트에 연결함
 - 일정 시간마다 클럭에 의해 Array FND에 숫자와 문자가 디스플레이 되도록 함
- 실습 목표
 - GPIO 입출력 포트의 방향 제어 및 출력 제어 방법 습득
 - FND LED 동작원리 습득

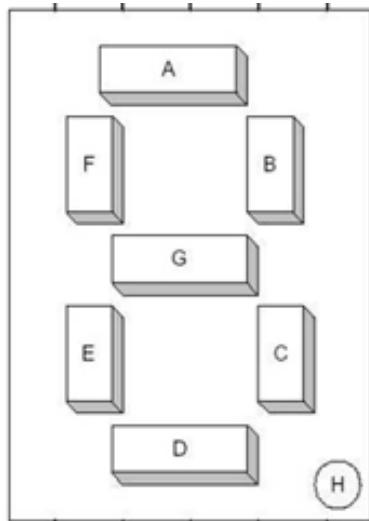
실습 3 : GPIO를 이용하여 FND LED 켜기

- FND(7-Segment LED) 구조
 - 7-세그먼트는 LED 8개를 그림과 같이 배열
 - 숫자나 간단한 기호 표현에 많이 사용
- 공통(Common)단자에 인가되는 전원에 따라서 Common Anode(+공통)과Common Cathode(-공통)으로 분류



실습 3 : GPIO를 이용하여 FND LED 켜기

- FND(7-Segment LED) 구동방법
 - Common-Cathode방식 : 각 단자에 '1'이 입력되면 해당 LED가 켜짐

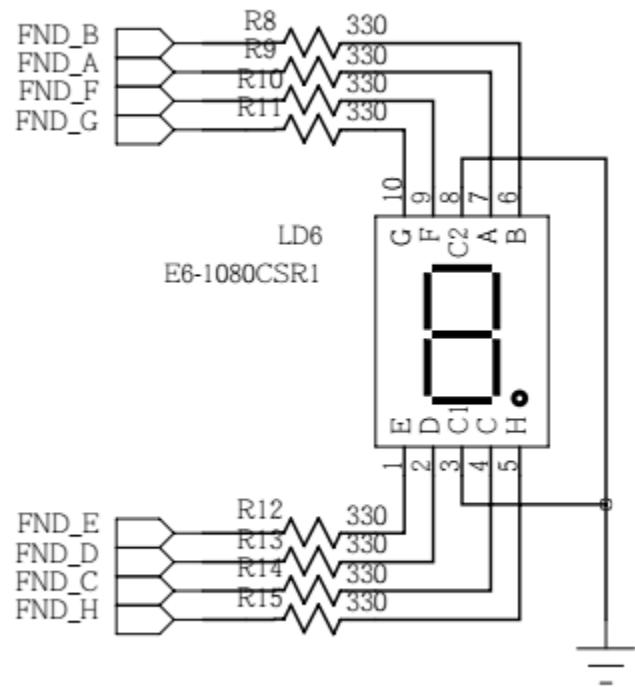


7-Segment
에서 16진수
표시방법

16 진수	7-세그먼트의 비트값								데이터 값 (HEX)
	H	G	F	E	D	C	B	A	
0	0	0	1	1	1	1	1	1	0X3F
1	0	0	0	0	0	1	1	0	0X06
2	0	1	0	1	1	0	1	1	0X5B
3	0	1	0	0	1	1	1	1	0X4F
4	0	1	1	0	0	1	1	0	0X66
5	0	1	1	0	1	1	0	1	0X6D
6	0	1	1	1	1	1	0	1	0X7D
7	0	0	1	0	0	1	1	1	0X27
8	0	1	1	1	1	1	1	1	0X7F
9	0	1	1	0	1	1	1	1	0X6F
A	0	1	1	1	0	1	1	1	0X77
B	0	1	1	1	1	1	0	0	0X7C
C	0	0	1	1	1	0	0	1	0X39
D	0	1	0	1	1	1	1	0	0X5E
E	0	1	1	1	1	0	0	1	0X79
F	0	1	1	1	0	0	0	1	0X71

실습 3 : GPIO를 이용하여 FND LED 켜기

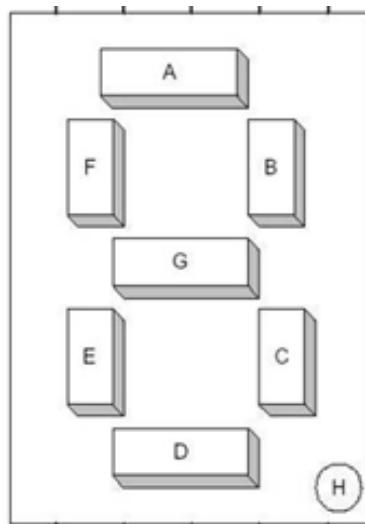
- 사용 모듈의 회로
 - FND 모듈(Common-Cathode)



실습 3 : GPIO를 이용하여 FND LED 켜기

- 구동 프로그램 : 사전지식

- MCU 모듈의 C포트를 FND의 불을 켜기 위한 출력 포트로 설정
 - GPIOx_MODER, GPIOx_OTYPER 레지스터를 설정
- 표를 참조하여 GPIOx_ODR(여기서는 GPIOC_ODR 레지스터)에 '1'을 출력

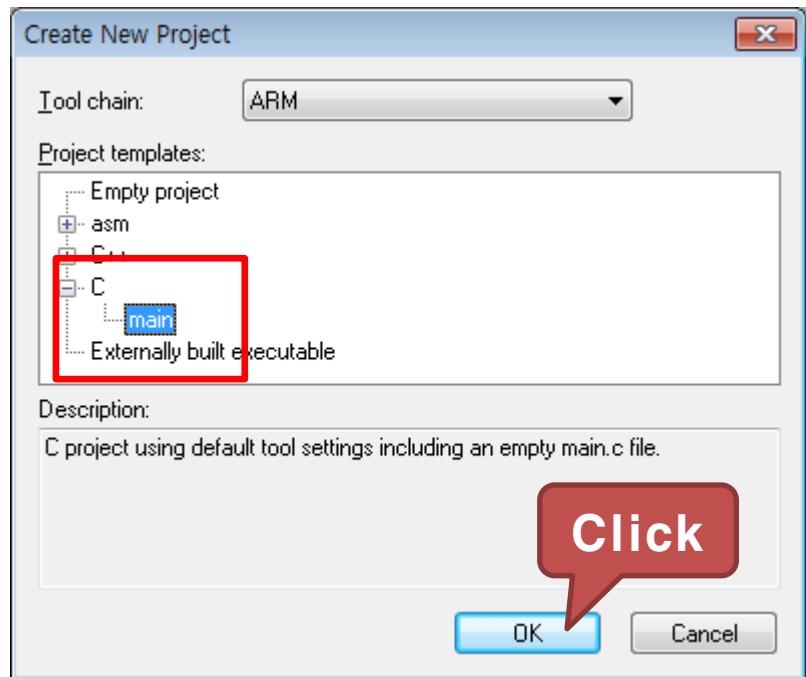
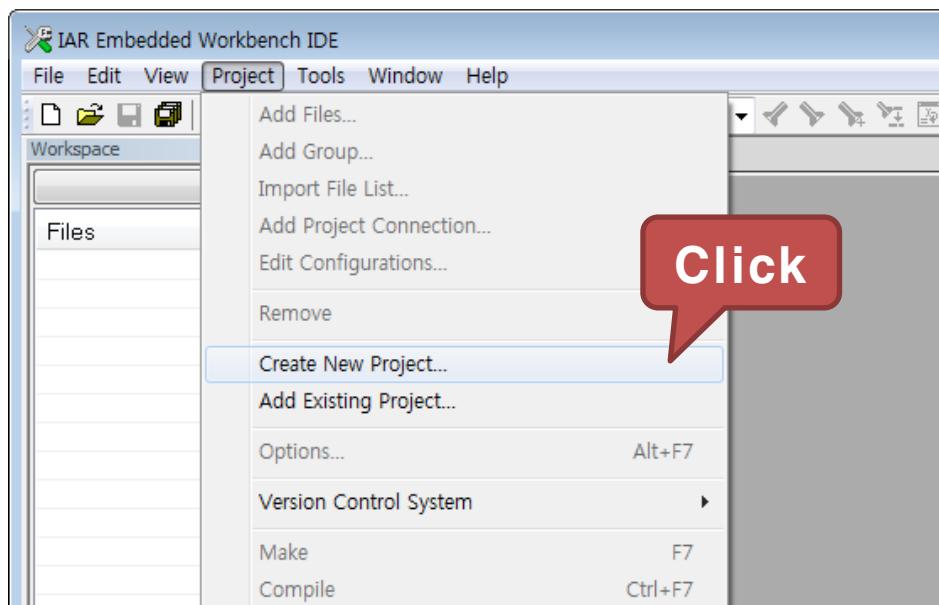


16 진수	7-세그먼트의 비트값								데이터 값 (HEX)
	H	G	F	E	D	C	B	A	
0	0	0	1	1	1	1	1	1	0X3F
1	0	0	0	0	0	1	1	0	0X06
2	0	1	0	1	1	0	1	1	0X5B
3	0	1	0	0	1	1	1	1	0X4F
4	0	1	1	0	0	1	1	0	0X66
5	0	1	1	0	1	1	0	1	0X6D
6	0	1	1	1	1	1	0	1	0X7D
7	0	0	1	0	0	1	1	1	0X27
8	0	1	1	1	1	1	1	1	0X7F
9	0	1	1	0	1	1	1	1	0X6F
A	0	1	1	1	0	1	1	1	0X77
B	0	1	1	1	1	1	0	0	0X7C
C	0	0	1	1	1	0	0	1	0X39
D	0	1	0	1	1	1	1	0	0X5E
E	0	1	1	1	1	0	0	1	0X79
F	0	1	1	1	0	0	0	1	0X71

실습 3 : GPIO를 이용하여 FND LED 켜기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch3” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “fnd”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 3 : GPIO를 이용하여 FND LED 켜기

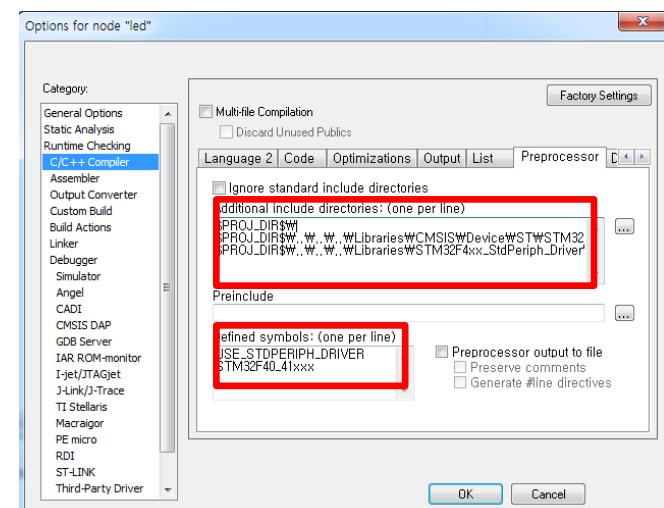
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch3\fnd	system_stm32f4xx.c
Cortex_Example\Projects\ch3\fnd	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 3 : GPIO를 이용하여 FND LED 켜기

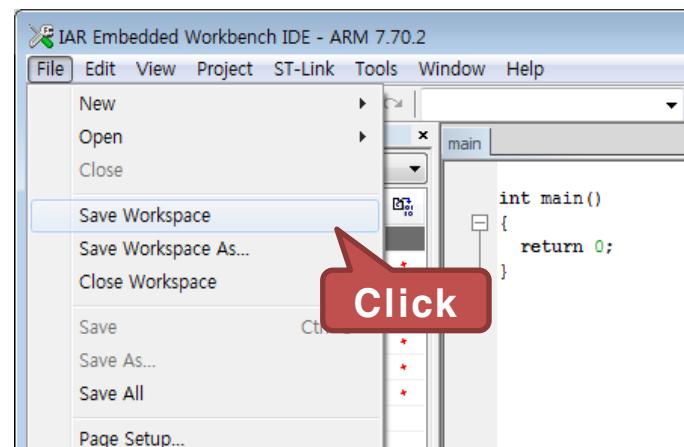
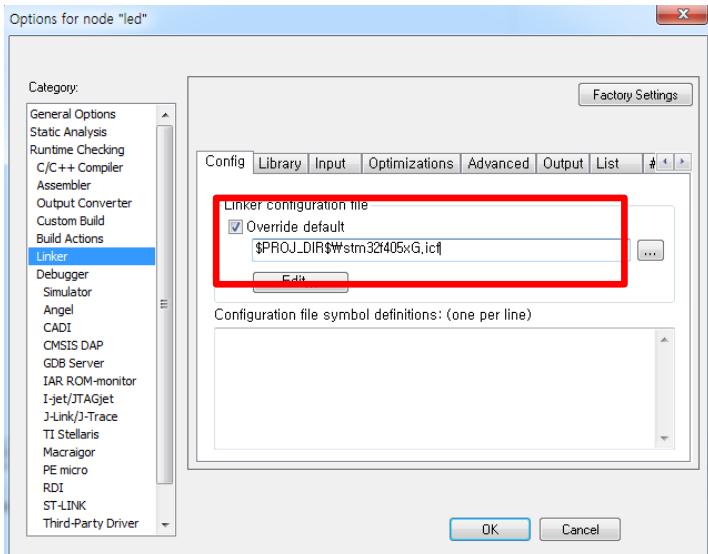
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 3 : GPIO를 이용하여 FND LED 켜기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 3 : GPIO를 이용하여 FND LED 켜기

- 구동 프로그램

- main.c 코드 작성

```
// stm32f4xx의 각 레지스터들을 정의한 헤더파일
#include "stm32f4xx.h"

// delay 함수
static void Delay(const uint32_t Count) {
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);

}

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    // 7-Segment에 표시할 글자의 입력 데이터를 저장
    unsigned int FND_DATA_TBL [] =
        {0x3F00, 0X0600, 0X5B00, 0X4F00, 0X6600, 0X6D00,
         0X7C00, 0X0700, 0X7F00, 0X6700, 0X7700, 0X7C00,
         0X3900, 0X5E00, 0X7900, 0X7100, 0X0800, 0X8000};

    unsigned char cnt=0;           // FND Table 카운터 변수
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
```

실습 3 : GPIO를 이용하여 FND LED 켜기

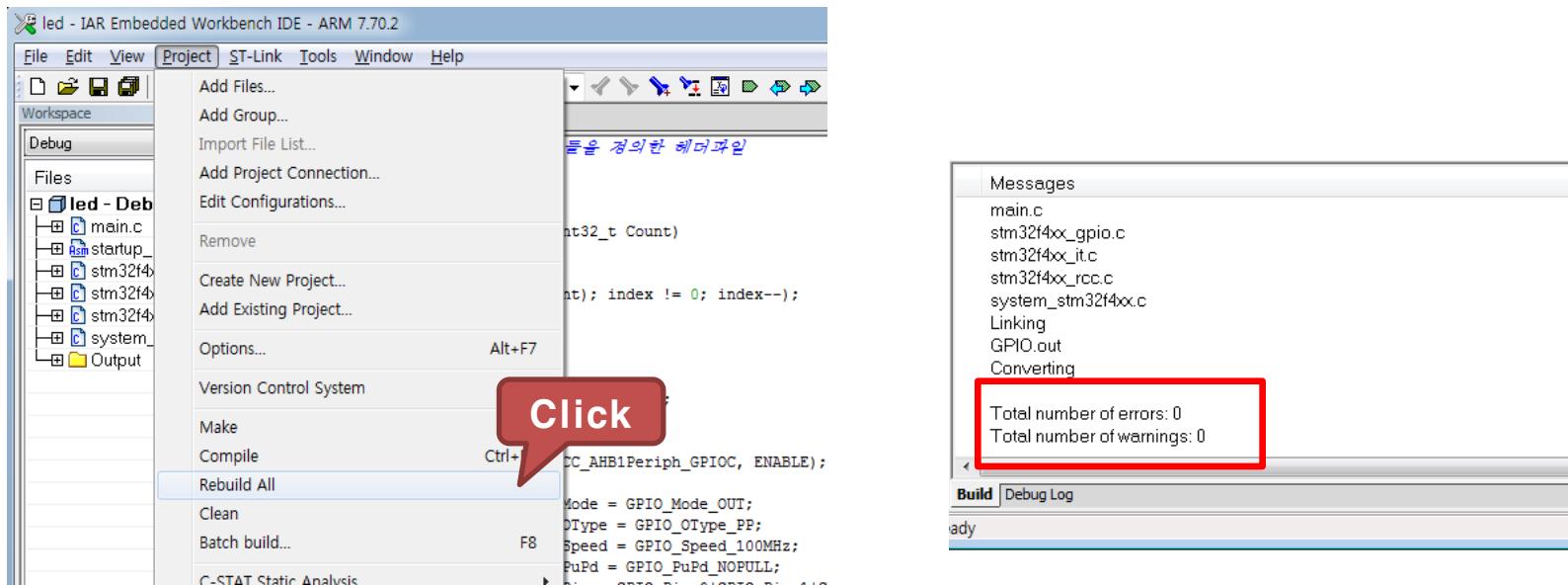
- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;  
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  
GPIO_InitStructure.GPIO_Pin =  
    GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11|  
    GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;  
GPIO_Init(GPIOC, &GPIO_InitStructure);  
while(1) {  
    GPIO_Write(GPIOC, FND_DATA_TBL[cnt]); // 포트 C에 값 출력  
    cnt++; // FND Table 카운터 변수를 증가  
    if(cnt>17) cnt=0; // 테이블 크기를 초과하는 경우 방지  
    Delay(500);  
}  
}
```

실습 3 : GPIO를 이용하여 FND LED 켜기

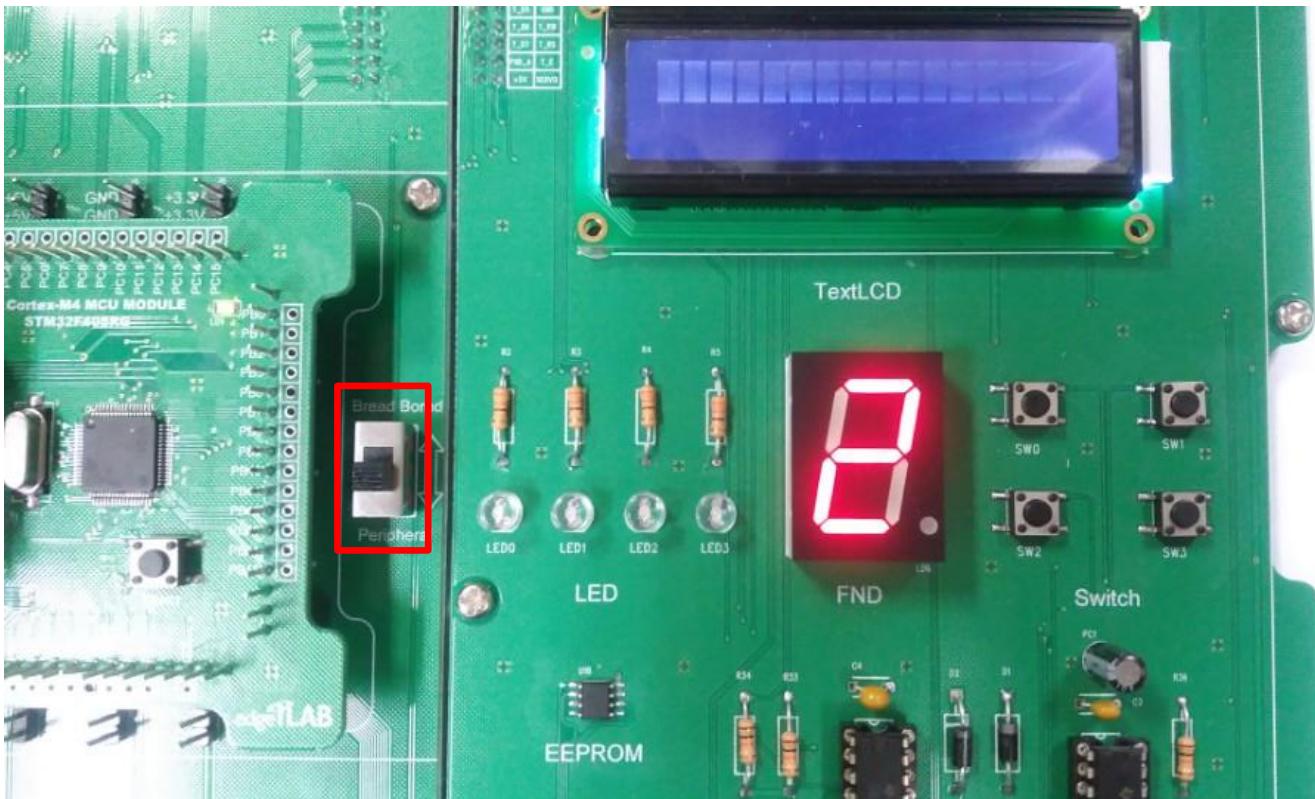
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 fnd 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 3 : GPIO를 이용하여 FND LED 켜기

- 실행 결과
 - 프로그램이 시작하면 500ms 마다 FND 에 0부터 9 , A ~ F 그리고 '_', ':' 을 순차적으로 출력





INTERNAL MEMORY의 이해

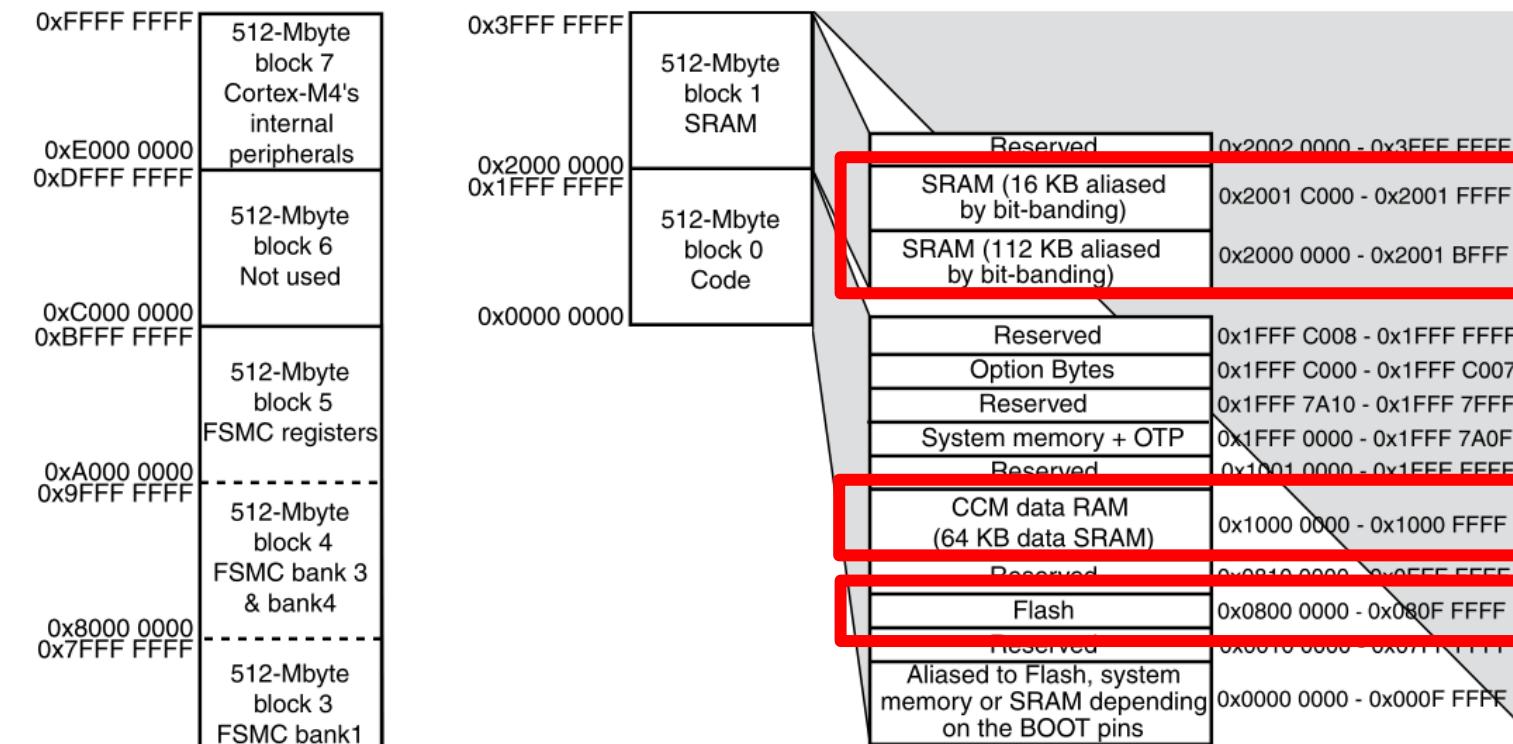
1. STM32F405의 데이터 메모리 구조
2. EWARM 디버깅 모드
3. EWARM 디버깅 모드로 변수값 확인하기



엣지아이랩

STM32F405의 데이터 메모리 구조

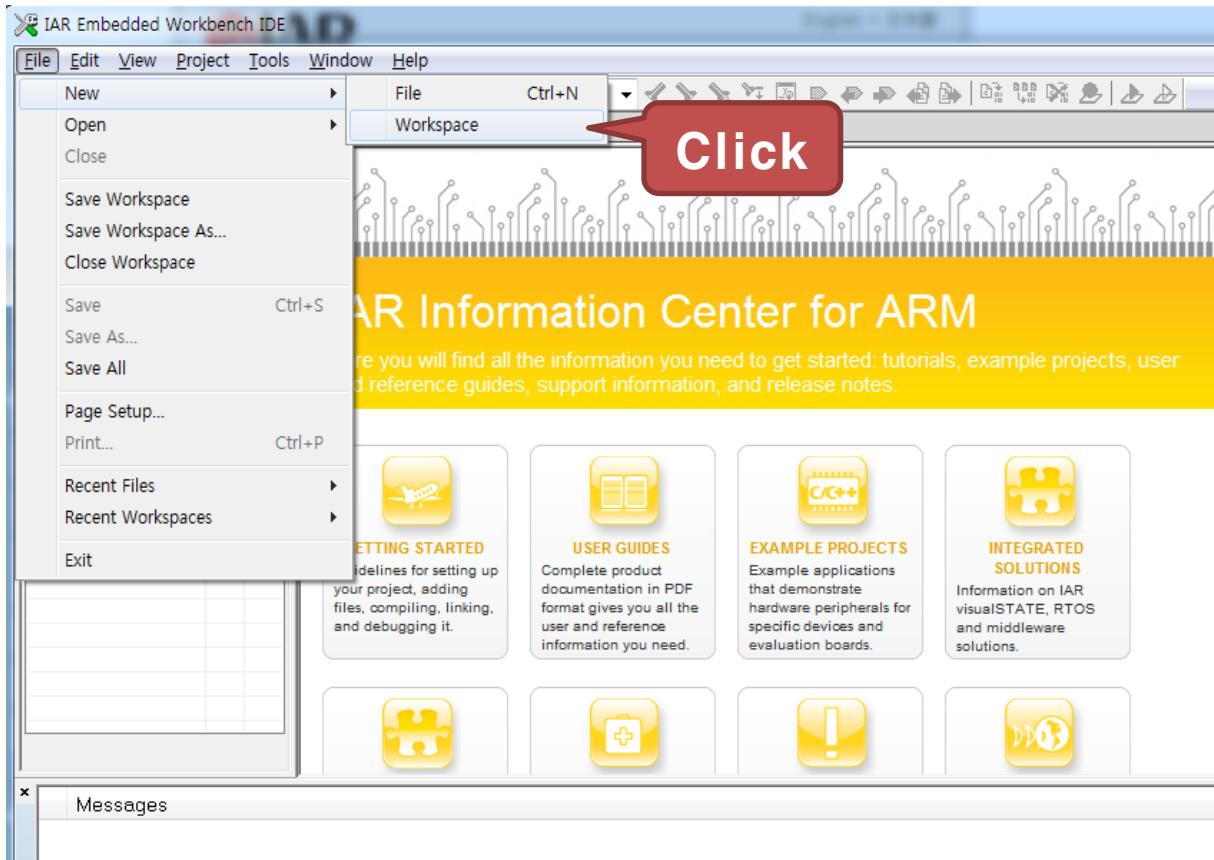
- STM32F405 내장 데이터 메모리



주소	레지스터 명
0x4002 0000 - 0x4002 03FF	GPIO Port A
0x4002 0400 - 0x4002 07FF	GPIO Port B
0x4002 0800 - 0x4002 0BFF	GPIO Port C

EWARM 디버깅 모드

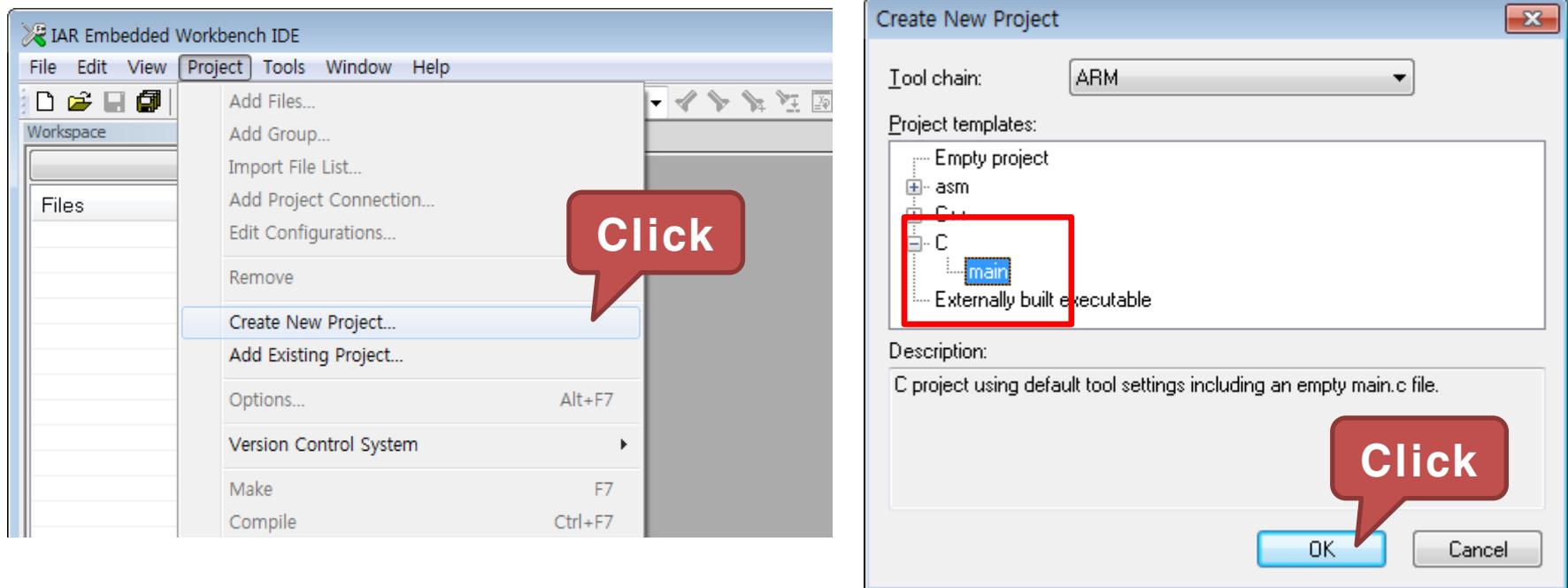
- IAR Embedded Workbench 프로그램을 실행
- 프로그램 좌측 상단의 “File” 탭에서 Open → “Workspace...”를 선택



EWARM 디버깅 모드

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch4” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “debugging”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



EWARM 디버깅 모드

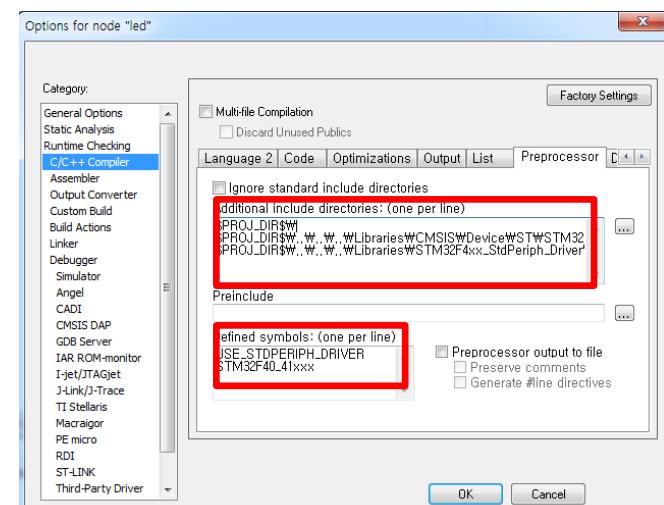
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch4\debugging	system_stm32f4xx.c
Cortex_Example\Projects\ch4\fdebugging	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

EWARM 디버깅 모드

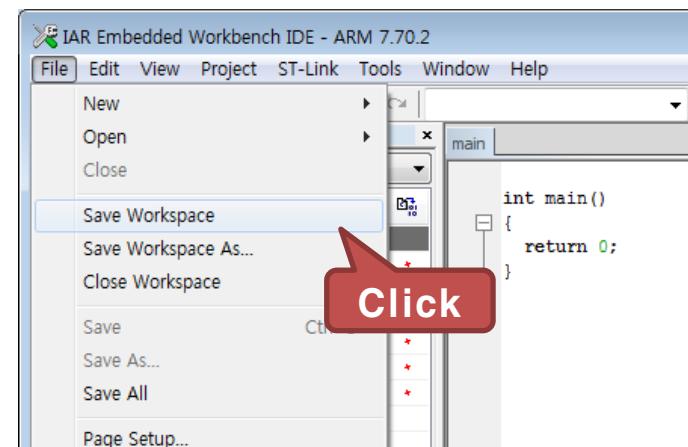
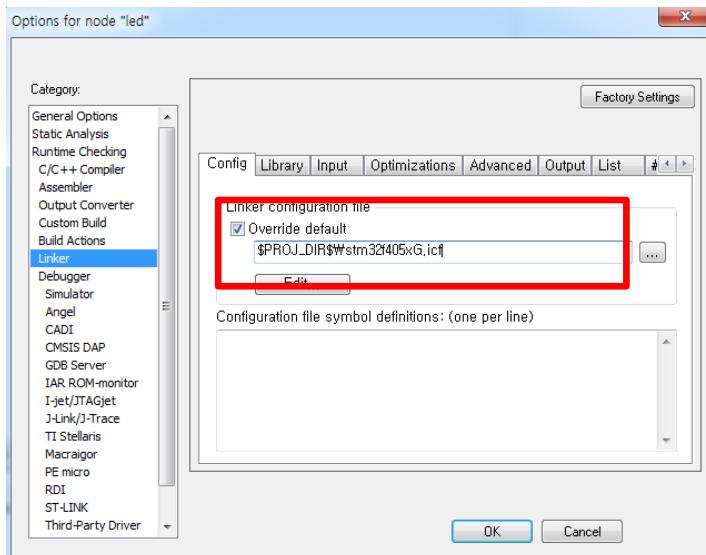
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



EWARM 디버깅 모드

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



EWARM 디버깅 모드

- 구동 프로그램
 - main.c 코드 작성

```
// stm32f4xx의 각 레지스터들을 정의한 헤더파일
#include "stm32f4xx.h"

// delay 함수
static void Delay(const uint32_t Count) {
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);

}

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    unsigned int LED_Data=0x0000;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
```

EWARM 디버깅 모드

- 구동 프로그램

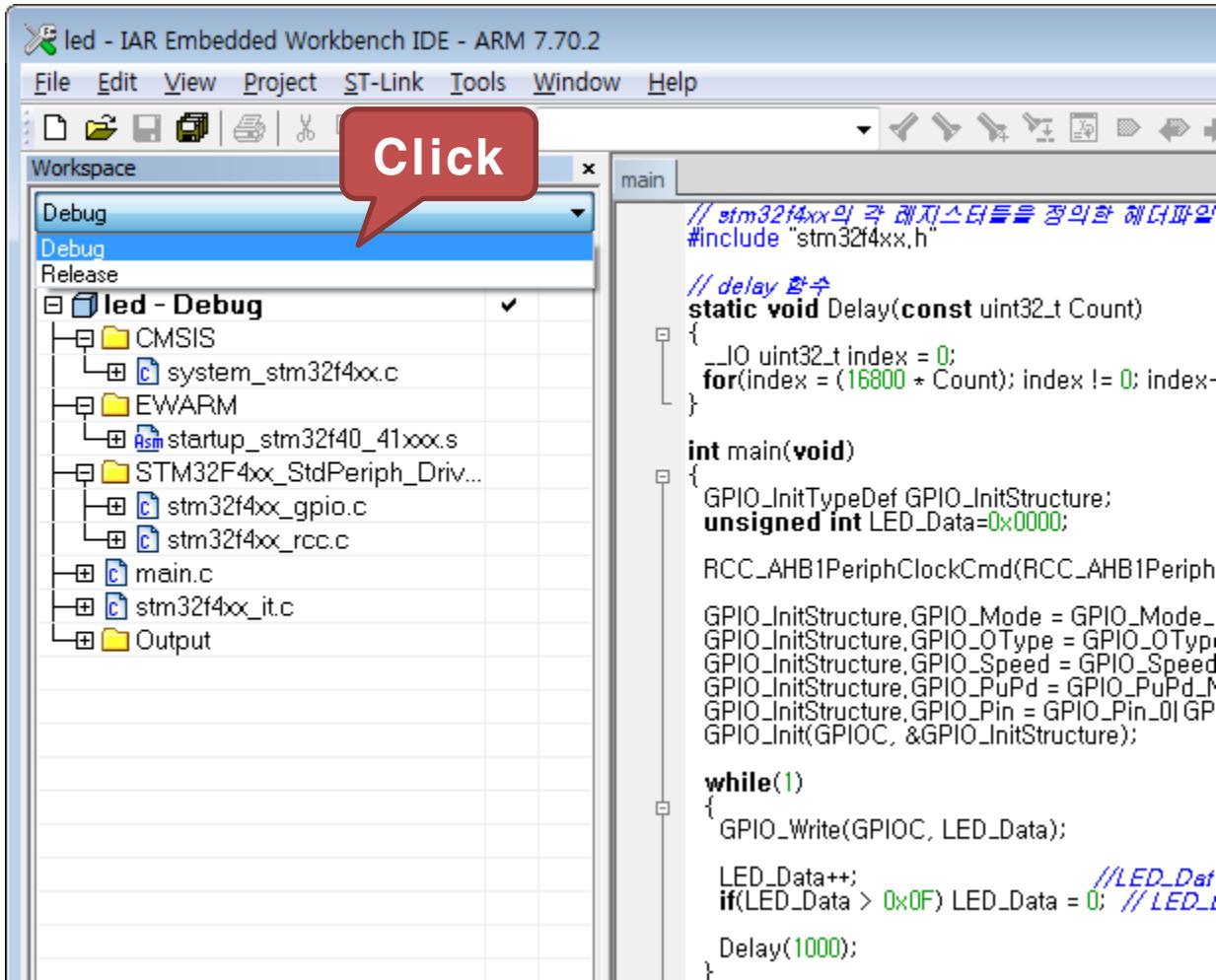
- main.c 코드 작성

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
GPIO_Init(GPIOC, &GPIO_InitStructure);

while(1) {
    GPIO_Write(GPIOC, LED_Data);
    LED_Data++;           // LED_Data값을 1씩 증가
    // LED_Data값이 0x0F보다 크면 0으로 초기화
    if(LED_Data > 0x0F) LED_Data = 0;
    Delay(1000);
}
```

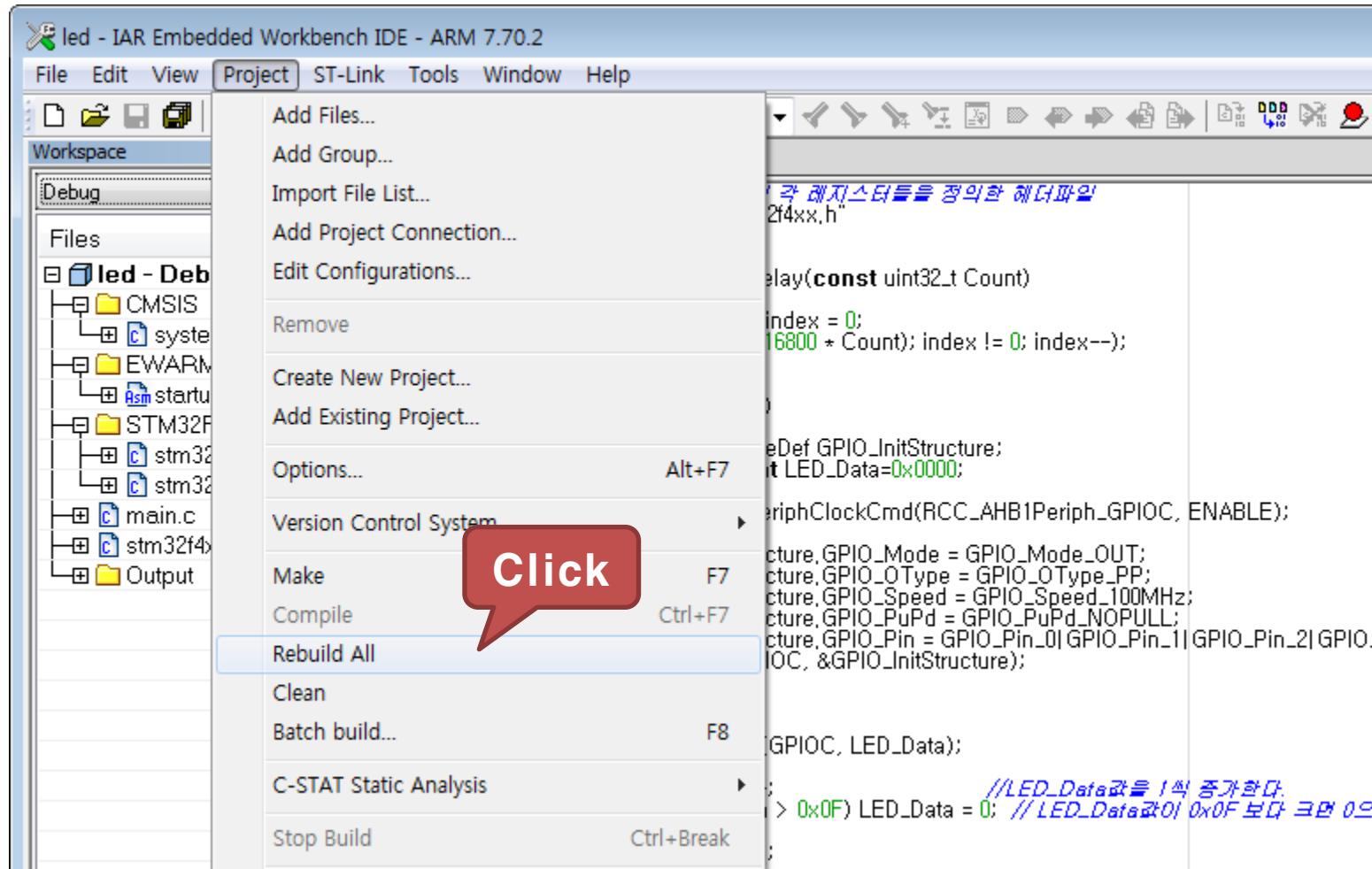
EWARM 디버깅 모드

- Workspace 탐색을 Debug로 전환



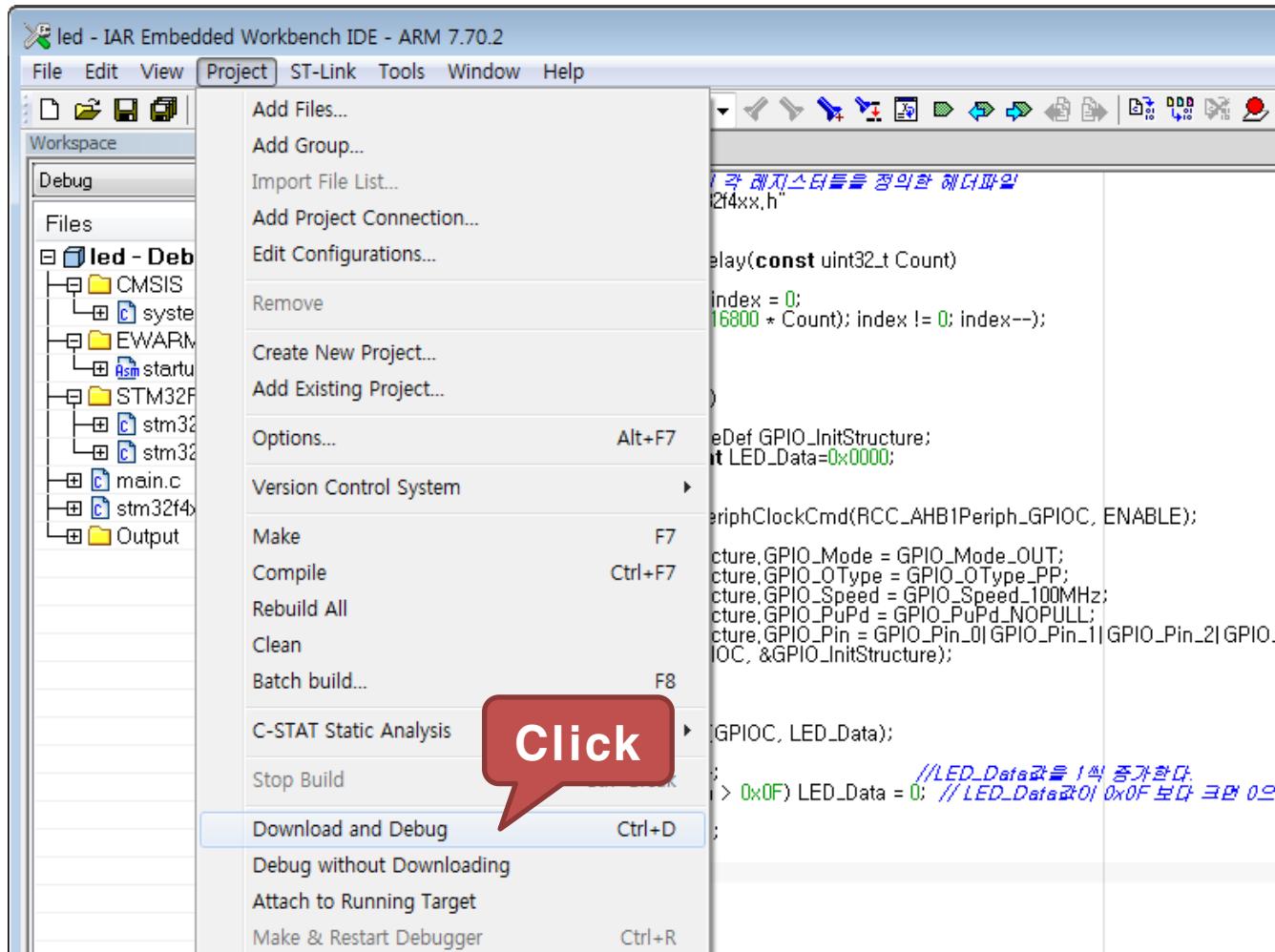
EWARM 디버깅 모드

● 프로젝트 빌드



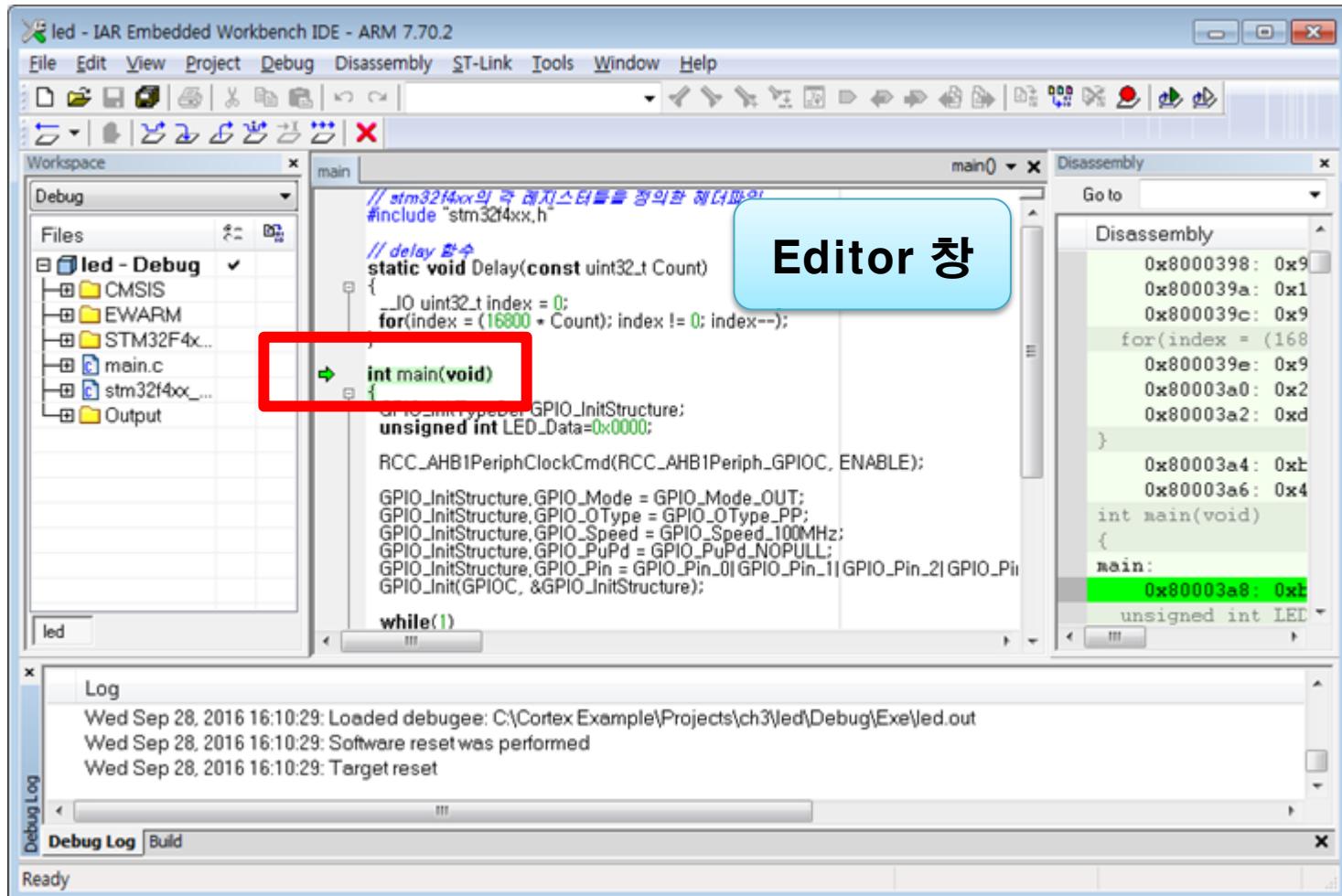
EWARM 디버깅 모드

- Project탭에서 “Download and Debug”를 선택하면 디버깅 모드 실행



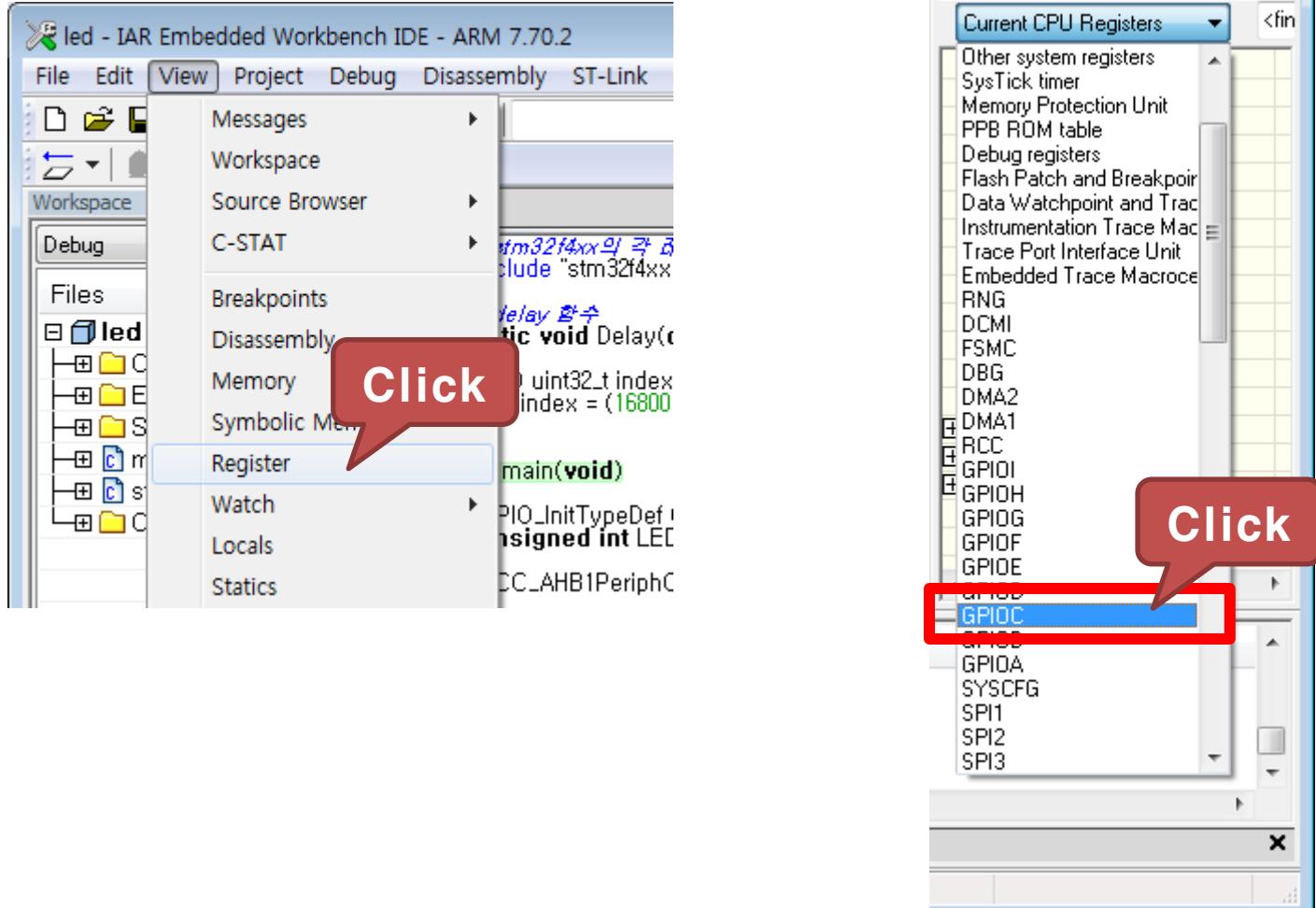
EWARM 디버깅 모드

- 디버깅 모드 실행 화면



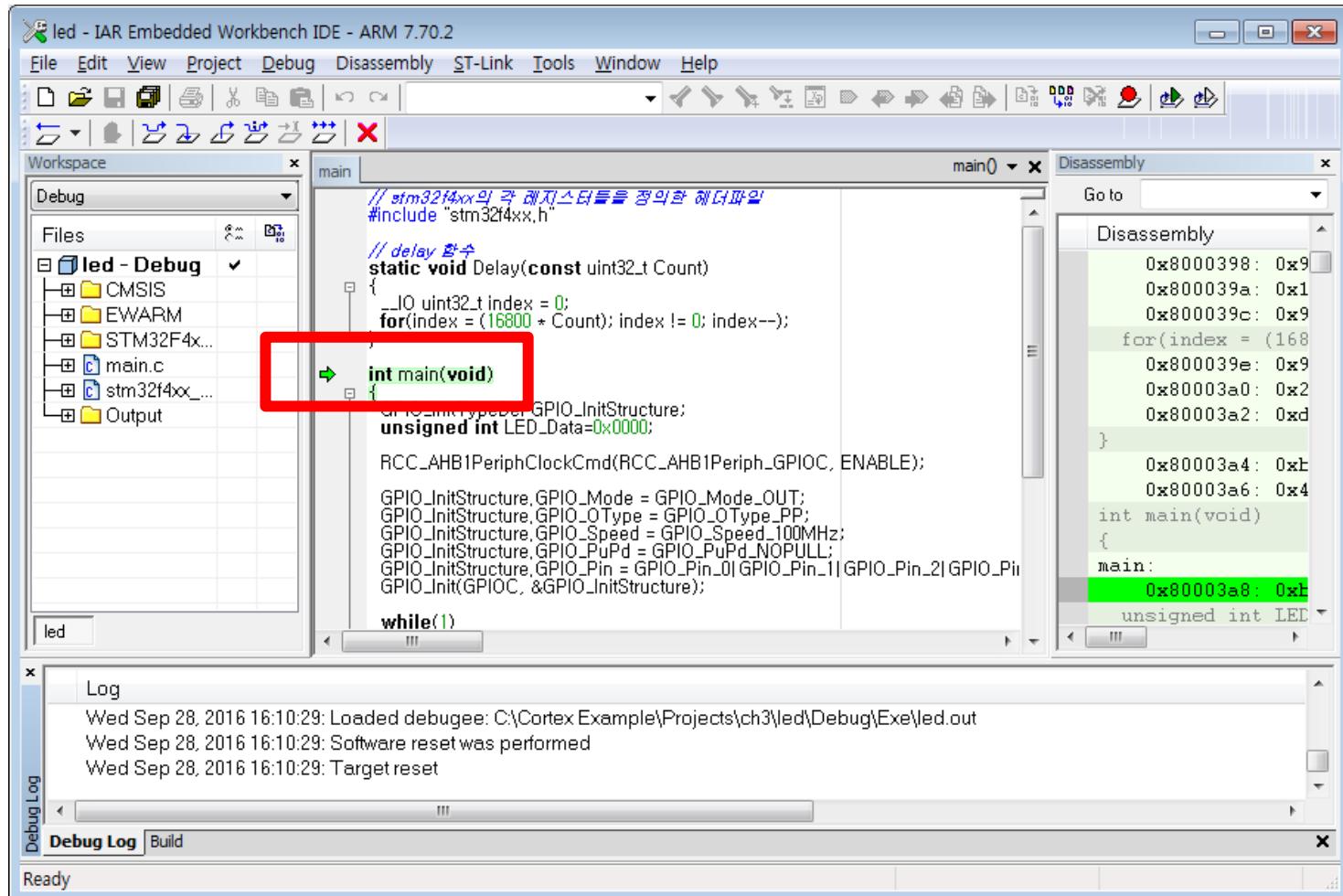
EWARM 디버깅 모드

- Register 창 실행 및 설정 방법



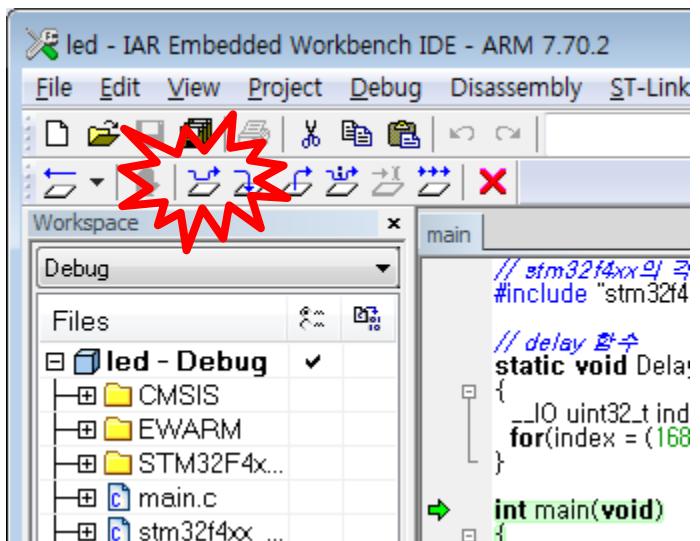
EWARM 디버깅 모드

● 디버깅 모드 실행 화면



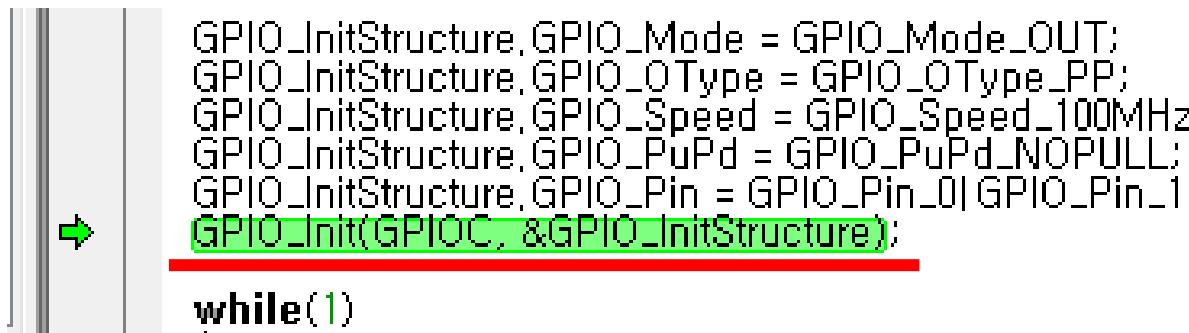
EWARM 디버깅 모드

- 프로그램을 실행하여 레지스터 값의 변화 관찰
- Debug 탭의 Step Into 버튼을 눌러 프로그램 명령 한 줄을 실행 또는 “F10” 누름



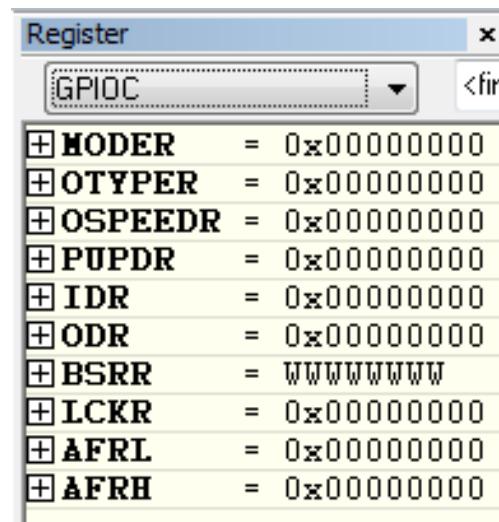
EWARM 디버깅 모드

- 아래 그림과 같이, 녹색 화살표가 GPIO_Init(GPIOC, &GPIO_InitStructure) 함수 앞에 올 때까지 F10, Step Over 명령을 실행



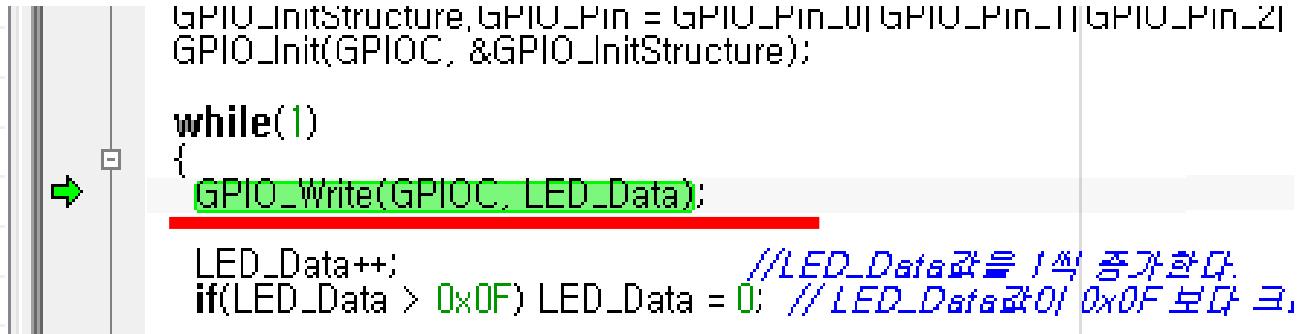
```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GI
GPIO_Init(GPIOC, &GPIO_InitStructure);
while(1)
```

- 포트 C의 레지스터에 아무런 값도 들어 있지 않음을 확인



EWARM 디버깅 모드

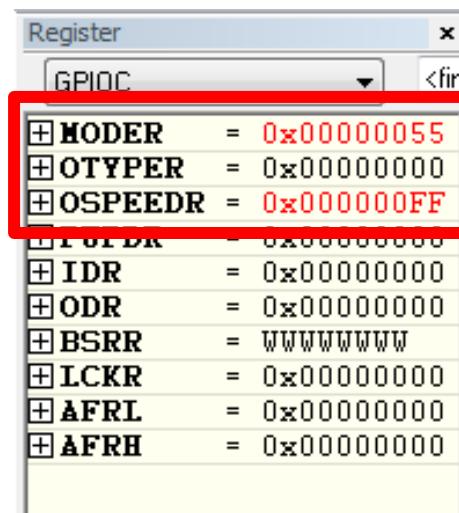
- 한 번 더 F10키 (Step Over)를 눌러, GPIO_Init() 명령 실행



```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2;
GPIO_Init(GPIOC, &GPIO_InitStructure);

while(1)
{
    GPIO_Write(GPIOC, LED_Data);
    LED_Data++;
    if(LED_Data > 0x0F) LED_Data = 0; //LED_Data값을 1씩 증가한다.
                                    //LED_Data값이 0x0F보다 크면,
```

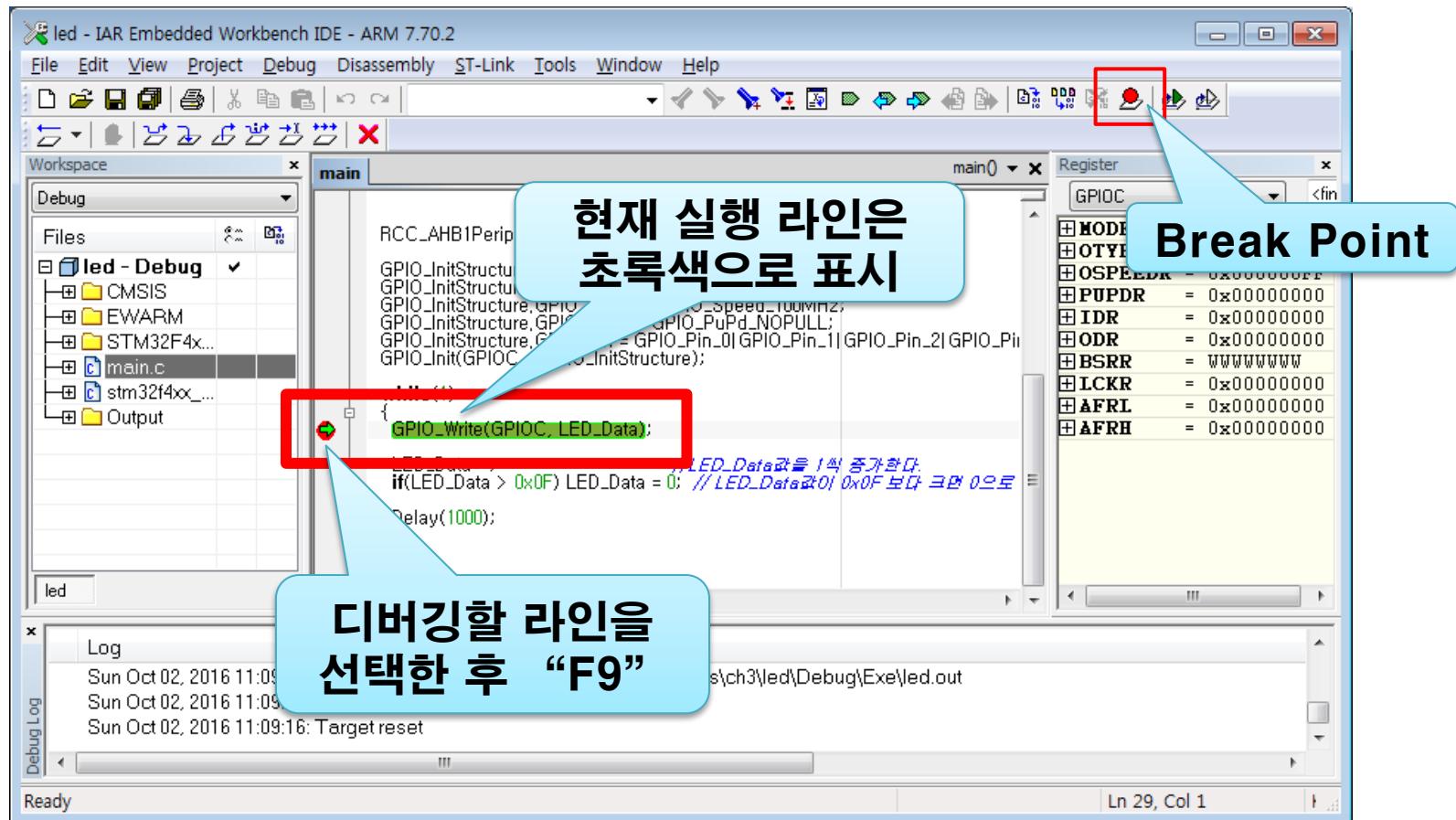
- 레지스터를 보니 예상대로 값이 저장되어 있음을 확인



EWARM 디버깅 모드

● Break Point

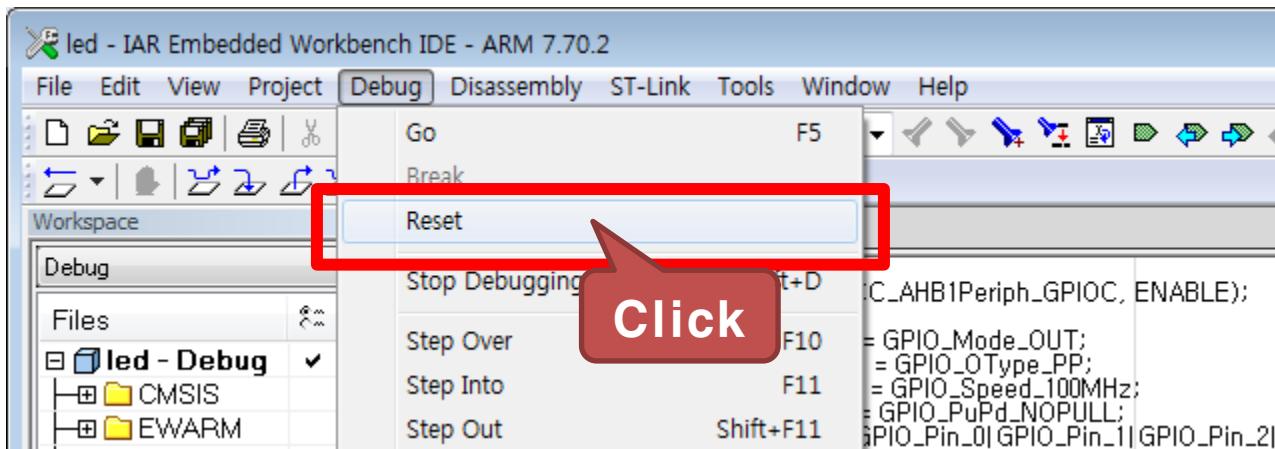
- 원하는 라인에 커서를 가져간 후 “F9” 키를 누르면 설정



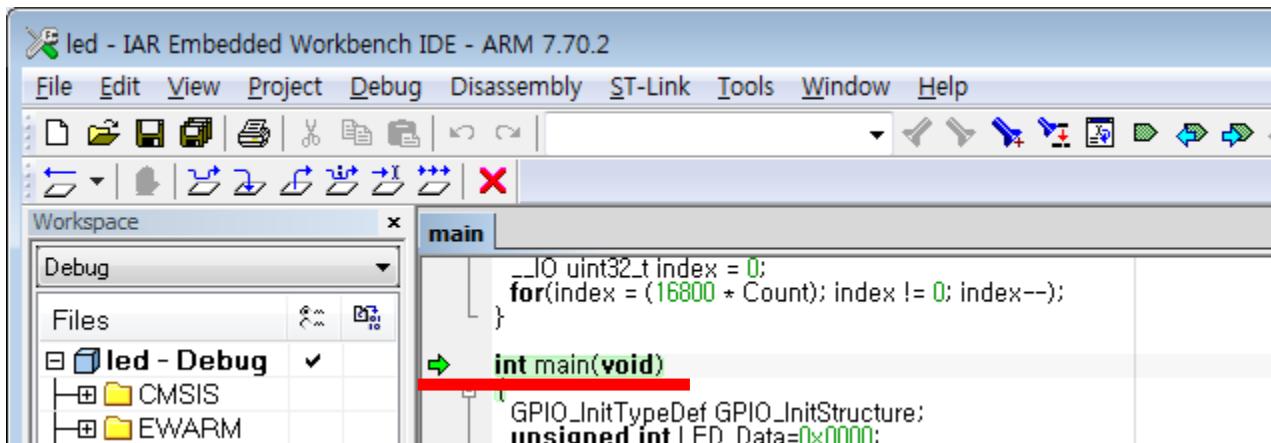
EWARM 디버깅 모드

- Break Point

- Reset 실행
- main() 함수 앞으로 화살표 위치



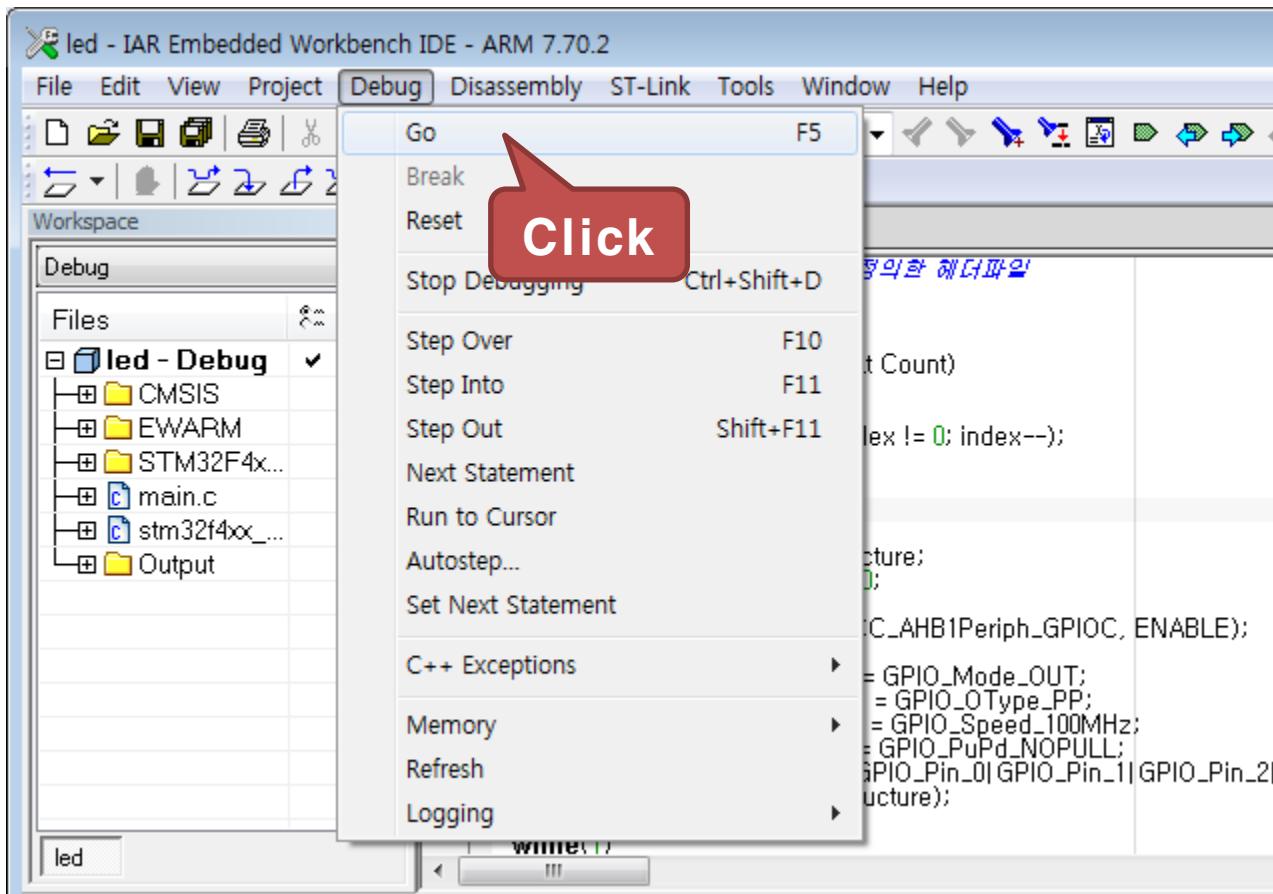
Click



EWARM 디버깅 모드

- Break Point

- Go 실행
- Break Point 위치까지 프로그램 실행



EWARM 디버깅 모드

- Break Point
 - Break 지점에서 멈춤

The screenshot shows the EWARM debugger interface. On the left is the code editor window titled 'main' with the file name 'main0'. The code initializes the GPIOC peripheral and then enters a infinite loop. A red circle highlights the first instruction of the loop, which is a call to `GPIO_Write`. To the right is the 'Register' window for the GPIOC peripheral. The `ODR` register is highlighted with a red box. Its value is shown as `0x00000000`, indicating that the LED is currently off.

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
GPIO_Init(GPIOC, &GPIO_InitStructure);

while(1)
{
    GPIO_Write(GPIOC, LED_Data);

    LED_Data++;
    if(LED_Data > 0xF) LED_Data = 0; // LED_Data값을 1씩 증가한다.
    // LED_Data값이 0x0F보다 크면 0으로
    Delay(1000);
}
```

Register	Value
<code>MODER</code>	<code>0x00000055</code>
<code>OTYPER</code>	<code>0x00000000</code>
<code>OSPEEDR</code>	<code>0x000000FF</code>
<code>PUPDR</code>	<code>0x00000000</code>
<code>IER</code>	<code>0x00000000</code>
<code>ODR</code>	<code>0x00000000</code>
<code>DDR</code>	<code>0x00000000</code>
<code>LCKR</code>	<code>0x00000000</code>
<code>AFRL</code>	<code>0x00000000</code>
<code>AFRH</code>	<code>0x00000000</code>

EWARM 디버깅 모드

- Break Point
 - Run(F5) 실행 후 Step Over(F10) 실행

The screenshot shows the EWARM IDE interface. On the left is the code editor with the file 'main.c' open. A red circle marks a break point at the line 'GPIO_Write(GPIOC, LED_Data);'. The code initializes the GPIOC peripheral and then enters a loop where it increments a variable 'LED_Data' and writes it to GPIOC. The Register window on the right shows the state of the GPIOC register. The 'ODR' register is highlighted with a red box and has a value of 0x00000001.

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

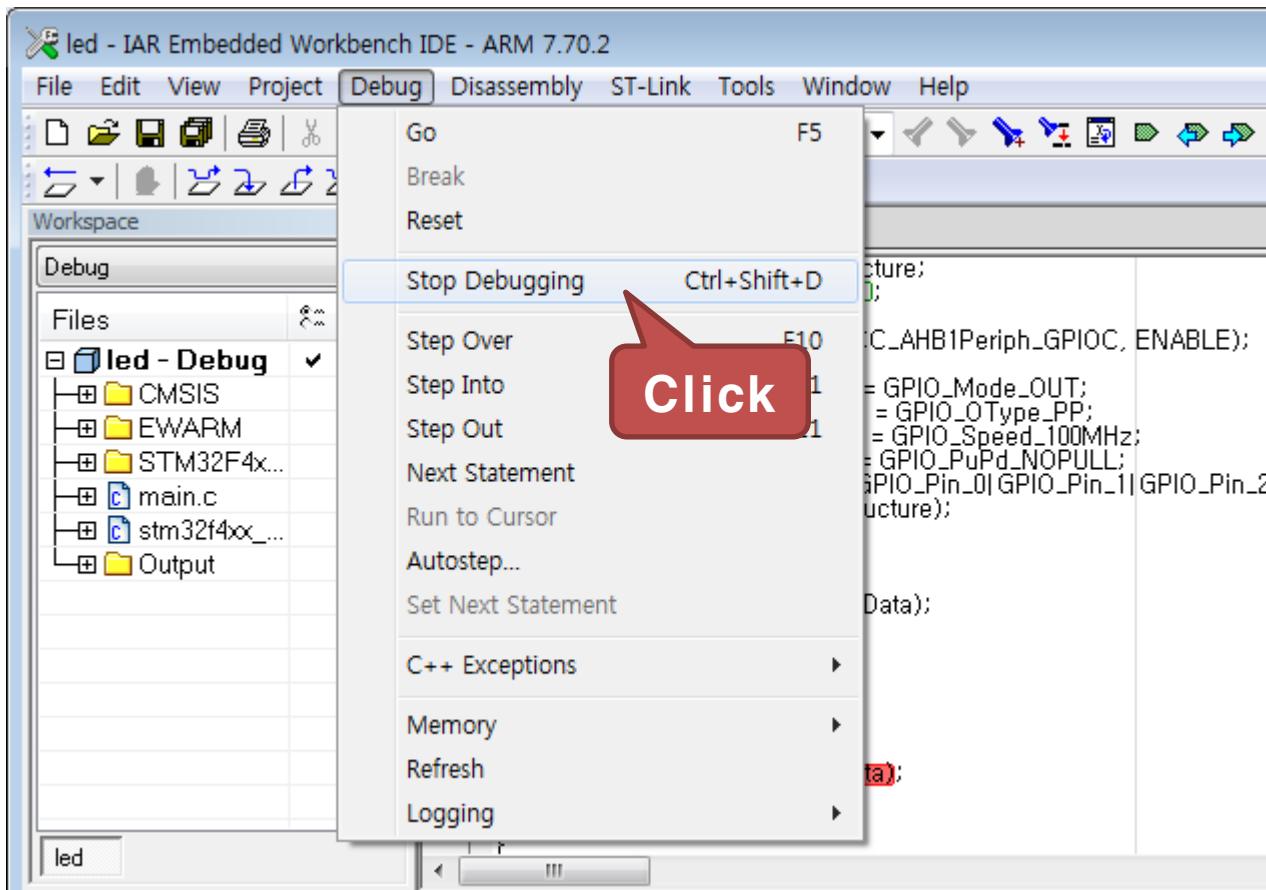
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
GPIO_Init(GPIOC, &GPIO_InitStructure);

while(1)
{
    GPIO_Write(GPIOC, LED_Data);
    LED_Data++; //LED_Data값을 1씩 증가한다.
    if(LED_Data > 0xF) LED_Data = 0; //LED_Data값이 0x0F보다 크면 0으로
    Delay(1000);
}
```

Register	Value
MODER	0x00000055
OTYPER	0x00000000
OSPEEDR	0x000000FF
PUPDR	0x00000000
IER	0x00000001
ODR	0x00000001
LCKR	0x00000000
AFRL	0x00000000
AFRH	0x00000000

EWARM 디버깅 모드로 변수값 확인하기

- Stop Debugging(Ctrl+Shift+D)
 - 디버깅 모드 종료



EWARM 디버깅 모드로 변수값 확인하기

- 예제 코드 수정
 - 다음과 같이 예제 코드를 수정

```
main
static void Delay(const uint32_t Count)
{
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--)
}

unsigned int Plus_Value = 0x00000000;
unsigned int Minus_Value = 0xFFFFFFFF;

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    unsigned int LED_Data=0x0000;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

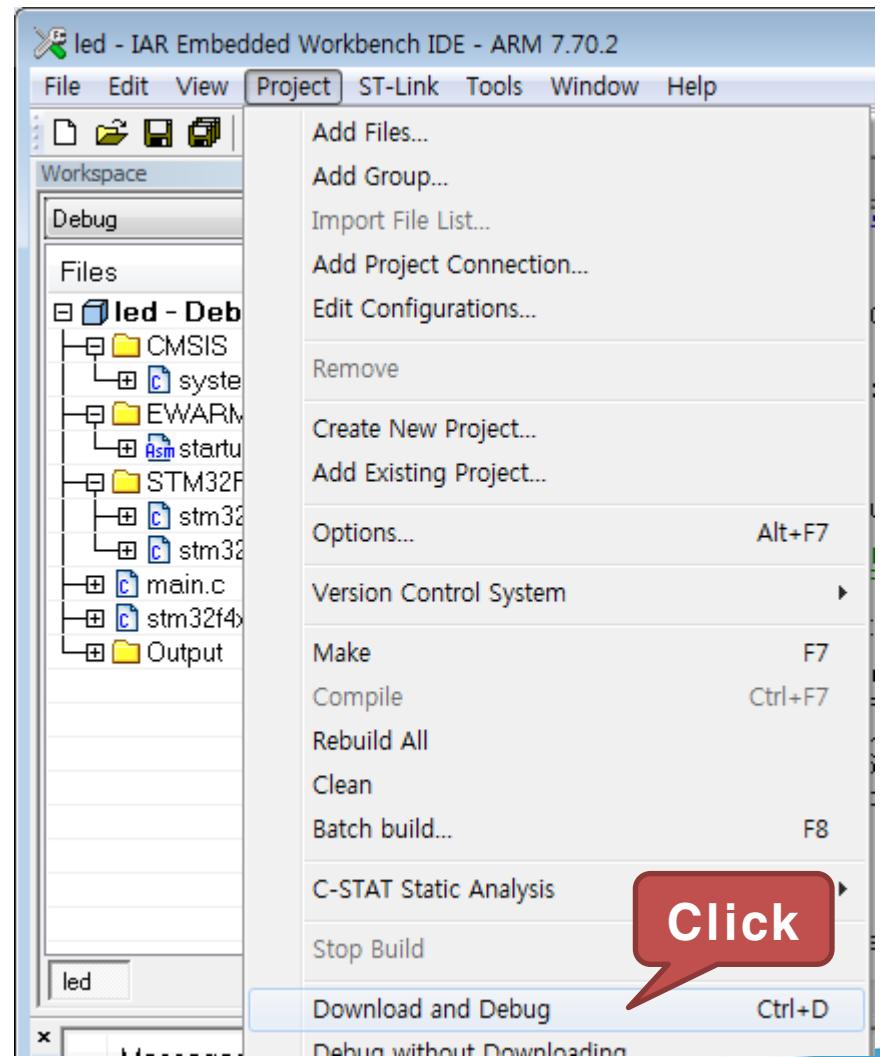
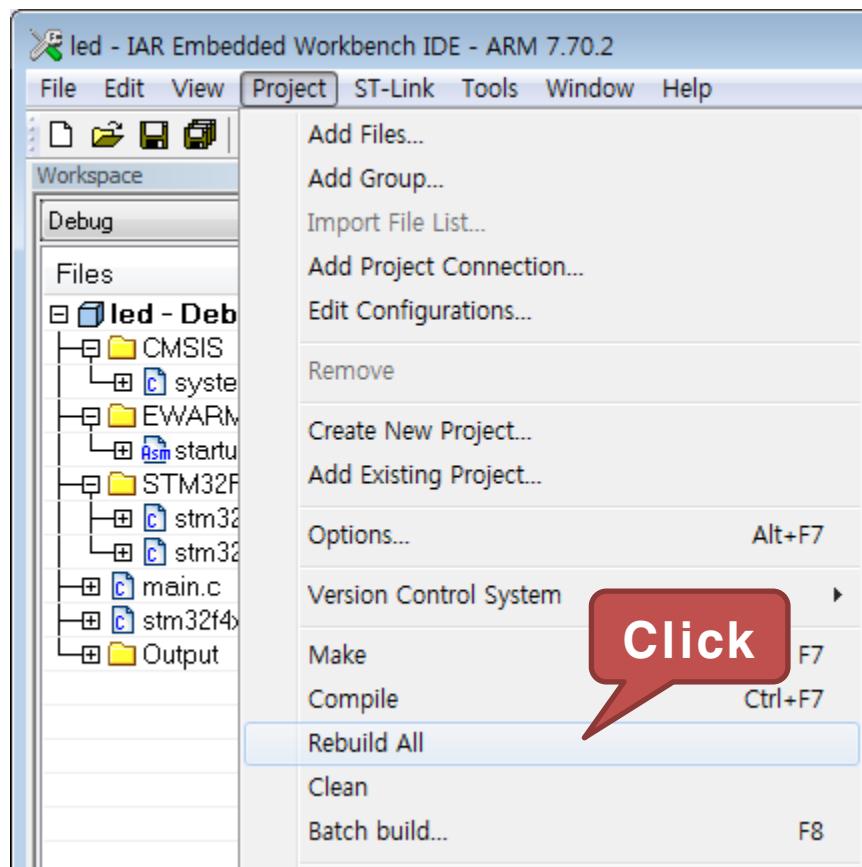
    while(1)
    {
        Plus_Value++;
        Minus_Value--;
        GPIO_Write(GPIOC, LED_Data);

        LED_Data++; //LED_Data값을 1씩 증가한다.
        if(LED_Data > 0xF) LED_Data = 0; //LED_Data값이 0xF보다 크면 0으로 초기화한다.

        Delay(1000);
    }
}
```

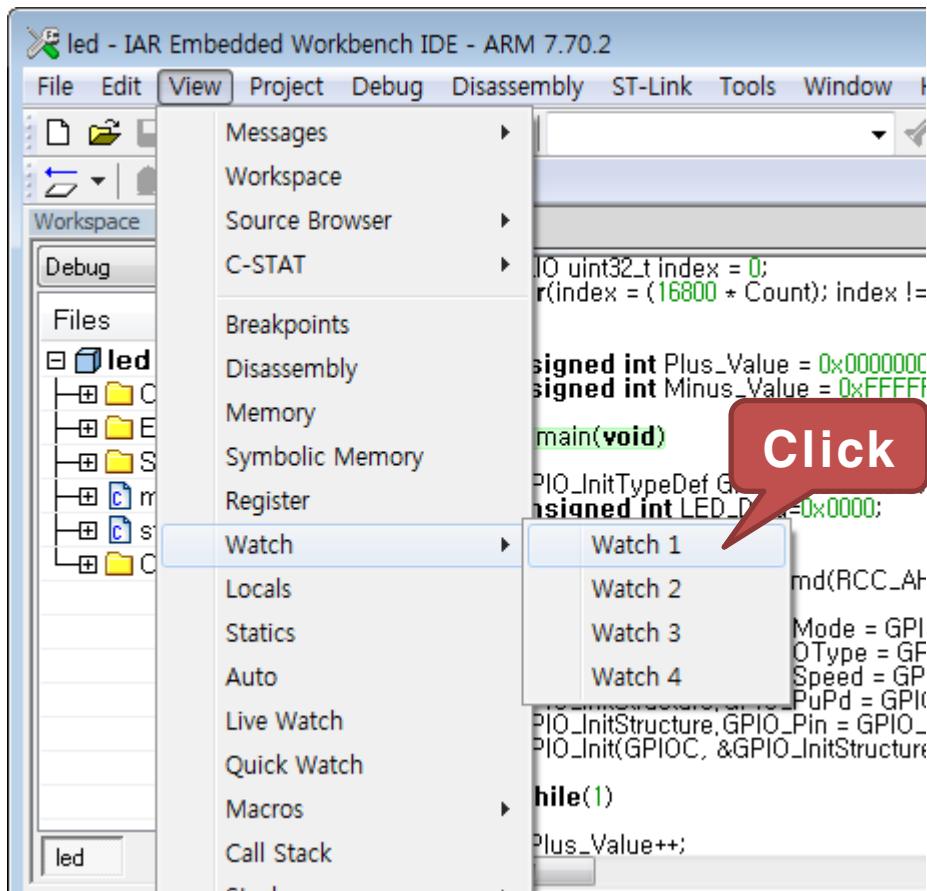
EWARM 디버깅 모드로 변수값 확인하기

- 예제 코드 다시 빌드
 - 예제 코드 빌드 후 디버깅 모드 시작



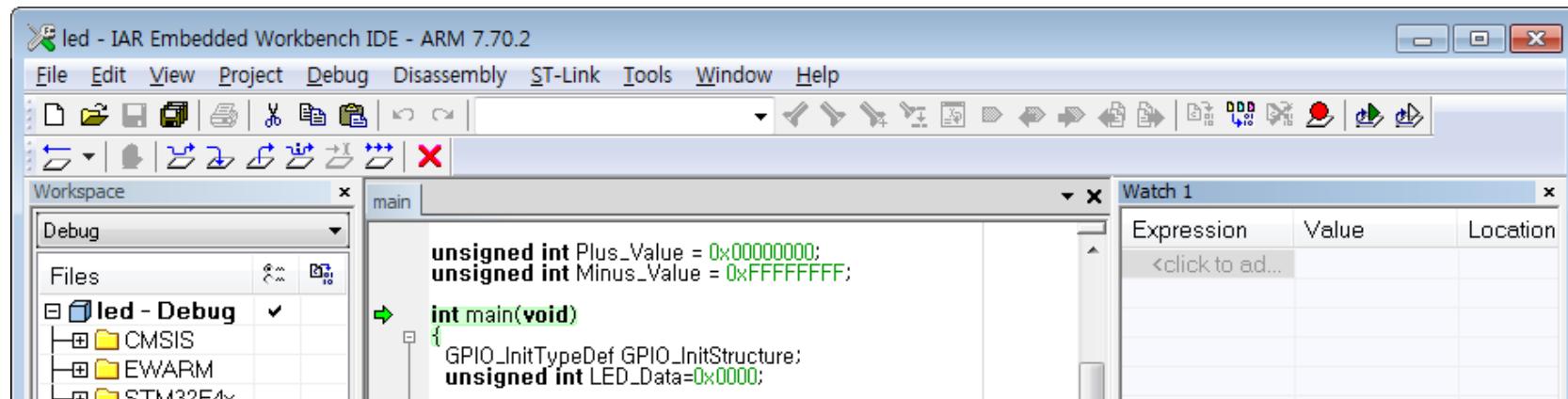
EWARM 디버깅 모드로 변수값 확인하기

- Watch 창
 - 변수 관찰 창을 열기

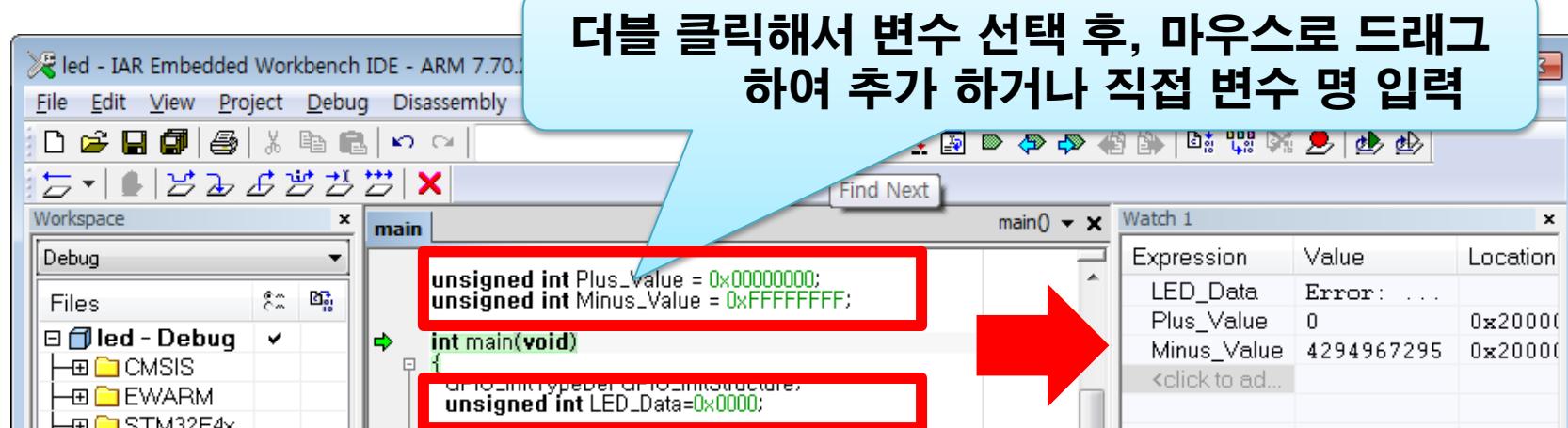


EWARM 디버깅 모드로 변수값 확인하기

- Watch 창
 - 변수 추가



더블 클릭해서 변수 선택 후, 마우스로 드래그
하여 추가하거나 직접 변수명 입력



EWARM 디버깅 모드로 변수값 확인하기

- Watch 창
 - 변수 값 확인

Expression	Value	Location	Type
LED_Data	Error: ...		
Plus_Value	0	0x20000004	unsigned int
Minus_Value	4294967295	0x20000000	unsigned int

EWARM 디버깅 모드로 변수값 확인하기

- Break Point(F9)
 - 앞에서 설정했던 Break Point 를 해제하고 새로운 Break Point를 설정
 - Plus_Value++ 명령 앞에다가 브레이크 포인트를 설정

The screenshot shows the EWARM IDE interface. On the left is the assembly code for the main function, and on the right is the Watch 1 window displaying variable values.

Watch 1 Window:

Expression	Value	Location	Type
LED_Data	Error: ...		
Plus_Value	0	0x20000004	unsigned int
Minus_Value	4294967295	0x20000000	unsigned int

Assembly Code (main function):

```
GPIO_InitTypeDef GPIO_InitStructure;
unsigned int LED_Data=0x0000;

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1;
GPIO_Init(GPIOC, &GPIO_InitStructure);

while(1)
{
    Plus_Value++;
    GPIO_Write(GPIOC, LED_Data);

    LED_Data++;
    if(LED_Data > 0xF) LED_Data = 0; //LED_Data가 16이상이면 0으로 초기화

    Delay(1000);
}
```

EWARM 디버깅 모드로 변수값 확인하기

- Run(F5) 실행
 - Run 실행 후 Break Point에서 멈춤

The screenshot shows the EWARM IDE interface. On the left, the code editor displays the main.c file with the following code:

```
main f0
GPIO_InitTypeDef GPIO_InitStructure;
unsigned int LED_Data=0x0000;

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOC, &GPIO_InitStructure);

while(1)
{
    Plus_Value++;
    LED_Data = LED_Data + 1;
    GPIO_Write(GPIOC, LED_Data);

    LED_Data++;
    if(LED_Data > 0x0F) LED_Data = 0; // LED_Data값이 0x0F 보다 큼.

    Delay(1000);
}
```

A blue callout bubble points from the text "Break Point에서 멈춤" to the line "Plus_Value++;" in the code editor, which is highlighted with a red rectangle. A red arrow also points from this line to the "Break Point에서 멈춤" text.

On the right, the "Watch 1" window shows the following variable information:

Expression	Value	Location	Type
LED_Data	0	R4	unsigned int
Plus_Value	0	0x20000004	unsigned int
Minus_Value	4294967295	0x20000000	unsigned int

EWARM 디버깅 모드로 변수값 확인하기

- 변수 포맷 변경
 - Hexadecimal format 으로 변경

The screenshot shows the EWARM IDE interface. On the left is the code editor with C code for initializing GPIO and performing a loop. In the center is the Watch 1 window displaying variable values. A context menu is open over the LED_Data row in the Watch 1 window. The menu items are: Remove, Remove All, Default Format, Binary Format, Octal Format, Decimal Format (which is checked), Hexadecimal Format (highlighted with a red box), Char Format, Show As, Save to File..., and Options... A blue callout bubble says "Right Click" pointing to the menu. A red callout bubble says "Click" pointing to the "Hexadecimal Format" option. A red arrow points from the "Hexadecimal Format" option in the menu down to the Watch 1 window, where the LED_Data value is now shown as 0x00000010.

Expression	Value	Location	Type
LED_Data	0		
Plus_Value	0		
	4294967295		

Expression	Value	Location	Type
LED_Data	0x00000010	R4	unsigned int
Plus_Value	0x00000000	0x20000004	unsigned int
Minus_Value	0xFFFFFFF	0x20000000	unsigned int
<click to add...			

EWARM 디버깅 모드로 변수값 확인하기

- Run(F5) 실행
 - Run 실행 후 변수 값 확인

The screenshot shows the EWARM IDE interface. On the left is the main code editor for the 'main' file, containing C code for initializing a GPIO pin and setting up a loop to toggle its value. On the right is the 'Watch 1' window, which displays the current values of variables: LED_Data (0x00000001), Plus_Value (0x00000001), and Minus_Value (0xFFFFFFF). A red box highlights these three variables. A blue callout bubble with the text 'Run 1회 실행 결과' (Result of 1st execution) points to the Watch 1 window.

```

main
GPIO_InitTypeDef GPIO_InitStructure;
unsigned int LED_Data=0x0000;

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
GPIO_Init(GPIOC, &GPIO_InitStructure);

while(1)
{
    Plus_Value++;
    Minus_Value--;
    GPIO_Write(GPIOC, LED_Data);

    LED_Data++;
    if(LED_Data > 0x0F) LED_Data = 0; //LED_Data값을 1씩 증가한다.
                                    //LED_Data값이 0x0F보다 크면 0으로 초기화된다.

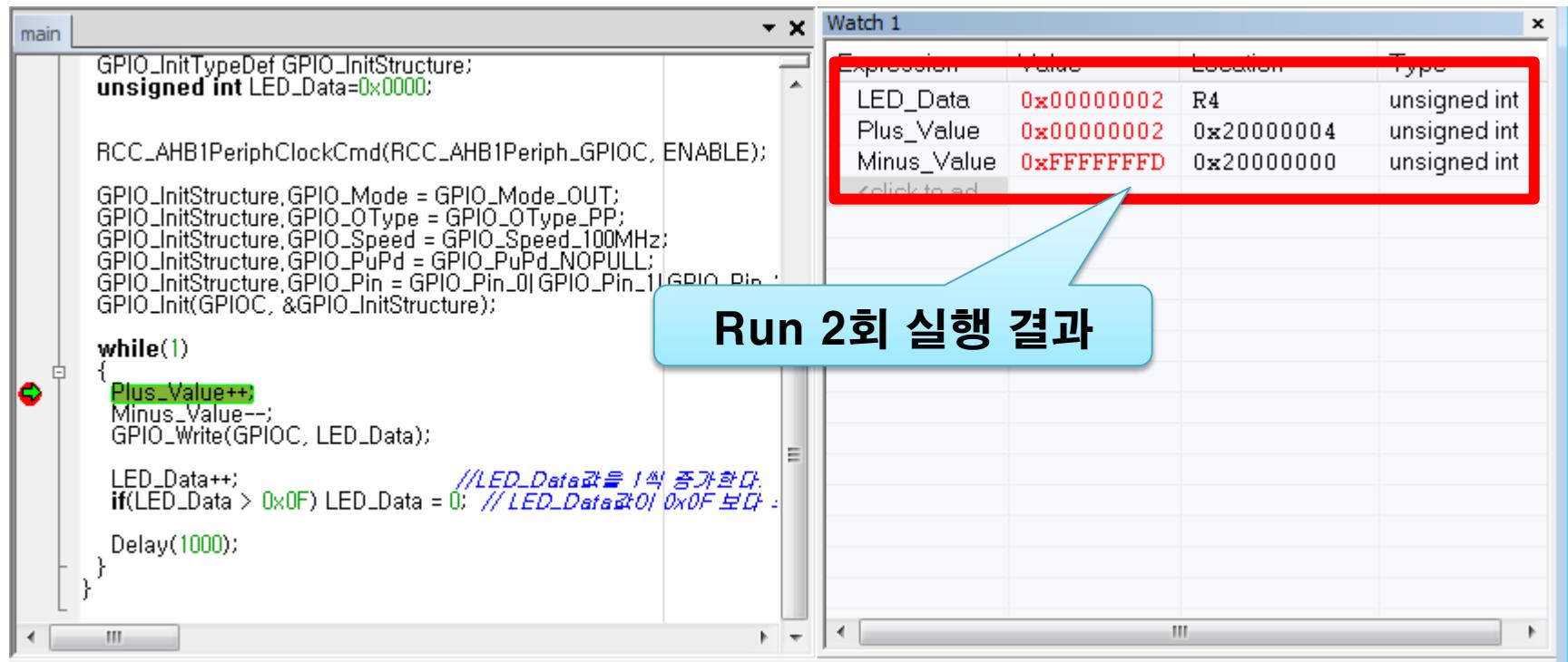
    Delay(1000);
}

```

Expression	Value	Location	Type
LED_Data	0x00000001	R4	unsigned int
Plus_Value	0x00000001	0x20000004	unsigned int
Minus_Value	0xFFFFFFF	0x20000000	unsigned int

EWARM 디버깅 모드로 변수값 확인하기

- Run(F5) 실행
 - 한 번더 Run 실행 후 변수 값 확인



Run 2회 실행 결과

Expression	Value	Location	Type
LED_Data	0x00000002	R4	unsigned int
Plus_Value	0x00000002	0x20000004	unsigned int
Minus_Value	0xFFFFFFF7	0x20000000	unsigned int

```
main
GPIO_InitTypeDef GPIO_InitStructure;
unsigned int LED_Data=0x0000;

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2;
GPIO_Init(GPIOC, &GPIO_InitStructure);

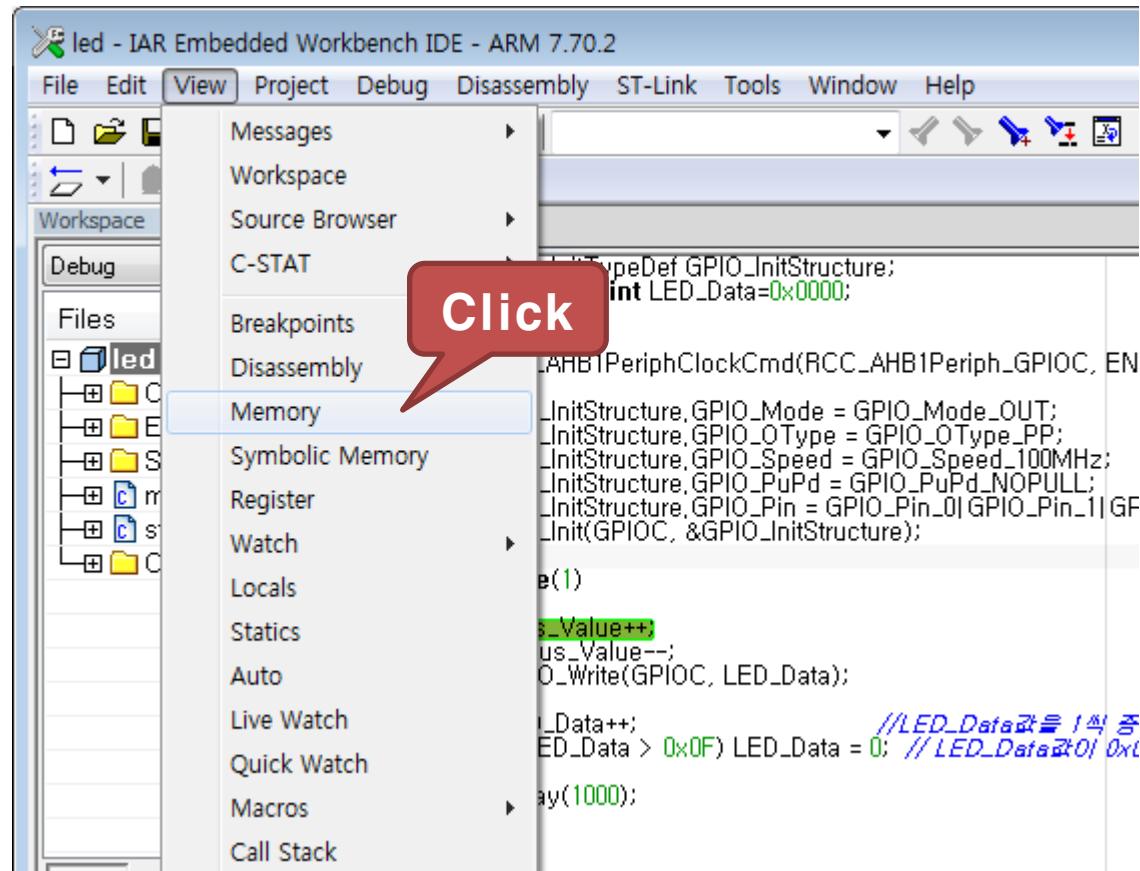
while(1)
{
    Plus_Value++;
    Minus_Value--;
    GPIO_Write(GPIOC, LED_Data);

    LED_Data++;
    if(LED_Data > 0x0F) LED_Data = 0; //LED_Data값을 1씩 증가한다.
    //LED_Data값이 0x0F보다 크면 그대로 증가시킨다.

    Delay(1000);
}
```

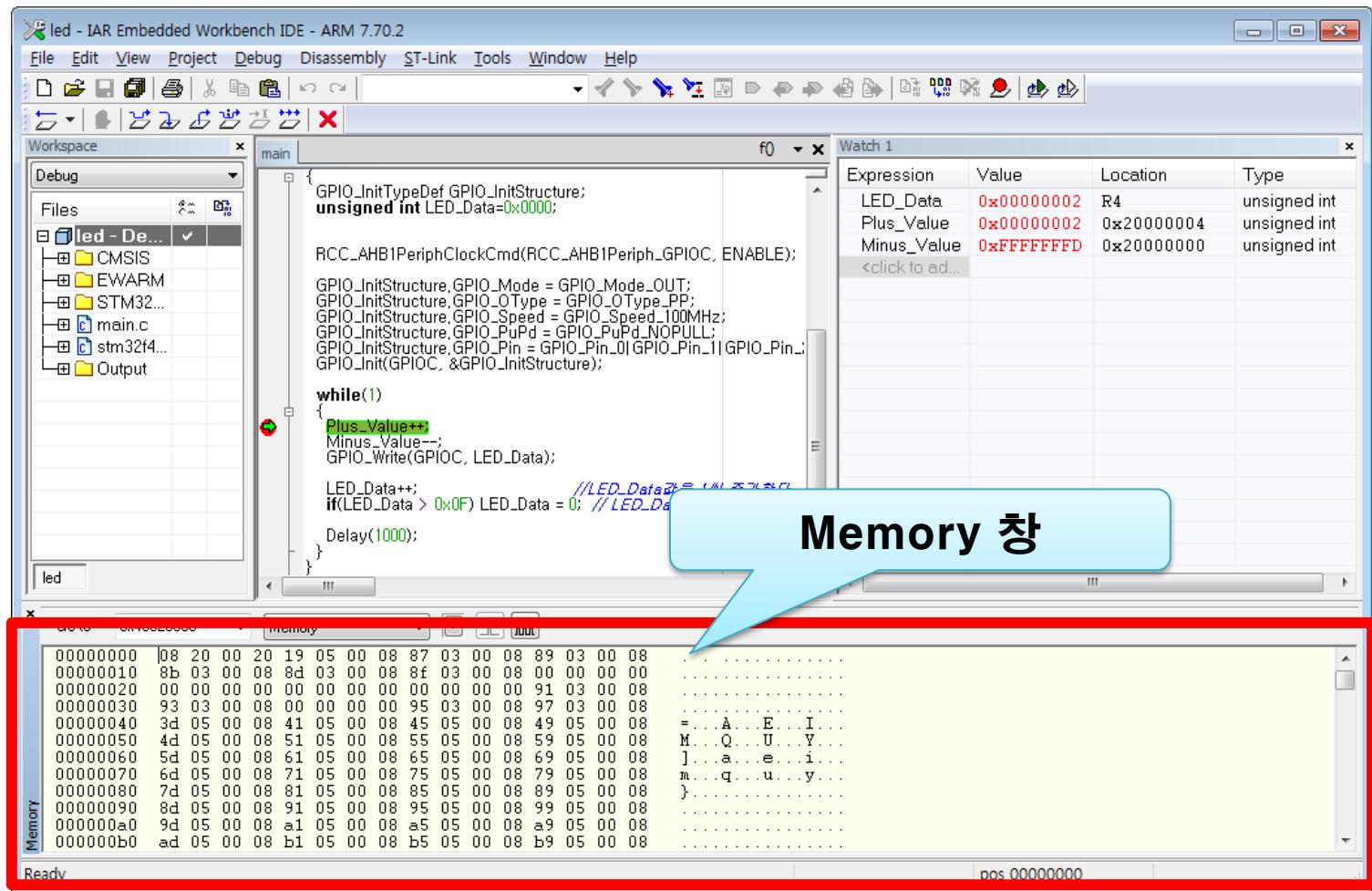
EWARM 디버깅 모드로 변수값 확인하기

- Memory 창
 - Memory 확인



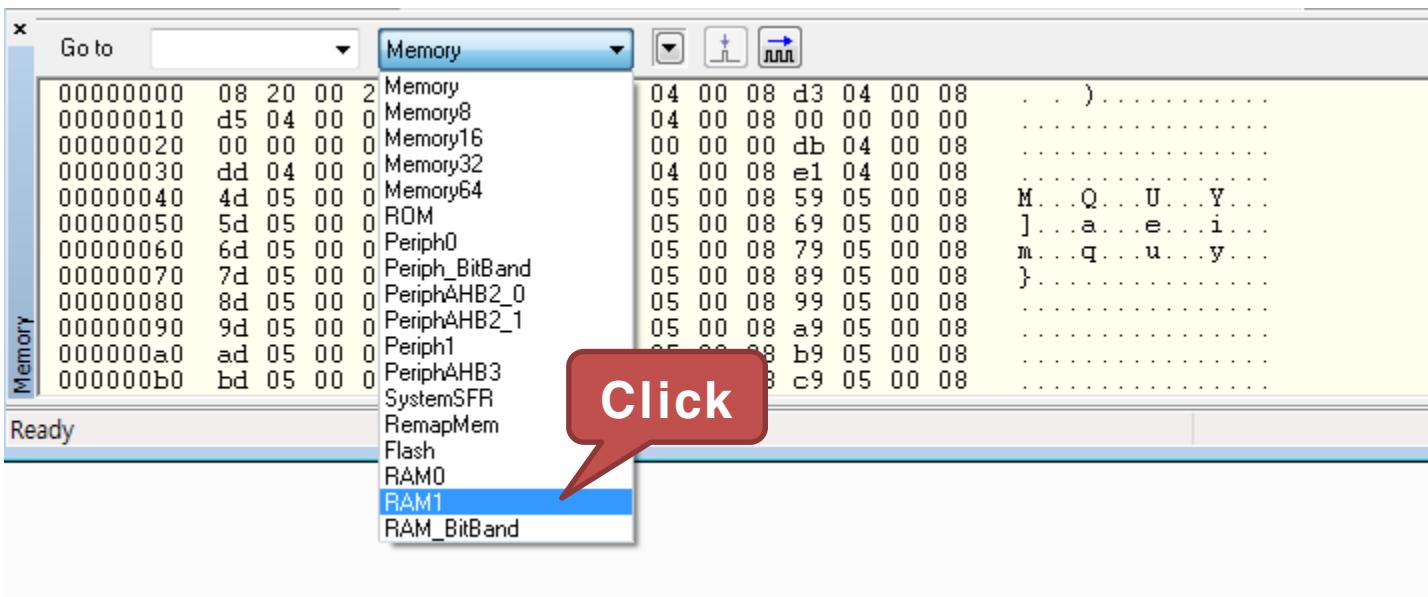
EWARM 디버깅 모드로 변수값 확인하기

- Memory 창
 - Memory 확인



EWARM 디버깅 모드로 변수값 확인하기

- Memory 창
 - RAM1 영역(0x20000000) 확인



EWARM 디버깅 모드로 변수값 확인하기

- Memory 창
 - Watch 창 정보

The screenshot shows the EWARM debugger interface with two windows:

- Watch 1 Window:** A table showing variable information:

Expression	Value	Location	Type
LED_Data	0x00000002	R4	unsigned int
Plus_Value	0x00000002	0x20000004	unsigned int
Minus_Value	0xFFFFFFF0	0x20000000	unsigned int
- Memory Window:** A hex dump of memory starting at address 20000000. A red box highlights the first 8 bytes (fd ff ff ff 02 00 00 00) which correspond to the value of Plus_Value in the Watch window. The memory dump continues with other data and ASCII representation.

인터럽트

- 폴링과 인터럽트 그리고 인터럽트 서비스루틴
- STM32F405 인터럽트
- 인터럽트로 LED 점멸시키기
- 인터럽트로 FND 점멸시키기



엣지아이랩

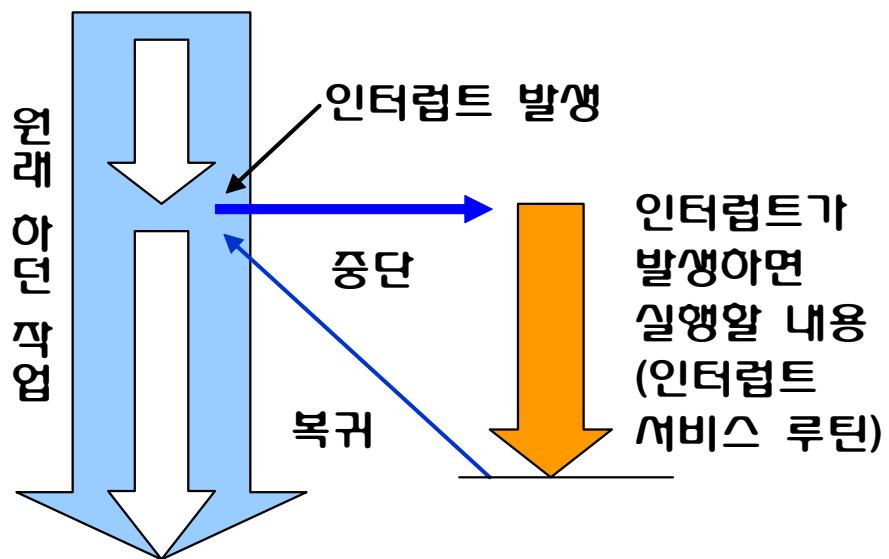
- 인터럽트(Interrupt)

- “방해하다”, “훼방놓다”
- 어떤 작업을 진행하고 있다가 갑자기 다른 일이 발생하여 먼저 처리해야 하는 상황을 인터럽트 발생이라 함
- 현재 수행중인 일을 잠시 중단하고 급한 일을 처리한 후 원래의 일을 다시 이어서 수행
- 이때, 그 급한 일을 해결하는 것이 인터럽트 서비스 루틴임
- 발생 시기를 예측할 수 없는 경우에 더 효율적

폴링과 인터럽트, 그리고 인터럽트 서비스 루틴

- 인터럽트와 인터럽트 서비스 루틴

- 인터럽트가 발생하면 프로세서는 현재 수행중인 프로그램을 멈추고
- 상태 레지스터와 PC(Program Counter)등을 스택에 잠시 저장한 후
인터럽트 서비스 루틴으로 점프
- 인터럽트 서비스 루틴을 실행한 후에는 이전의 프로그램으로 복귀하여 정상적인
절차를 실행



폴링과 인터럽트, 그리고 인터럽트 서비스 루틴

- 마이크로 컨트롤러의 외부 입력 방법
 - 폴링(polling)
 - 사용자가 명령어를 사용하여 입력 핀의 값을 계속 읽어서 변화를 알아내는 방식
 - 인터럽트(interrupt)
 - MCU 자체가 하드웨어적으로 그 변화를 체크하여 변화시에만 일정한 동작을 하는 방식
- 인터럽트의 구성요소
 - 발생원 : 누가 인터럽트를 요청했는가?
 - 우선순위 : 2개 이상의 요청시 누구를 먼저 서비스 할까?
(중요도 : Priority)
 - 인터럽트벡터 : 서비스루틴의 시작번지는 어디인가?

STM32F405 인터럽트

- STM32F405 인터럽트
 - 여러 개의 다른 인터럽트 발생원 제공
 - 모든 인터럽트는 개별적인 인터럽트 허용 비트를 할당 받음
 - 외부 인터럽트를 처리하는 포트가 고정되어 있지 않음
 - 각 외부 인터럽트 소스들을 허용하는 레지스터
 - 인터럽트 마스크 레지스터(EXTI_IMR)
 - 이벤트 마스크 레지스터(EXTI_EMR)
 - 소프트웨어 인터럽트 이벤트 레지스터(ETXI_SWIER)

STM32F405 인터럽트

- SYSCFG_EXTICRn (SYSCFG EXTernal Interrupt Configuration Register)

- 외부 인터럽트 설정 레지스터
- EXTIx[3:0] : EXTIx 설정 비트 ($y = 0 \sim 15$)

출력 포트의 타입을 설정하는 비트

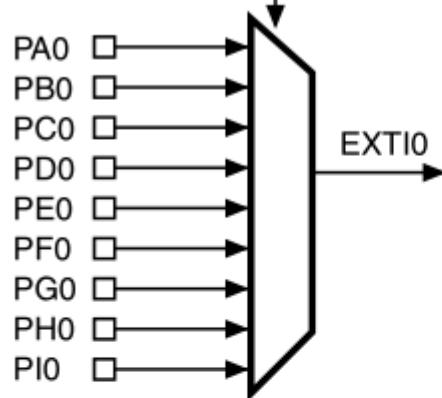
- 0000: PA[x] pin
- 0001: PB[x] pin
- 0010: PC[x] pin
- 0011: PD[x] pin
- 0100: PE[x] pin
- 0101: PF[x] pin
- 0110: PG[x] pin
- 0111: PH[x] pin
- 1000: PI[x] pin

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI3[3:0]				EXTI3[3:0]				EXTI3[3:0]			

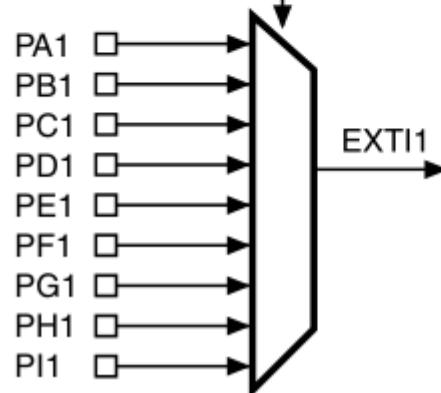
STM32F405 인터럽트

- 외부 인터럽트 GPIO 연결
 - SYSCFG_EXTICR1 register
 - SYSCFG_EXTICR2 register
 - SYSCFG_EXTICR3 register
 - SYSCFG_EXTICR4 register

EXTI0[3:0] bits in the SYSCFG_EXTICR1 register



EXTI1[3:0] bits in the SYSCFG_EXTICR1 register



STM32F405 인터럽트

- EXTI_IMR (Interrupt mask register)
 - 인터럽트 마스크 레지스터
 - MR[22:0] : '0'이면 인터럽트 요청 허용, '1'이면 인터럽트 요청 무시
 - MR[15:0] : 외부 인터럽트
 - MR16 : PVD(Programmable voltage detector) output
 - MR17 : RTC Alarm
 - MR18 : USB OTG FS Wakeup
 - MR19 : Ethernet Wakeup
 - MR20 : USB OTG HS Wakeup
 - MR21 : RTC Tamper and TimeStamp
 - MR22 : RTC Wakeup

31-23									22	21	20	19	18	17	16
Reserved									MR22	MR21	MR20	MR19	MR18	MR17	MR16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0

STM32F405 인터럽트

- EXTI_EMR (Event mask register)
 - 이벤트 마스크 레지스터
 - MR[22:0] : '0'이면 이벤트 요청 허용, '1'이면 이벤트 요청 무시
 - MR[15:0] : 외부 인터럽트
 - MR16 : PVD(Programmable voltage detector) output
 - MR17 : RTC Alarm
 - MR18 : USB OTG FS Wakeup
 - MR19 : Ethernet Wakeup
 - MR20 : USB OTG HS Wakeup
 - MR21 : RTC Tamper and TimeStamp
 - MR22 : RTC Wakeup

31-23										22	21	20	19	18	17	16
Reserved										MR22	MR21	MR20	MR19	MR18	MR17	MR16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0	

STM32F405 인터럽트

- 외부 인터럽트의 트리거
 - 인터럽트를 발생시키기 위한 “방아쇠”
 - 인터럽트 발생의 유무를 판단하는 근거가 됨
 - 트리거 방법
 - Edge Trigger : 입력 신호가 변경되는 순간을 트리거로 사용하는 경우
 - 하강에지(Falling Edge) 트리거 : '1'에서 '0'로 변경되는 시점을 사용
 - 상승에지(Rising Edge) 트리거 : '0'에서 '1'로 변경되는 시점을 사용
 - Level Trigger : 입력 신호가 일정 시간동안 원하는 레벨을 유지시 트리거로 사용하는 경우
 - 평상시 High(1)로 있다가 Low(0)로 변화되어 일정시간 유지되면 트리거 하게 됨

STM32F405 인터럽트

- EXTI_RTSR (Rising trigger selection register)
 - 상승 트리거 설정 레지스터
 - TR[22:0] : '0'이면 상승 트리거 enable, '1'이면 상승 트리거 disable
 - TR[15:0] : 외부 인터럽트
 - TR16 : PVD(Programmable voltage detector) output
 - TR17 : RTC Alarm
 - TR18 : USB OTG FS Wakeup
 - TR19 : Ethernet Wakeup
 - TR20 : USB OTG HS Wakeup
 - TR21 : RTC Tamper and TimeStamp
 - TR22 : RTC Wakeup

31-23									22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0

STM32F405 인터럽트

- EXTI_FTSR (Falling trigger selection register)
 - 하강 트리거 설정 레지스터
 - TR[22:0] : '0'이면 하강 트리거 enable, '1'이면 하강 트리거 disable
 - TR[15:0] : 외부 인터럽트
 - TR16 : PVD(Programmable voltage detector) output
 - TR17 : RTC Alarm
 - TR18 : USB OTG FS Wakeup
 - TR19 : Ethernet Wakeup
 - TR20 : USB OTG HS Wakeup
 - TR21 : RTC Tamper and TimeStamp
 - TR22 : RTC Wakeup

31-23									22	21	20	19	18	17	16
Reserved									TR22	TR21	TR20	TR19	TR18	TR17	TR16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0

STM32F405 인터럽트

- EXTI_SWIER (Software interrupt event register)
 - 소프트웨어 인터럽트 이벤트 레지스터
 - SWIER[22:0] : 소프트웨어로 인터럽트 이벤트를 발생시킬 때 사용
 - SWIER[15:0] : 외부 인터럽트
 - SWIER16 : PVD(Programmable voltage detector) output
 - SWIER17 : RTC Alarm
 - SWIER18 : USB OTG FS Wakeup
 - SWIER19 : Ethernet Wakeup
 - SWIER20 : USB OTG HS Wakeup
 - SWIER21 : RTC Tamper and TimeStamp
 - SWIER22 : RTC Wakeup

31-23									22	21	20	19	18	17	16
Reserved									SWIER 22	SWIER 21	SWIER 20	SWIER 19	SWIER 18	SWIER 17	SWIER 16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWIER 15	SWIER 14	SWIER 13	SWIER 12	SWIER 11	SWIER 10	SWIER 9	SWIER 8	SWIER 7	SWIER 6	SWIER 5	SWIER 4	SWIER 3	SWIER 2	SWIER 1	SWIER 0

STM32F405 인터럽트

- EXTI_PR (Pending register)

- 펜딩 레지스터

- PR[15:0] : '0'이면 트리거 요청 무시, '1'이면 트리거 요청 발생
 - PR16 : PVD(Programmable voltage detector) output
 - PR17 : RTC Alarm
 - PR18 : USB OTG FS Wakeup
 - PR19 : Ethernet Wakeup
 - PR20 : USB OTG HS Wakeup
 - PR21 : RTC Tamper and TimeStamp
 - PR22 : RTC Wakeup

31-23									22	21	20	19	18	17	16
Reserved									PR22	PR21	PR20	PR19	PR18	PR17	PR16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0

STM32F405 인터럽트

- STM32F405 인터럽트 프로그램
 - main.c

```
int main(){
    .....
    /*인터럽트 설정*/
    EXTI_Init(xxxx);

    /*NVIC 설정*/
    NVIC_Init(xxxx);
    .....
}

void xxx_IRQHandler(void)
{
    /* 인터럽트가 발생되었을 때 실행할 명령어 세트를 선언 */
}
```

실습 1: 인터럽트로 LED 점멸

● 실습 개요

- STM32F405 마이크로컨트롤러의 인터럽트 기능을 이용하여 LED를 점멸시키기
- 일정시간마다 LED가 순차적으로 켜지도록 하고, 버튼 스위치를 누르면 LED가 멈추었다가 다시 누르면 동작하도록 함
- 버튼 스위치가 눌려지면 인터럽트가 발생하도록 해야 함
- 입력포트 1개(인터럽트가 가능한 포트), 출력포트 1개 사용

● 실습 목표

- 인터럽트 발생 원리 이해
- 인터럽트 제어 방법의 습득(관련 레지스터 이해)
- 입출력 포트에 관한 이해(특히 인터럽트 관련 포트)

실습 1: 인터럽트로 LED 점멸

- 구동 프로그램 : 사전 지식
 - 인터럽트를 위한 입력 포트 선택
 - 포트 B의 12번 비트는 EXTI12로 사용할 수 있는 포트임
 - SYSCFG_EXTICR4 레지스터의 [3:0] 비트를 0001로 세팅
 - 외부 인터럽트 Enable
 - EXTI_IMR 레지스터의 12번 비트를 1로 세팅
 - 인터럽트 트리거 방법 정의
 - 상승 에지에서 트리거하도록 EXTI_RTSR 레지스터의 12번 비트를 1로 세팅
 - Main 루틴 기술
 - 인터럽트 서비스 루틴의 선언
 - void xxx_IRQHandler(void); 방식을 사용
 - EXTI12 → void EXTI15_10_IRQHandler(void);

실습 1: 인터럽트로 LED 점멸

- EXTI 라이브러리
 - STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
 - EXTI_InitTypeDef 구조체

EXTI_InitTypeDef구조체	설 명
uint32_t EXTI_Line	인터럽트 입력을 선택한다. EXTI_Linex (x = 0 ~ 22)
EXTIMode_TypeDef EXTI_Mode	인터럽트인지 이벤트인지 설정한다. (EXTI_Mode_Interrupt, EXTI_Mode_Event)
EXTITrigger_TypeDef EXTI_Trigger	트리거를 설정한다. EXTI_Trigger_Rising 상승엣지 EXTI_Trigger_Falling 하강 엣지 EXTI_Trigger_Rising_Falling 엣지 변화시
FunctionalState EXTI_LineCmd	인터럽트를 Enable/Disable한다.

실습 1: 인터럽트로 LED 점멸

- NVIC 라이브러리

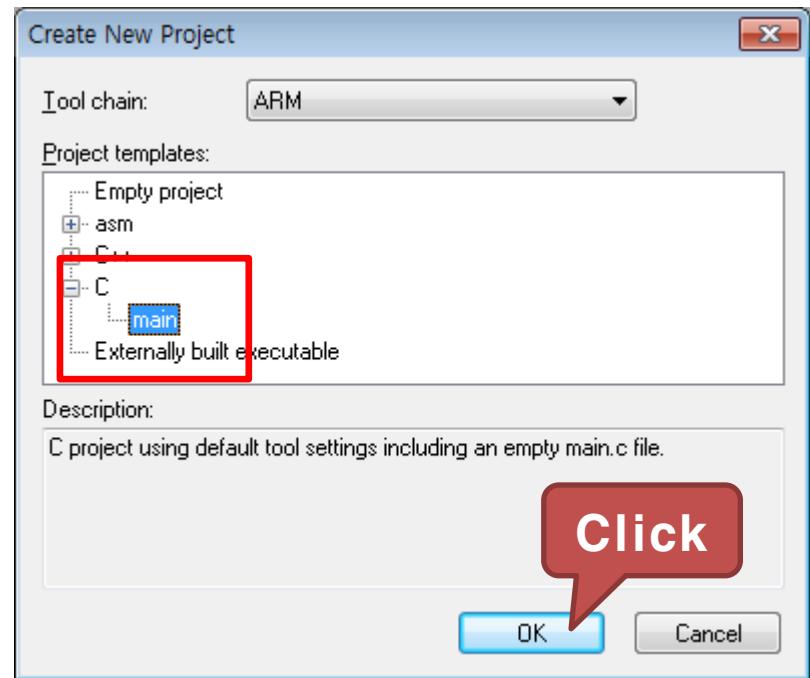
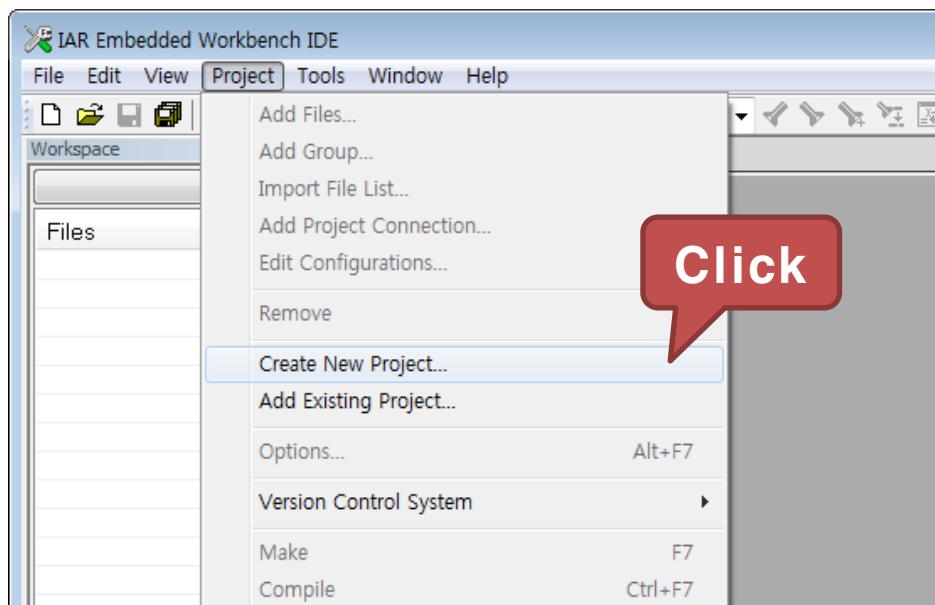
- STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
- NVIC_InitTypeDef 구조체

NVIC_InitTypeDef구조체	설명
uint8_t NVIC_IRQChannel	인터럽트 IRQ를 선택한다. (misch 참조)
uint8_t NVIC_IRQChannelPreemptionPriority	최우선순위를 입력, 이 값이 작을수록 높은 우선순위를 갖는다.
uint8_t NVIC_IRQChannelSubPriority	보조우선순위를 입력, 이 같은 서로 같은 우선순위를 가진 IRQ들의 순위를 결정한다.
FunctionalState NVIC_IRQChannelCmd	인터럽트를 Enable/Disable한다.(Core레벨)

실습 1: 인터럽트로 LED 점멸

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch5” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 이 폴더의 이름을 “interruptLed”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1: 인터럽트로 LED 점멸

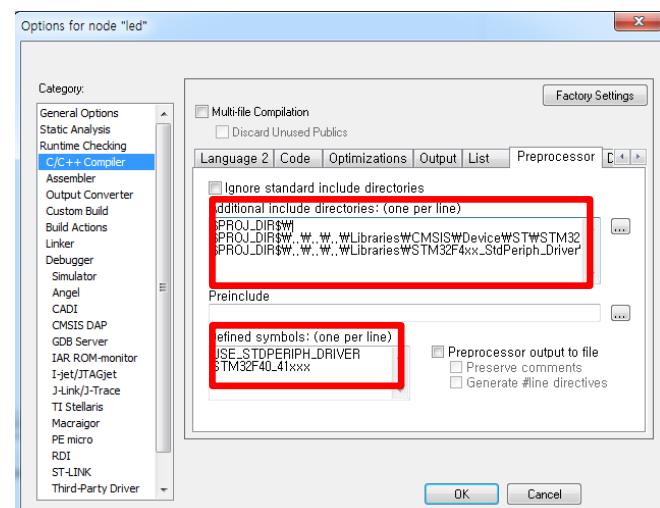
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch5\interruptLed	system_stm32f4xx.c
Cortex_Example\Projects\ch5\interruptLed	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_exti.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_syscfg.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_misc.c

실습 1: 인터럽트로 LED 점멸

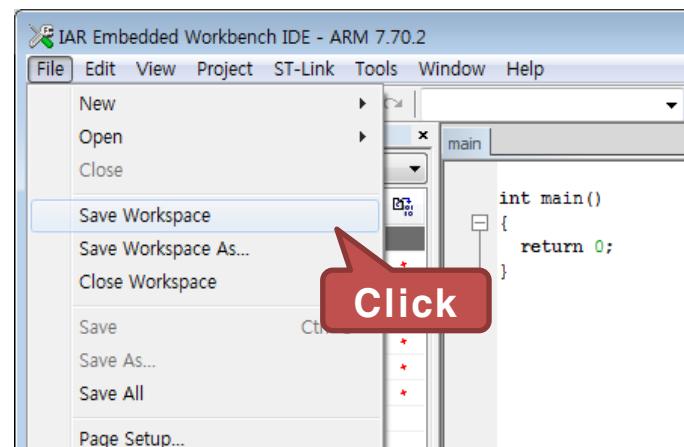
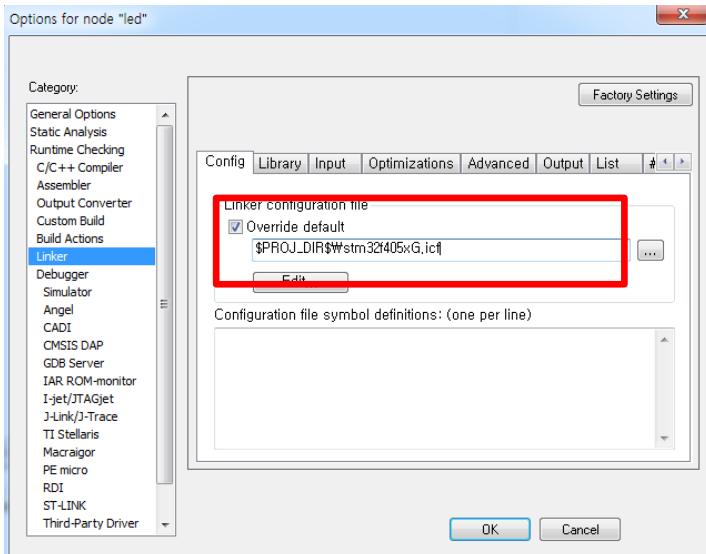
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1: 인터럽트로 LED 점멸

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1: 인터럽트로 LED 점멸

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

unsigned char Time_STOP = 0;

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    unsigned int LED_Data=0x0010;
    // GPIOB, GPIOC, SYSCFG 클럭 인가
    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOC, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
}
```

실습 1: 인터럽트로 LED 점멸

- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;  
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  
GPIO_InitStructure.GPIO_Pin =  
    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;  
GPIO_Init(GPIOC, &GPIO_InitStructure);  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;  
// B 포트의 12번 비트를 입력으로 선언  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;  
GPIO_Init(GPIOB, &GPIO_InitStructure);  
  
// 포트B의 12번핀을 외부 인터럽트 12와 연결  
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource12);
```

실습 1: 인터럽트로 LED 점멸

- 구동 프로그램
 - main.c 코드 작성

```
//...  
// 외부 인터럽트12가 상승엣지에서 발생되게 설정하고, 허용  
EXTI_InitStructure.EXTI_Line = EXTI_Line12;  
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;  
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;  
EXTI_InitStructure.EXTI_LineCmd = ENABLE;  
EXTI_Init(&EXTI_InitStructure);  
  
// 외부 인터럽트12 인터럽트의 우선순위를 설정  
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);  
NVIC_InitStructure.NVIC IRQChannel = EXTI15_10_IRQn;  
NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 0x01;  
NVIC_InitStructure.NVIC IRQChannelSubPriority = 0x01;  
NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;  
NVIC_Init(&NVIC_InitStructure);
```

실습 1: 인터럽트로 LED 점멸

- 구동 프로그램

- main.c 코드 작성

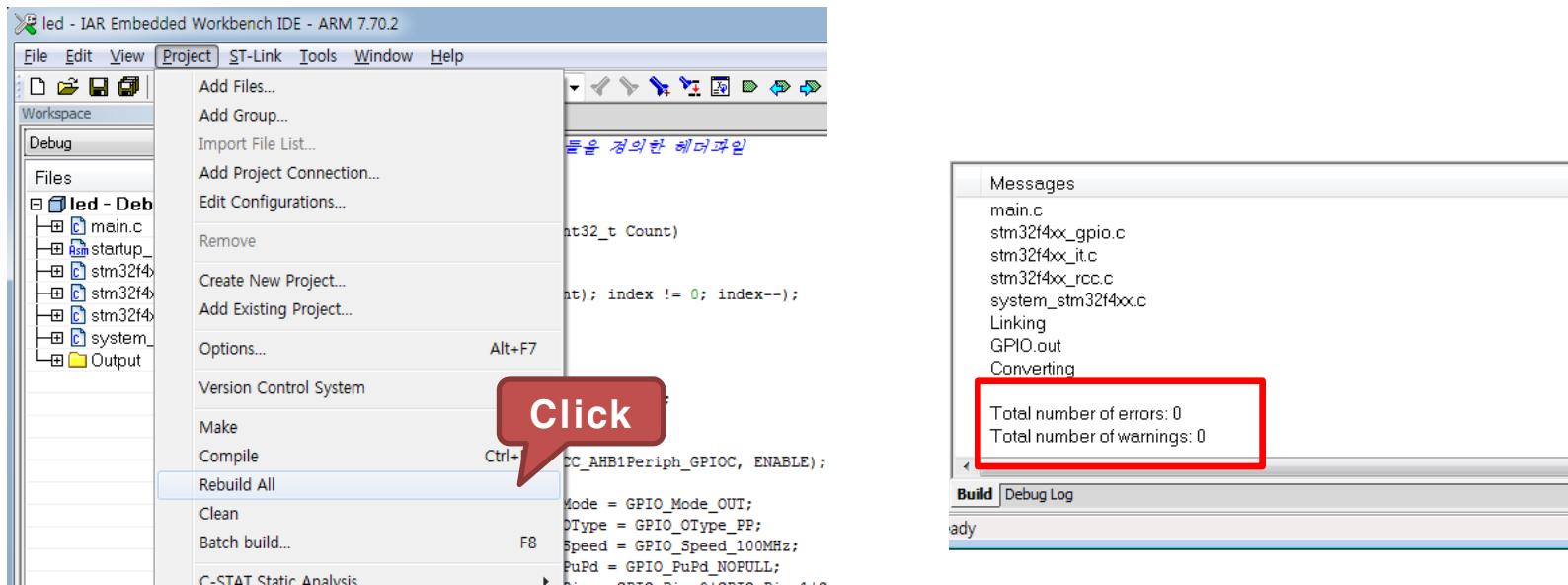
```
while(1) {
    GPIO_ResetBits(GPIOC, LED_Data);
    if(Time_STOP == 0) { // Time_Stop이 0인 경우
        if(LED_Data == 0x0010) LED_Data = 0x0001;
        else LED_Data <<= 1;
    }
    GPIO_SetBits(GPIOC, LED_Data);
    Delay(100); }
}

void EXTI15_10_IRQHandler(void) {
    if(EXTI_GetITStatus(EXTI_Line12) != RESET) {
        if(Time_STOP == 0) Time_STOP = 1; // Time_Stop에 1을 입력
        else Time_STOP = 0; // Time_Stop에 0을 입력
        EXTI_ClearITPendingBit(EXTI_Line12);
    }
}
```

실습 1: 인터럽트로 LED 점멸

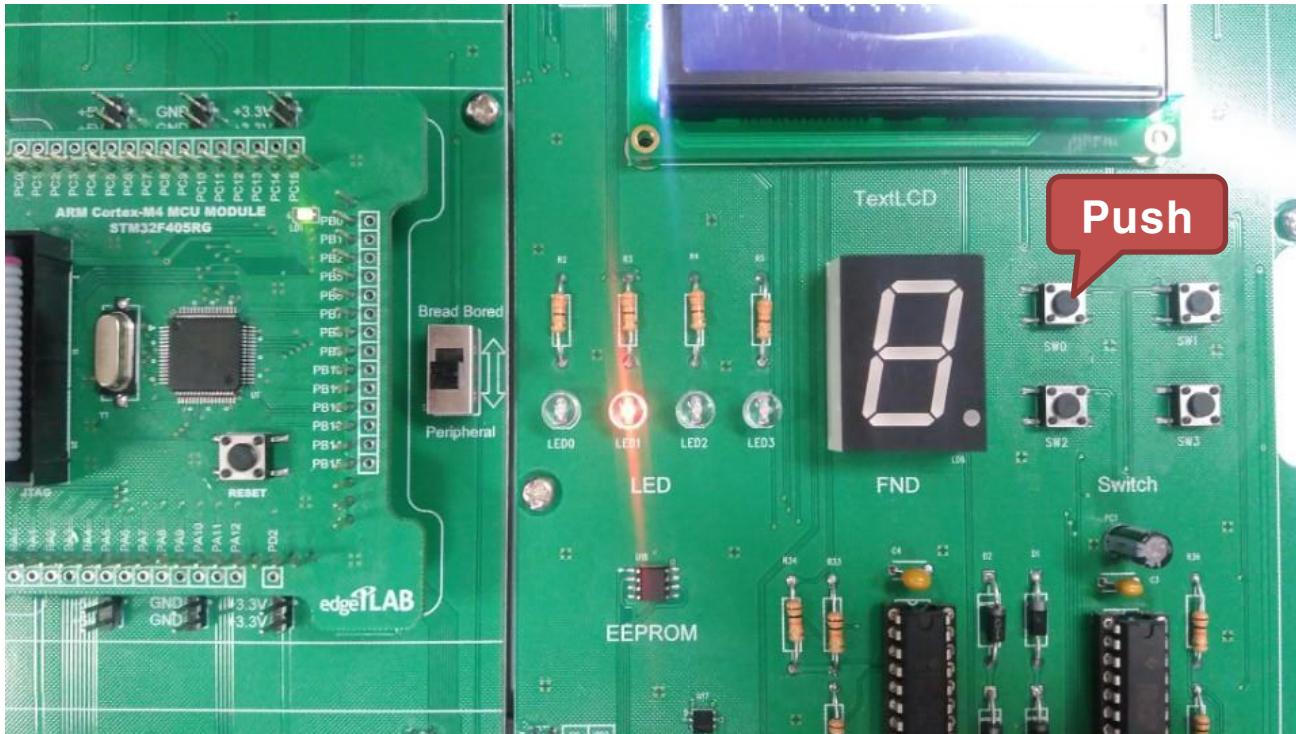
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 interruptLed 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 1: 인터럽트로 LED 점멸

- 실행 결과
 - LED의 불이 100ms마다 우측으로 쉬프트 되면서 켜짐
 - 0번 버튼을 누르면 LED가 멈추었다가 한번 더 누르면 다시 동작



실습 2: 인터럽트로 FND 점멸

- 실습 개요
 - 스위치 모듈과 FND 모듈에 연결하여 FND 점멸 실습
 - 일정 시간마다 클럭에 의해 FND에 숫자와 문자가 디스플레이 되도록 하고, 스위치를 누르면 FND 디스플레이가 초기화되도록 하며, 또한 다른 버튼을 누르면 잠시 멈추었다가 다시 이어서 동작을 하도록 함
 - 포트 B의 13번 핀을 Int13의 인터럽트로 사용
 - Int13에 의해서 FND 점멸 동작의 Stop/Start 기능을 구현
- 실습 목표
 - 인터럽트 활용 방법의 습득(관련 레지스터 이해)
 - Array FND 동작 원리 이해

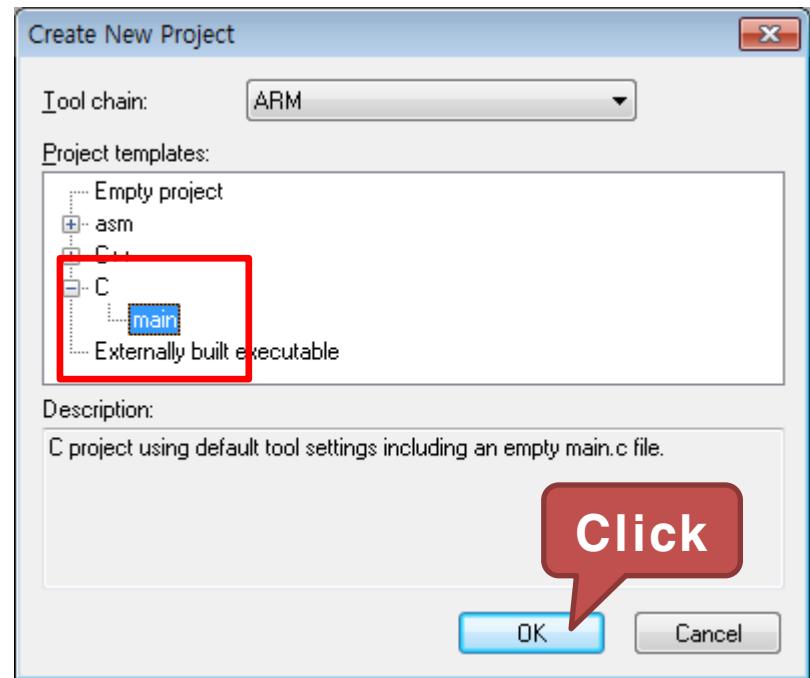
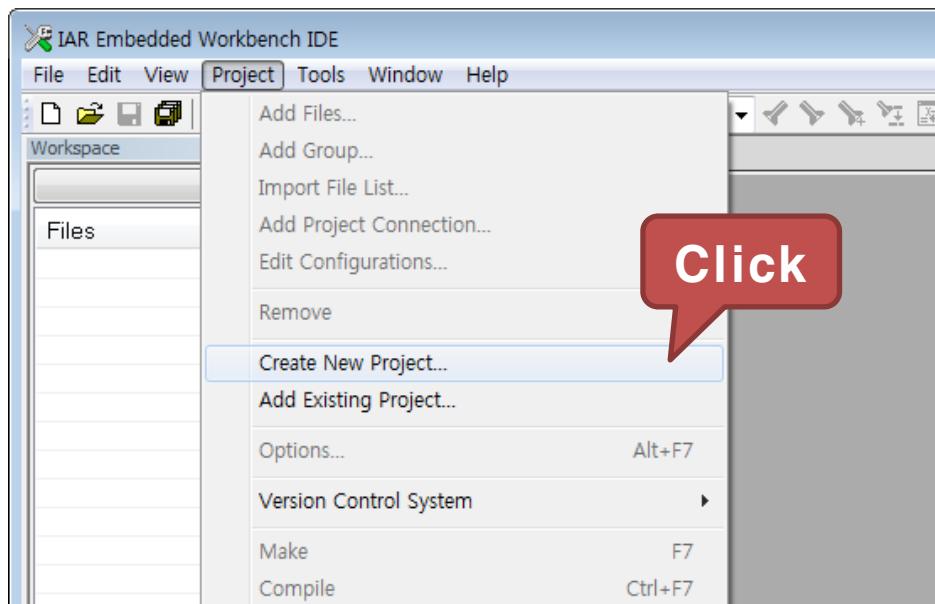
실습 2: 인터럽트로 FND 점멸

- 구동 프로그램 : 사전 지식
 - 인터럽트를 위한 입력 포트 선택
 - 포트 B의 13번 비트는 EXTI13으로 사용할 수 있는 포트
 - SYSCFG_EXTICR4 레지스터의 [7:4] 비트를 0001로 세팅
 - 외부 인터럽트 Enable
 - EXTI_IMR 레지스터의 13번 비트를 1로 세팅
 - 인터럽트 트리거 방법 정의
 - 상승 에지에서 트리거하도록 EXTI_RTSR 레지스터의 13번 비트를 1로 세팅
 - Main 루틴 기술
 - 인터럽트 서비스 루틴의 선언
 - void xxx_IRQHandler(void); 방식을 사용
 - EXTI12 → void EXTI15_10_IRQHandler(void);

실습 2: 인터럽트로 FND 점멸

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch5” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 이 폴더의 이름을 “interruptFnd”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2: 인터럽트로 FND 점멸

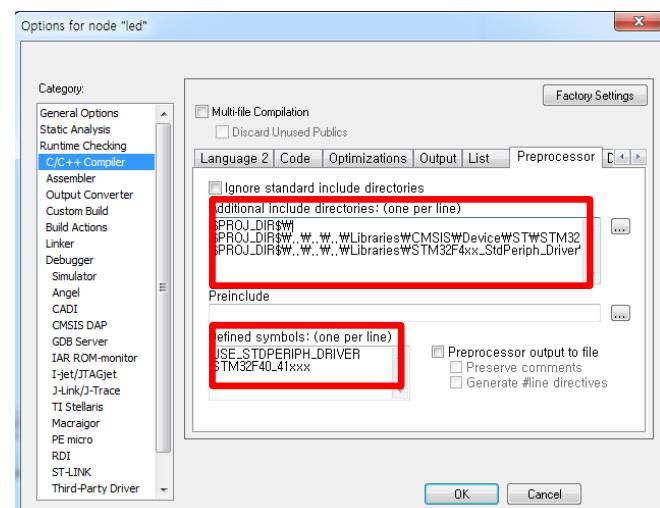
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch5\interruptFnd	system_stm32f4xx.c
Cortex_Example\Projects\ch5\interruptFnd	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_exti.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_syscfg.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_misc.c

실습 2: 인터럽트로 FND 점멸

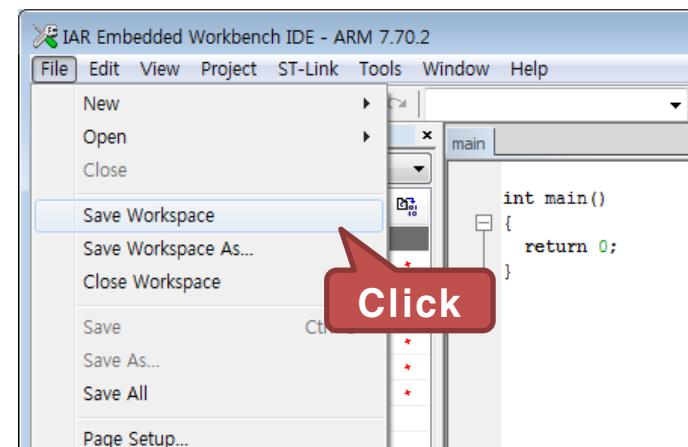
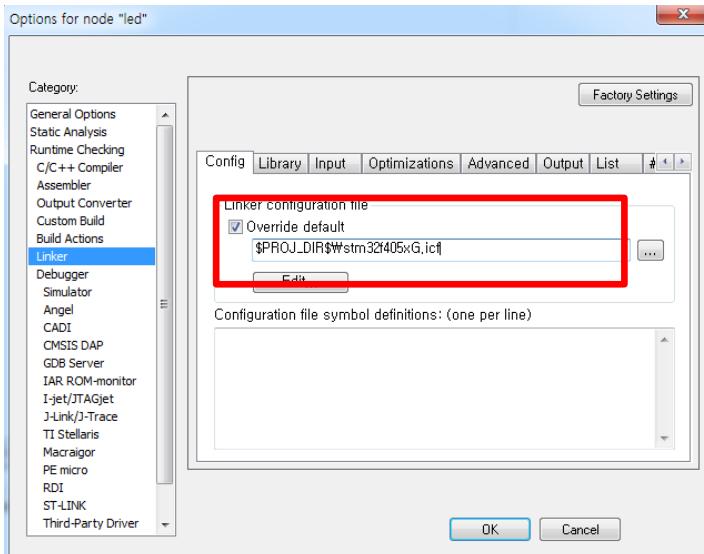
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2: 인터럽트로 FND 점멸

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2: 인터럽트로 FND 점멸

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

unsigned char Time_STOP = 0;

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    EXTI_InitTypeDef    EXTI_InitStructure;
    NVIC_InitTypeDef    NVIC_InitStructure;
    // 7-Segment에 표시할 글자의 입력 데이터를 저장
    unsigned int FND_DATA_TBL [] =
        {0x3F00, 0X0600, 0X5B00, 0X4F00, 0X6600, 0X6D00,
         0X7C00, 0X0700, 0X7F00, 0X6700, 0X7700, 0X7C00,
         0X3900, 0X5E00, 0X7900, 0X7100, 0X0800, 0X8000};
```

실습 2: 인터럽트로 FND 점멸

- 구동 프로그램

- main.c 코드 작성

```
unsigned char cnt=0;      // FND Table 카운터 변수
RCC_AHB1PeriphClockCmd(
    RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOC,  ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG,  ENABLE);
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11|
    GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
GPIO_Init(GPIOC, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
// B 포트의 13번 비트를 입력으로 선언
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

실습 2: 인터럽트로 FND 점멸

- 구동 프로그램

- main.c 코드 작성

```
// 포트B의 13번핀을 외부 인터럽트 13과 연결  
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource13);  
// 외부 인터럽트13이 상승엣지에서 발생되게 설정하고, 허용  
EXTI_InitStructure.EXTI_Line = EXTI_Line13;  
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;  
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;  
EXTI_InitStructure.EXTI_LineCmd = ENABLE;  
EXTI_Init(&EXTI_InitStructure);  
  
// 외부 인터럽트13 인터럽트의 우선순위를 설정  
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);  
NVIC_InitStructure.NVIC IRQChannel = EXTI15_10_IRQn;  
NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 0x01;  
NVIC_InitStructure.NVIC IRQChannelSubPriority = 0x01;  
NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;  
NVIC_Init(&NVIC_InitStructure);
```

실습 2: 인터럽트로 FND 점멸

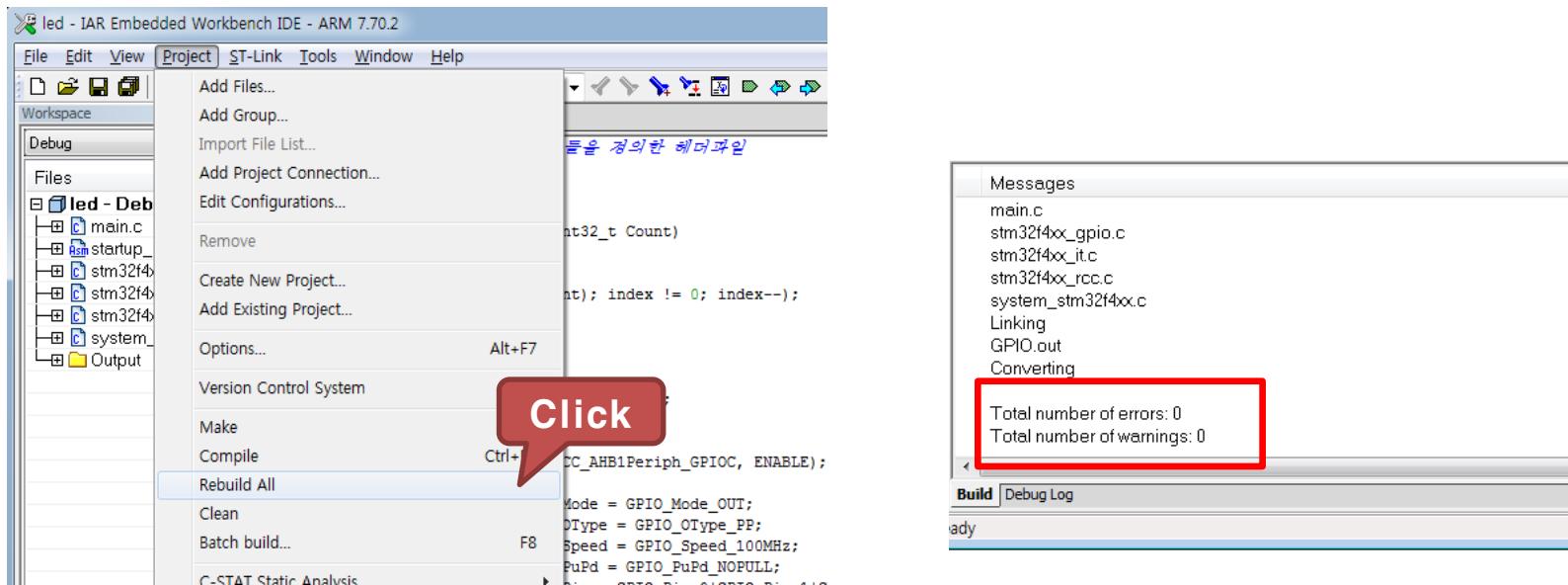
- 구동 프로그램

- main.c 코드 작성

```
while(1) {
    GPIO_Write(GPIOC, FND_DATA_TBL[cnt]); // 포트 C에 FND값 출력
    if(Time_STOP == 0) { // Time_Stop이 0인 경우
        cnt++;
        // FND Table 카운터 변수를 증가
        if(cnt>17) cnt=0; // 테이블 크기를 초과하는 경우 방지
    }
    Delay(200);
}
void EXTI15_10_IRQHandler(void) {
    if(EXTI_GetITStatus(EXTI_Line13) != RESET) {
        if(Time_STOP == 0) Time_STOP = 1; // Time_Stop에 1을 입력
        else Time_STOP = 0; // Time_Stop에 0을 입력
        EXTI_ClearITPendingBit(EXTI_Line13);
    }
}
```

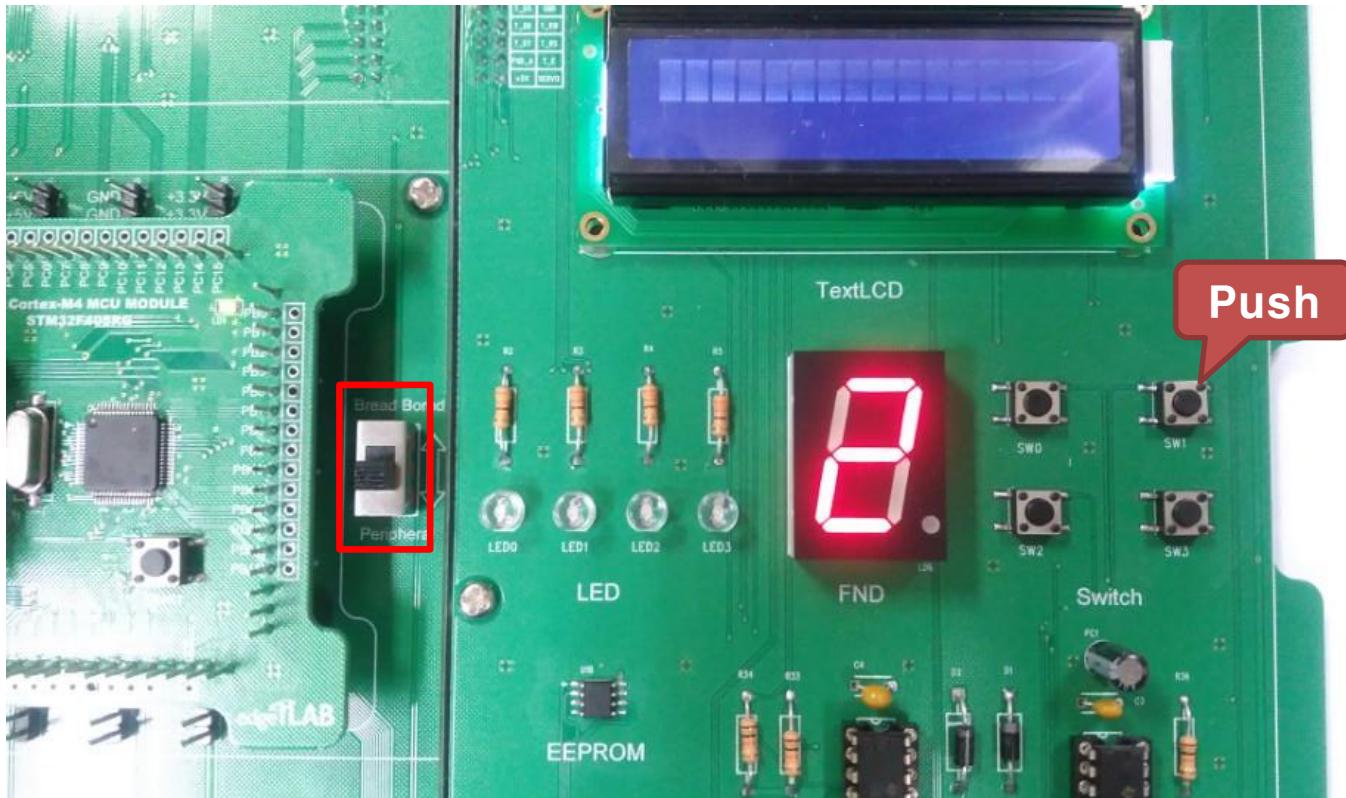
실습 2: 인터럽트로 FND 점멸

- 프로젝트 빌드 / 다운로드
 - 코드 작성이 완료되면 interruptFnd 프로젝트 빌드
 - “Project” 탭에서 “Rebuild All” 클릭
 - 오류가 발생하지 않았다면, 프로젝트 다운로드
 - “Project” 탭에서 “Download”의 “Download active application”을 클릭하여 빌드 한 프로젝트를 다운로드 함
 - 다운로드 완료하면 MCU 모듈의 “Reset” 버튼 클릭



실습 2: 인터럽트로 FND 점멸

- 실행 결과
 - FND의 숫자가 200ms마다 증가하면서 켜짐
 - 1번 버튼을 누르면 FND가 멈추었다가 한번 더 누르면 다시 동작





타이머와 카운터

- 클럭과 카운터
- STM32F405의 타이머/카운터
- 8비트 타이머/카운터의 일반 동작모드
- 타이머로 LED 점멸시키기
- 타이머로 디지털 시계 만들기



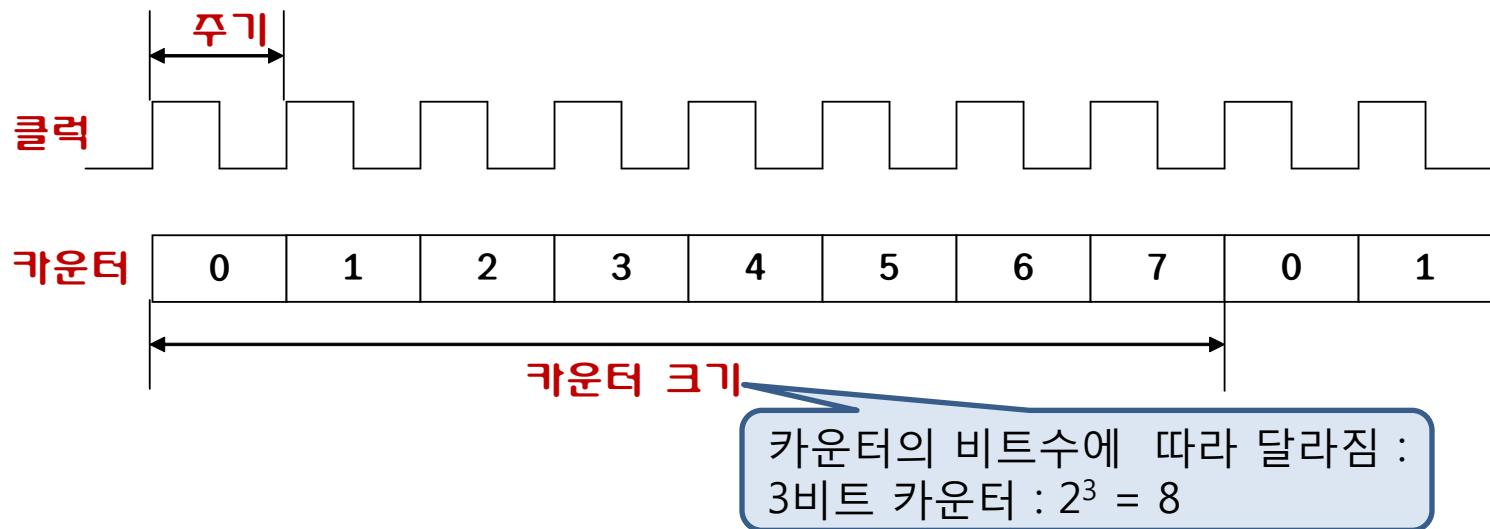
엣지아이랩

타이머/카운터

- 타이머와 카운터
 - 정확한 시간의 측정이 필요 (자명종과 스톱워치)
 - 임베디드 시스템에서 타이머와 카운터가 시간측정의 일을 담당
 - 타이머/카운터는 일정한 개수만큼의 클럭을 세어 시간을 측정하므로,
정확한 시간 재기가 가능
 - 타이머는 필요한 시간을 미리 레지스터에 설정하고, 다른 작업과 병행하게
타이머가 동작하고, 설정한 조건에서 인터럽트 발생하게 함으로써,
MCU의 효율을 극대화

클럭과 카운터

- 클럭
 - 시계
 - 일정한 시간 간격으로 0과 1의 값이 번갈아 나타남
 - 주어진 일을 순서대로 정확한 시간에 처리하기 위해 사용
- 카운터
 - 클럭을 세는 장치



STM32F405 타이머/카운터

- 타이머와 카운터

- 타이머 : MCU 내부 클럭을 세는 장치
 - 동기모드
 - 타이머는 MCU의 내부클럭을 세어 일정시간 간격의 펄스를 만들어 내거나 일정시간 경과 후에 인터럽트를 발생
 - 카운터 : MCU의 외부에서 입력되는 클럭을 세는 장치
 - 비동기모드
 - 카운터는 외부 핀(TOSC1, TOSC2, T1, T2, T3)을 통해서 들어오는 펄스를 계수(Edge Detector)하여 Event Counter로서 동작

STM32F405 타이머/카운터

- STM32F405의 타이머 카운터
 - Advanced-control timers
 - 타이머 1, 8
 - 향상된 기능의 16비트 타이머로 서로 기능 유사
 - 타이머 2 ~ 5
 - 다양한 기능의 16/32비트 타이머로 서로 기능 유사
 - 3과 4는 16비트, 2와 5는 32 비트
 - 타이머 9 ~ 14
 - 다양한 기능의 16비트 타이머로 서로 기능 유사
 - Basic timers
 - 타이머 6, 7
 - 기본 기능의 타이머로 서로 기능 유사
 - 2개의 워치독 타이머
 - 1개의 systick timer (24bit down counter)

STM32F405 타이머/카운터

- STM32F405의 타이머 카운터
 - 인터럽트 기능
 - 업데이트 인터럽트
 - 카운터 오버/언더플로우에 의해 발생
 - 출력비교 인터럽트
 - 카운터 값이 출력비교 레지스터의 값과 같게 되는 순간에 발생
 - 입력 캡쳐 인터럽트
 - 외부로부터의 트리거 신호에 의해서 카운터의 초기값을 입력캡쳐
 - PWM 출력 기능
 - Pulse Width Modulation

Advanced-control timers

- Advanced-control timers의 특징
 - 1, 8번 타이머
 - 16비트 업/다운/오토리로드(Up/Down/Auto-reload)카운터
 - 16비트 카운터 : $2^{16} = 65536$, 즉 0~65535까지 셀 수 있음
 - 16비트의 프리스케일러(prescaler) 보유
 - 기능
 - 입력 캡쳐(Input capture)
 - 출력 비교(Output compare)
 - PWM 기능(Edge and Center-aligned Mode)
 - 단일 펄스 모드 출력(One-pulse mode output)

Advanced-control timers

- Advanced-control timers의 특징
 - 각종 인터럽트 기능
 - 업데이트(Update)
 - 트리거 이벤트(Trigger event)
 - 입력 캡쳐(Input capture)
 - 출력 비교(Output compare)
 - Break input
 - 클럭 소스
 - 내부클럭
 - Tix 외부클럭
 - TIMx_ETR핀 외부클럭
 - 다른 타이머를 클럭으로 설정

General-purpose timers

- General-purpose timers의 특징
 - 2 ~ 5번, 9 ~ 14번 타이머/카운터
 - 이 중, 2와 5번은 32비트 타이머
 - 16/32비트 업/다운/오토리로드(Up/Down/Auto-reload)카운터
 - 16비트 → $2^{16} = 65536$, 즉 0~65535까지 셀 수 있음
 - 32비트 → $2^{32} = 4294967296$, 즉 0~4294967296까지 셀 수 있음
 - 16비트의 프리스케일러(prescaler) 보유
 - 기능
 - 입력 캡쳐(Input capture)
 - 출력 비교(Output compare)
 - PWM 기능(Edge and Center-aligned Mode)
 - 단일 펄스 모드 출력(One-pulse mode output)

General-purpose timers

- General-purpose timers의 특징
 - 각종 인터럽트 기능
 - 업데이트(Update)
 - 트리거 이벤트(Trigger event)
 - 입력 캡쳐(Input capture)
 - 출력 비교(Output compare)
 - 클럭 소스
 - 내부클럭
 - Tix 외부클럭(타이머 2~5, 9, 12 만 가능)
 - TIMx_ETR핀 외부클럭(타이머 2, 3, 4 만 가능)
 - 다른 타이머를 클럭으로 설정
(타이머 2~5, 9, 12 만 가능)

Basic timers

- Basic timers의 특징
 - 6, 7번 타이머
 - 16비트 오토리로드(Auto-reload) 카운터
 - 16비트 카운터 : $2^{16} = 65536$, 즉 0 ~ 65535까지 셀 수 있음
 - 16비트의 프리스케일러(prescaler) 보유
 - 인터럽트 기능
 - 업데이트(update)
 - 클럭 소스
 - 내부클럭

STM32F405 타이머의 동작모드

- STM32F405 타이머의 일반 동작 모드
 - 가장 기본적인 동작모드가 업카운트 모드
- STM32F405 타이머의 동작
 - 동작 모드 결정
 - 타이머에 사용할 클럭 소스와 프리스케일러 결정
 - 원하는 타이머의 주기 및 그 주기 동안의 시간을 정확히 세기 위한 타이머 클럭의 수 결정
 - 카운터 레지스터의 카운터 오버플로우값 설정(오버플로우 인터럽트 사용)

$$\text{Timer Period} = \frac{1}{\text{Clock Frequency} / \text{Prescaler}} \cdot \text{Auto - Reload Value}$$

STM32F405 타이머의 동작모드

- STM32F405 타이머의 동작 예
 - 동작 모드 결정 : 일반동작모드
 - 타이머에 사용할 클럭 소스와 프리스케일러 결정
 - 내부클럭 : 84MHz
 - 프리스케일러 : 84000
 - 타이머 클럭 주파수 : $84000000/84000 = 1\text{KHz}$
 - 타이머 클럭 주기 : 약 1ms ($1/1000 = 0.001$)
 - 원하는 타이머의 주기 및 그 주기 동안의 시간을 정확히 세기 위한 타이머 클럭의 수 결정
 - 1 주기당 타이머 클럭 개수 = 타이머 클럭 주파수 * 타이머 시간
 - 만약 1s의 타이머를 만들기 원한다면 $1\text{k} * 10000 = 1\text{s}$
 - 카운터 레지스터의 카운터 시작 값 설정
 - TIMx-ARR레지스터(Auto-reload register) = 1000
 - 카운터값이 TIMx-ARR값보다 커지게 되면 카운터가 0이 되면서 오버플로우 발생

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CR1 (control register 1)
 - 타이머 제어 레지스터 1
 - 타이머x의 각종 기능 제어
 - CKD[1:0](Clock Division)
 - CKD 비트는 타이머 클럭(CK_INT)과 디지털 필터(ETR,TIx)에 사용되는 샘플링 클럭 사이의 분배비를 나타냄
 - 00 : t DTS = t CK_INT
 - 01 : t DTS = 2 × t CK_INT
 - 10 : t DTS = 4 × t CK_INT
 - 11 : Reserved
 - ARPE(Auto-Reload Preload enable)
 - 언더/오버플로우 발생시 TIMx_ARR 레지스터의 사용여부를 결정
 - 1로 설정하면 enable 됨

15~10	9	8	7	6	5	4	3	2	1	0
Reserved	CKD[1:0]	ARPE	CMS	DIR	OPM	URS	UDIS	CEN		

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CR1 (control register 1)
 - 타이머 제어 레지스터 1
 - CMS(Center-aligned Mode Selection)
 - 이 비트를 조절하여 카운터의 동작을 결정
 - 00 : Edge-aligned mode
 - 카운터가 DIR 비트에 의한 방향으로(up/down) 카운트
 - 01 : Center-aligned mode1
 - 카운터가 up, down 방향으로 교대로 카운트되며, Output compare interrupt가 설정되어 있다면, 이는 다운카운팅일 때 발생
 - 10 : Center-aligned mode2
 - 카운터가 up, down 방향으로 교대로 카운트되며, Output compare interrupt가 설정되어 있다면, 이는 업 카운팅일 때 발생
 - 11 : Center-aligned mode3
 - 카운터가 up, down 방향으로 교대로 카운트되며, Output compare interrupt가 설정되어 있다면, 이는 업/다운 카운팅일 때 발생

15~10	9	8	7	6	5	4	3	2	1	0
Reserved	CKD[1:0]	ARPE	CMS	DIR	OPM	URS	UDIS	CEN		

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CR1 (control register 1)
 - 타이머 제어 레지스터 1
 - DIR (Direction)
 - 이 비트가 1이면, 카운터는 업 카운터로, 0일 경우 다운 카운터로 동작
 - OPM (One Pulse Mode)
 - 0 : 카운터가 업데이트 이벤트에도 멈추지 않음
 - 1 : 카운터가 다음 업데이트 이벤트까지 카운팅을 멈춤(CEN 비트를 클리어)
 - URS (Update Request Source)
 - 0 : 카운터나 UGq비트의 설정, 슬레이브 모드 컨트롤러에 의한 업데이트 발생을 원인으로 업데이트 이벤트를 발생시킴
 - 1 : 오로지 DMA 요청이나 카운터에 의해서만 업데이트 이벤트를 발생시킴
 - UDIS (Update Disable)
 - 0 : 업데이트 이벤트(UEV) enable
 - 1 : 카운터는 동작하지만 업데이트 이벤트는 Disable
 - CEN (Counter enable)
 - 1로 설정하면 카운터가 동작하고, 0으로 설정하면 카운터가 멈춤

15~10	9	8	7	6	5	4	3	2	1	0
Reserved	CKD[1:0]	ARPE	CMS	DIR	OPM	URS	UDIS	CEN		

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CR2 (control register 2)
 - 타이머 제어 레지스터 2
 - TI1S(TIM1 Selection)
 - 0 : TIMx_CH1 핀이 TI1 입력으로 연결
 - 1 : TIMx_CH1, 2, 3 핀이 TI1 입력으로 연결 (XOR combination)
 - MMS (Master Mode Selection)
 - 이 비트들은 동기화를 위해 슬레이브 타이머에 보낼 마스터 모드의 정보 (trigger output : TRGO)를 설정
 - 000 : Reset - TIMx_EGR레지스터의 UG비트가 trigger output로 사용
 - 트리거입력에 의해 발생되면, TRGO의 신호는 리셋동안 지연
 - 001 : Enable - 카운터 활성화 신호, CNT_EN이 TRGO로 사용
 - 이는 여러 타이머를 동시에 실행시키거나, 슬레이브 타이머를 활성화하기 유용. CNT_EN은 CEN제어 비트와 gated mode의 트리거 입력 사이의 OR연산에 의해 발생. 마스터/슬레이브 모드가 설정된 경우가 아니라면, CNT_EN은 trigger output(TRGO)에 의해 제어되며, 여기엔 약간의 딜레이가 있음

15~8	7	6	5	4	3	2	1	0
Reserved	TI1S	MMS[2:0]	CCDS	Reserved				

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CR2 (control register 2)
 - MMS (Master Mode Selection)
 - 010 : Update - Update 이벤트가 TRGO로 선택
 - 예를 들면, 마스터타이머가 슬레이브 타이머의 prescaler로 사용
 - 011 : Compare pulse - Capture/compare match가 발생된 것 같이 CC1IF 플래그가 1로 설정되어 있을 때, trigger output은 positive pulse를 전송
 - 100 : Compare - OC1REF 신호가 trigger output(TRGO)로 사용
 - 101 : Compare - OC1REF 신호가 trigger output(TRGO)로 사용
 - 110 : Compare - OC1REF 신호가 trigger output(TRGO)로 사용
 - 111 : Compare - OC1REF 신호가 trigger output(TRGO)로 사용
 - CCDS (Capture / Compare DMA Selection)
 - 0 : CCx이벤트가 발생한 것 같이 CCx DMA 요청을 전송
 - 1 : 업데이트 이벤트가 발생한 것 같이 CCx DMA 요청을 전송

15~8	7	6	5	4	3	2	1	0
Reserved	TI1S	MMS[2:0]	CCDS	Reserved				

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_DIER (DMA/Interrupt enable register)
 - 타이머 DMA/인터럽트 설정 레지스터
 - DMA 인터럽트와 타이머 인터럽트를 설정하는 레지스터
 - 교재에서는 DMA를 직접적으로 다루지 않으므로, 그 내용은 데이터시트를 참조
 - TIE (Trigger Interrupt Enable)
 - 0 : Trigger interrupt disabled
 - 1 : Trigger interrupt enabled
 - CC4IE (Capture / Compare 4 Interrupt Enable)
 - 0 : CC4 interrupt disabled
 - 1 : CC4 interrupt enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	TI1S	TIE	Res	CC4IE	CCDS	CC2IE	CC1IE	UIE

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_DIER (DMA/Interrupt enable register)
 - CC3IE (Capture / Compare 3 Interrupt Enable)
 - 0 : CC3 interrupt disabled
 - 1 : CC3 interrupt enabled
 - CC2IE (Capture / Compare 2 Interrupt Enable)
 - 0 : CC2 interrupt disabled
 - 1 : CC2 interrupt enabled
 - CC1IE (Capture / Compare 1 Interrupt Enable)
 - 0 : CC1 interrupt disabled
 - 1 : CC1 interrupt enabled
 - UIE (Update Interrupt Enable)
 - 0 : Update interrupt disabled
 - 1 : Update interrupt enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	TDE	Res	CC4DE	CC3DE	CC2DE	CC1DE	UDE	TI1S	TIE	Res	CC4IE	CCDS	CC2IE	CC1IE	UIE

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_SR (status register)
 - 타이머 상태 레지스터
 - CC4OF (Capture/Compare 4 Overcapture Flag)
 - CC3OF (Capture/Compare 3 Overcapture Flag)
 - CC2OF (Capture/Compare 2 Overcapture Flag)
 - CC1OF (Capture/Compare 1 Overcapture Flag)
 - 현재 채널이 입력캡쳐모드로 설정되어 있을때, 하드웨어에서 이 비트를 1로 설정하며, 소프트웨어에서 0으로 클리어 할 수 있음
 - 0 : overcapture가 발생되지 않음
 - 1 : CCnIF (n:1~4)비트가 설정되어 있는 동안, 카운터값이 TIMx_CCRn 레지스터 값과 일치(이벤트 감지)되었음을 알림
 - TIF (Tigger Interrupt Flag)
 - slave mode에서 사용되는 비트이며, 자세한 내용은 데이터 시트를 참조

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved	TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF	

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_SR (status register)
 - CC4IF (Capture/Compare 4 interruptFlag)
 - CC3IF (Capture/Compare 3 interruptFlag)
 - CC2IF (Capture/Compare 2 interruptFlag)
 - CC1IF (Capture/Compare 1 interruptFlag)
 - 1) CCn (n:1~4)가 출력으로 설정되어 있을 때
 - TIMx_CNT 레지스터와 TIMx_CCRn레지스터가 match 되면 이 비트는 하드웨어에 의해 1로 설정, 그리고 소프트웨어에서 0으로 클리어
 - 2) CCn (n:1~4)가 입력으로 설정되어 있을 때
 - ICn(n:1~4)핀의 펄스카운터를 저장하는 TIMx_CCRn레지스터와 카운터 값이 match되면 이 비트는 하드웨어에 의해 1로 설정
 - UIF (Update interrupt flag)
 - update 이벤트가 발생되면 이 비트는 하드웨어에 의해 1로 설정되며, 소프트웨어에 의해 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	CC4OF	CC3OF	CC2OF	CC1OF	Reserved	TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF			

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_EGR (event generation register)
 - 타이머 이벤트 설정 레지스터
 - CC4G (Capture/Compare 4 generation)
 - CC3G (Capture/Compare 3 generation)
 - CC2G (Capture/Compare 2 generation)
 - CC1G (Capture/Compare 1 generation)
 - 이 비트를 소프트웨어에서 1로 설정하면, capture/compare event를 발생
 - 1) CCn (n:1~4)채널이 출력으로 설정되어 있을 때
 - CCnIF비트가 설정되어 있으면, 인터럽트/DMA요청을 발생
 - 2) CCn(n: 1~4)채널이 입력으로 설정되어 있을 때
 - TIMx_CCRn레지스터가 카운터값이 같아졌을때(capture), CCnIF비트가 1로 설정되어 있다면, 인터럽트/DMA요청을 발생.
 - CCnOF플래그가 1로 set되기 전에 CCnIF플래그가 먼저 설정되어야 함

15~7	6	5	4	3	2	1	0
Reserved	TG	Res	CC4G	CC3F	CC2G	CC1G	UG

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_EGR (event generation register)
 - UG (Update generation)
 - 이 비트는 소프트웨어에 의해 Set되며, 하드웨어에 의해 자동적으로 클리어
 - 1 : 카운터를 초기화하고, 카운터 값을 카운터방향에 따른 초기값(업카운팅이면, ARR 레지스터값으로, 다운 카운팅/중앙정렬모드인 경우는 0으로)으로 설정

15~7	6	5	4	3	2	1	0
Reserved	TG	Res	CC4G	CC3F	CC2G	CC1G	UG

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)
 - 타이머 캡쳐/비교 모드 설정 레지스터 1
 - 채널들은 입력(capture mode) 또는 출력(compare mode)로 사용
 - 채널의 방향은 CCxS 비트로 설정
 - 다른 모든 비트들은 입/출력 모드에 대한 각각의 동작 설정을 할 수 있음
 - OCxx 비트들은 채널이 출력일 때, ICxx 비트들은 채널이 입력일 때 사용되므로 상황에 맞게 레지스터의 동작을 설정해야 함

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]					

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)
 - Output compare mode
 - OC2S[1:0] (Output Compare 2 Selection)
 - 이 비트들은 타이머 채널의 입/출력 방향을 결정하는 역할
 - 00: CC2 채널을 출력으로 설정
 - 01: CC2 채널을 입력으로 설정한다 IC2는 TI2로 연결
 - 10: CC2 채널을 입력으로 설정한다 IC2는 TI1로 연결
 - 11: CC2 채널을 입력으로 설정한다 IC2는 TRC로 연결
 - 이 모드는 TIMx_SMCR레지스터의 TS비트를 통해 internal trigger input0이 선택되어 있을 때만 동작

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]					

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)
 - Output compare mode
 - OC1CE (Output Compare 1 Clear Enable)
 - 0 : OC1Ref 가 ETRF 입력에 의해 영향 받지 않음
 - 1 : OC1Ref 핀이 ETRF 입력의 High level을 검출하는 동안 0으로 클리어
 - OC1M (Output Compare 1 Mode)
 - 출력기준신호 OC1REF(OC1,OC1N에서 유래한)의 동작을 결정
 - 000 : Frozen - 출력에 영향을 끼치지 않음
 - 001 : Active mode - 카운터와 TIMx_CCR1가 매치되었을 때, OC1REF신호는 High가 됨
 - 010 : Inactive mode - 카운터와 TIMx_CCR1가 매치되었을 때, OC1REF신호는 Low가 됨

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]					

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)
 - Output compare mode
 - OC1M (Output Compare 1 Mode)
 - 011 : Toggle mode - TIMx_CNT=TIM_CCR1 일때, OC1REF은 토글
 - 100 : Inactive level - OC1REF = low
 - 101 : Active level - OC1REF = High
 - 110 : PWM mode 1 - 업카운팅에서 채널1은 TIMx_CNT가 TIMxCCR1 보다 작을 동안은 OC1REF는 High가 되고, TIMx_CNT가 TIMxCCR1보다 클 때는 low가 됨. 다운카운팅일때는, 업카운팅일때의 반대
 - 111 : PWM mode 2 - 업카운팅에서 채널1은 TIMx_CNT가 TIMxCCR1 보다 작을 동안은 OC1REF는 Low가 되고, TIMx_CNT가 TIMxCCR1보다 클 때는 High가 됨. 다운카운팅일때는, 업카운팅일때의 반대

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]					

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)
 - Output compare mode
 - OC1PE (Output Compare 1 Preload enable)
 - 0 : TIMx_CCR1레지스터를 disable. 이렇게 하면 TIMx_CCR1레지스터의 값을 수정할 수 있음
 - 1 : TIMx_CCR1레지스터를 enable
 - OC1FE (Output Compare 1 Fast enable)
 - 0 : CC1 은 트리거가 ON일 때, 카운터와 CCR1값에 의해 노멀하게 동작 트리거의 엣지가 발생했을 때, CC1출력을 활성화시키기 위한 최소 딜레이는 5클럭 사이클
 - 1 : active edge 의 트리거 입력은 CC1출력과 비교하기 위해 동작, 그러면 OC는 비교결과와 독립적으로 compare level으로 설정 트리거 입력을 샘플하기 위해 딜레이를 가지고, CC1출력은 3클럭 사이클로 감소되어 활성화됨 OCFE는 PMM1, 2모드에서 동작
 - CC1S(Capture/Compare 1 Selection)
 - 이 비트들은 채널의 방향(입/출력)을 결정

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]						
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]									

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)

- Input compare mode

- IC2F (Input Capture 2 Filter)
 - IC2PSC [2:0] (Input Capture 2 Prescaler)
 - CC2S[1:0] (Capture/compare 2 Selection)
 - IC1F[3:0] (Input Capture 1 Filter)

- TI1 입력 샘플에 사용될 주파수와 TI1에 적용된 디지털 필터의 length에 사용될 주파수를 결정

- 0000 : No filter, sampling is done at f DTS
 - 0001 : f SAMPLING =f CK_INT , N=2
 - 0010 : f SAMPLING =f CK_INT , N=4
 - 0011 : f SAMPLING =f CK_INT , N=8
 - 0100 : f SAMPLING =f DTS /2, N=6
 - 0101 : f SAMPLING =f DTS /2, N=8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]					

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)
 - Input compare mode
 - IC1F[3:0] (Input Capture 1 Filter)
 - 0110 : f SAMPLING =f DTS /4, N=6
 - 0111 : f SAMPLING =f DTS /4, N=8
 - 1000 : f SAMPLING =f DTS /8, N=6
 - 1001 : f SAMPLING =f DTS /8, N=8
 - 1010 : f SAMPLING =f DTS /16, N=5
 - 1011 : f SAMPLING =f DTS /16, N=6
 - 1100 : f SAMPLING =f DTS /16, N=8
 - 1101 : f SAMPLING =f DTS /32, N=5
 - 1110 : f SAMPLING =f DTS /32, N=6
 - 1111 : f SAMPLING =f DTS /32, N=8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]					

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)
 - Input compare mode
 - IC1PSC (Input Capture 1 Prescaler)
 - CC1입력에 동작하는 프리스케일러의 비율을 결정
 - 프리스케일러는 TIMx_CCER레지스터의 CC1E=0일 동안 리셋됨.
 - IC2PSC [2:0] (Input Capture 2 Prescaler)
 - 00 : no prescaler, capture is done each time an edge is detected on the capture input
 - 01 : capture is done once every 2 events
 - 10 : capture is done once every 4 events
 - 11 : capture is done once every 8 events

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]		OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]			
IC2F[3:0]			IC2PSC[1:0]			IC1F[3:0]			IC1PSC[1:0]						

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR1 (capture/compare mode register 1)
 - Input compare mode
 - CC1S(Capture/Compare 1 Selection)
 - 채널의 방향(입/출력)을 결정
 - 00 : CC1 채널을 Output으로 설정
 - 01 : CC1 채널을 입력으로 설정, IC1는 TI1로 연결
 - 10 : CC1 채널을 입력으로 설정, IC1는 TI1로 연결
 - 11 : CC1채널을 입력으로 설정, IC1는 TRC로 연결
이 모드는 TIMx_SMCR레지스터의 TS비트를 통해 internal trigger input이 선택되어 있을 때만 동작

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		
IC2F[3:0]			IC2PSC[1:0]				IC1F[3:0]			IC1PSC[1:0]					

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCMR2 (capture/compare mode register 2)
 - 타이머 캡쳐/비교 모드 설정 레지스터 2
 - TIMx_CCMR2 레지스터는 TIMx_CCMR1레지스터와 채널만 다를 뿐, 각 비트의 역할이 같으므로 TIMx_CCMR1레지스터 설명을 참조

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]						
IC4F[3:0]			IC4PSC[1:0]				IC3F[3:0]			IC3PSC[1:0]									

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCER (capture/compare enable register)
 - 타이머 캡쳐/비교 모드 enable 레지스터
 - CC4NP (Capture / Comapre 4 output Polarity)
 - CC1NP 비트 참조
 - CC4P (Capture / Comapre 4 output Polarity)
 - CC1P 비트 참조
 - CC4E (Capture / Comapre 4 output enable)
 - CC1E 비트 참조
 - CC3NP (Capture / Comapre 3 output Polarity)
 - CC1NP 비트 참조
 - CC3P (Capture / Comapre 3 output Polarity)
 - CC1P 비트 참조
 - CC3E (Capture / Comapre 3 output enable)
 - CC1E 비트 참조

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res	CC4P	CC4E	CC3NP	Res	CC3P	CC3E	CC2NP	Res	CC2P	CC2E	CC1NP	Res	CC1P	CC1E

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCER (capture/compare enable register)
 - CC2NP (Capture / Comapre 2 output Polarity)
 - CC1P 비트 참조
 - CC2P (Capture / Comapre 2 output Polarity)
 - CC1E 비트 참조
 - CC2E (Capture / Comapre 2 output enable)
 - CC1P 비트 참조
 - CC1NP (Capture / Comapre 1 output Polarity)
 - CC1 채널이 출력으로 사용되면 클리어를 유지해야 함
 - CC1 채널이 입력으로 사용될 경우 CC1P와 조합해서 사용

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res	CC4P	CC4E	CC3NP	Res	CC3P	CC3E	CC2NP	Res	CC2P	CC2E	CC1NP	Res	CC1P	CC1E

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCER (capture/compare enable register)
 - CC1P (Capture / Comapre 1 output Polarity)
 - CC1 채널이 출력일 때의 내용은 PWM장을 참조
 - CC1 채널이 입력으로 설정되어 있을 때(CC1NP/CC1P 비트)
 - 00 : 비반전/상승 에지
 - 01 : 반전/하강 에지
 - 10 : 예약
 - 11 : 비 반전/양쪽 에지
 - CC1E (Capture / Comapre 1 output enable)
 - CC1 채널이 출력일 때의 내용은 PWM장을 참조
 - CC1 채널이 입력으로 설정되어 있을 때
 - 0 : Capture disabled
 - 1 : Cpature enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res	CC4P	CC4E	CC3NP	Res	CC3P	CC3E	CC2NP	Res	CC2P	CC2E	CC1NP	Res	CC1P	CC1E

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CNT (counter register)
 - 타이머 카운터 레지스터
 - CNT[31:16]
 - 32비트 타이머만 사용
 - CNT[15:0]
 - 카운터를 저장하는 레지스터

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNT[31:16] (타이머에 따라)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_PSC (Prescaler register)
 - 타이머 프리스케일러 레지스터
 - CNT[15:0]
 - 카운터 클럭 주파수 CK_CNT는 $f_{CK_PSC} / (PSC[15:0] + 1)$ 공식으로 구함

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_ARR (auto-reload register)
 - ARR[31:16]
 - 32비트 타이머만 사용
 - ARR[15:0]
 - auto-reload 될 카운터 값을 저장하는 레지스터

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (타이머에 따라)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															

STM32F405 타이머(TIM2 to TIM5) 레지스터

- TIMx_CCRn (capture/compare register n)
 - 타이머 캡쳐/비교 레지스터 n(n=1~4)
 - CCRn[31:16]
 - 32비트 타이머만 사용
 - CCRn[15:0]
 - CCn이 출력일 때는, TIMx_CNT 카운터와 비교되는 값을 가지고 있고, OC1 출력을 발생
 - CCn이 입력일 때는, CCRn 레지스터의 값은 마지막 input capture n event에 의해 변환되어 카운터 값이 됨

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCRn[31:16] (타이머에 따라)															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CCRn[15:0]															

STM32F405 타이머 update 인터럽트

- STM32F405 update 인터럽트 프로그램
 - main.c

```
#include "stm32f4xx.h"
int main(void)
{
    // ...
    TIM_TimeBaseInitTypeDef TIMx_TimeBaseInitStruct;
    NVIC_InitTypeDef NVIC_InitStructure;
    // ...
    // TIMx_IRQHandler 의 인터럽트 허용 및 실행주기 설정
    NVIC_Init(&NVIC_InitStructure);

    // 타이머x Update 설정
    TIM_TimeBaseInit(xxx);
    TIM_Cmd(TIMx,ENABLE);
    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
    // ...
}
```

STM32F405 타이머 update 인터럽트

- STM32F405 update 인터럽트 프로그램
 - main.c

```
void TIMx_IRQHandler(void)
{
    if(TIM_GetFlagStatus(TIMx,TIM_IT_Update)==SET)
    {
        TIM_ClearITPendingBit(TIMx,TIM_FLAG_Update);
        // 처리 루틴
    }
}
```

실습 1 : 타이머로 LED 점멸

- 실습 개요
 - STM32F405 마이크로컨트롤러의 타이머 기능을 이용하여 LED를 점멸시키기
 - 타이머를 이용하여 정확히 1초마다 LED가 점멸하도록 함
 - 타이머3의 일반 동작 모드를 사용
- 실습 목표
 - 타이머3의 동작원리 이해
 - 타이머3 제어 방법의 습득(관련 레지스터 이해)
 - 오버플로우 인터럽트 제어 프로그램 방법 습득

실습 1 : 타이머로 LED 점멸

- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
 - 사용할 타이머/카운터 결정
 - 여기서는 타이머/카운터 3을 사용
 - 동작 모드 결정
 - 여기서는 일반 동작 모드 사용
 - 카운트 방향을 업카운트(0 부터 TIMx-ARR까지)로 선택
 - TIM3_CR1레지스터의 DIR을 0으로 클리어
 - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
 - 여기서는 클럭(84MHz)을 사용
 - 타이머 3의 클럭은 메인 클럭(168Mhz)의 $\frac{1}{2}$ 을 사용
 - 샘플링 클럭의 분주비는 1:1로 설정
 - 타이머 클럭 프리스케일러는 8399로 설정(TIM3_CR1/CKD:00, TIM3_PSC: 8399)
 - 84MHz 클럭의 1분주비와 8400 프리스케일러로 나누어 지므로 10KHz(0.1ms)로 동작
 - 타이머 주기 결정
 - 1초를 만들기 위해 1ms을 타이머 주기로 결정
 - TIM3_ARR레지스터의 값을 10000으로 설정
 - 타이머 주기 * 카운터 수의 공식에 의해 1초마다 Update(오버플로우)가 발생하여 1초마다 인터럽트가 발생

실습 1 : 타이머로 LED 점멸

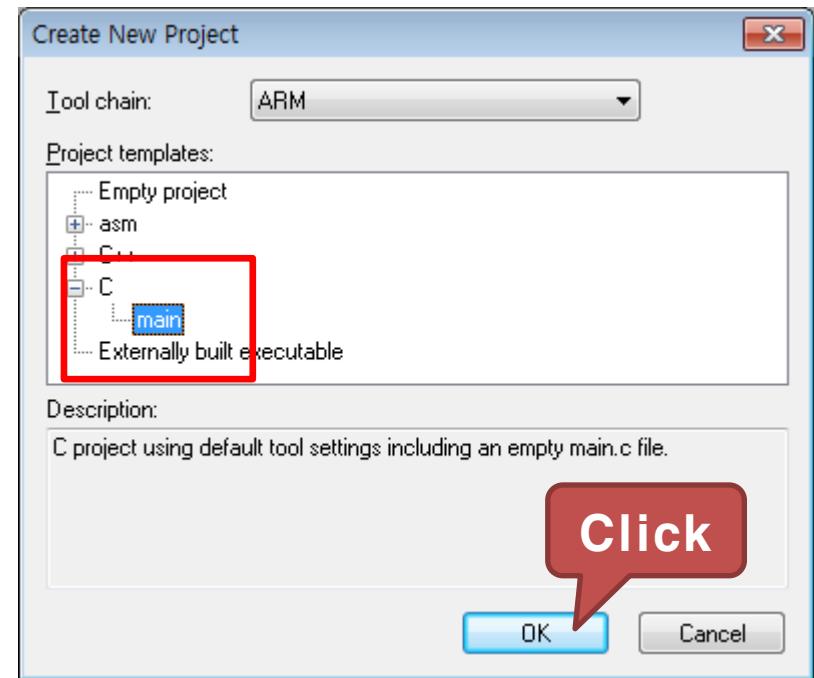
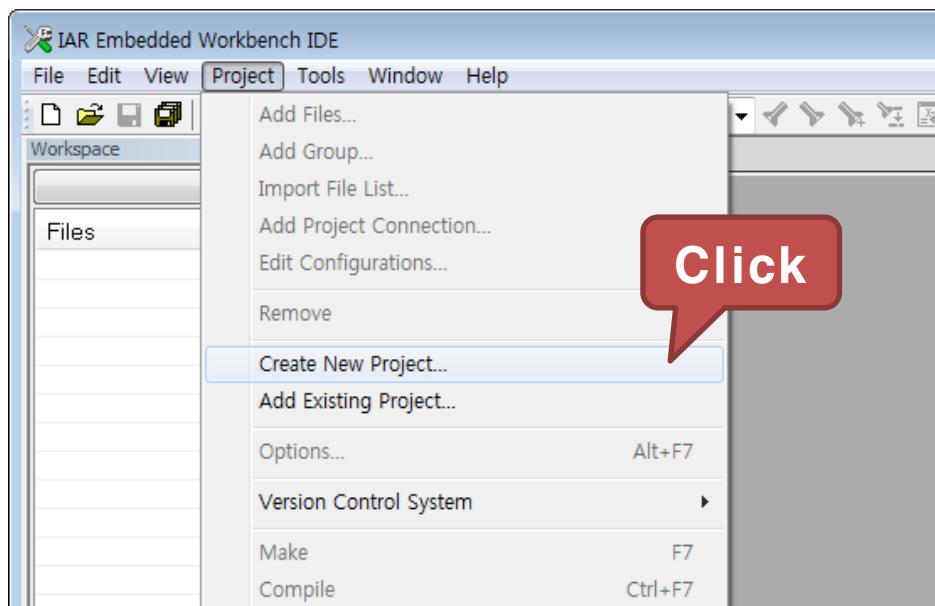
- TIM 라이브러리
 - STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
 - TIM_TimeBaseInitTypeDef 구조체

TIM_TimeBaseInitTypeDef구조체	설 명
uint16_t TIM_Prescaler	타이머 프리스케일러를 설정한다. (TIMx_PSC 레지스터)
uint16_t TIM_CounterMode	타이머 카운트 방향 설정 (TIMx_CR1의 DIR비트)
uint32_t TIM_Period	타이머 주기값을 설정한다. (TIMx_ARR레지스터)
uint16_t TIM_ClockDivision	타이머 클럭 분주 설정 (TIMx_CR1의 CKD비트)
uint8_t TIM_RepetitionCounter	반복 카운터값을 설정한다. (TIMx_RCR레지스터, 타이머 1과 8 만 사용)

실습 1 : 타이머로 LED 점멸

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch6” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “timerLed”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1 : 타이머로 LED 점멸

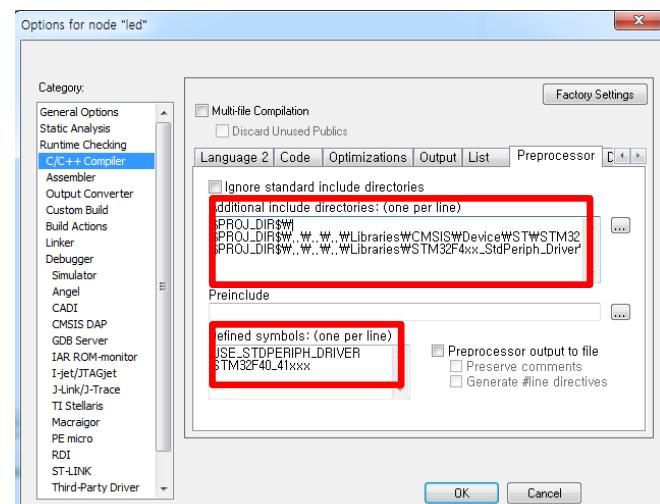
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch6\timerLed	system_stm32f4xx.c
Cortex_Example\Projects\ch6\timerLed	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_misc.c

실습 1 : 타이머로 LED 점멸

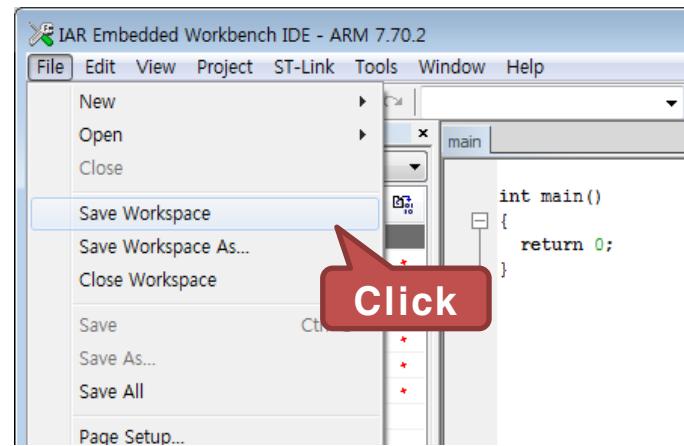
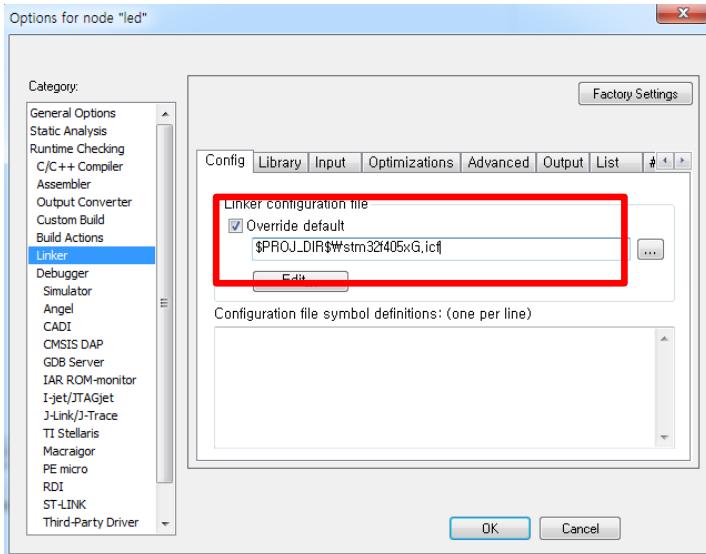
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1 : 타이머로 LED 점멸

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1 : 타이머로 LED 점멸

- 구동 프로그램
 - main.c 코드 작성

```
#include "stm32f4xx.h"
unsigned int LED_Data=0x0000;

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    NVIC_InitTypeDef      NVIC_InitStructure;
    // GPIOC, Timer3 클럭 인가
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
```

실습 1 : 타이머로 LED 점멸

- 구동 프로그램

- main.c 코드 작성

```
GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
GPIO_Init(GPIOC, &GPIO_InitStructure);
// system_stm32f4xx.c 참조
// TIM3 input clock (TIM3CLK) is set to 2 * APB1 clock (PCLK1),
// since APB1 prescaler is different from 1.
// TIM3CLK = 2 * PCLK1
// PCLK1 = HCLK / 4 => TIM3CLK = HCLK / 2 = SystemCoreClock /2
// TIM_CounterMode를 TIM_CounterMode_Up로 설정하면,
// 카운트값이 TIMx_ARR값과 같아질때, Update 이벤트가 발생
// TIM_Prescaler를 8399, TIM_ClockDivision를 TIM_CKD_DIV1(1분주)로
// 설정하면, 타이머3 카운트 클럭은
// (84MHz/1)/(8399+1) = 10KHz (0.1ms)임.
// TIM_Period를 9999로 설정했으니(TIMx_ARR), 카운터가 이 값과
// 같아지면, (0.1ms x 10000(9999+1) = 1s)
// (168Mhz/2)/8400 = 10KHz(1ms)
```

실습 1 : 타이머로 LED 점멸

- 구동 프로그램
 - main.c 코드 작성

```
//...
TIM_TimeBaseStructure.TIM_Prescaler = 8400-1;
TIM_TimeBaseStructure.TIM_Period = 10000-1;
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

// 타이머 3 인터럽트의 우선순위를 설정
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
NVIC_InitStructure.NVIC IRQChannel = TIM3_IRQn;
NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 0x01;
NVIC_InitStructure.NVIC IRQChannelSubPriority = 0x01;
NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

실습 1 : 타이머로 LED 점멸

- 구동 프로그램

- main.c 코드 작성

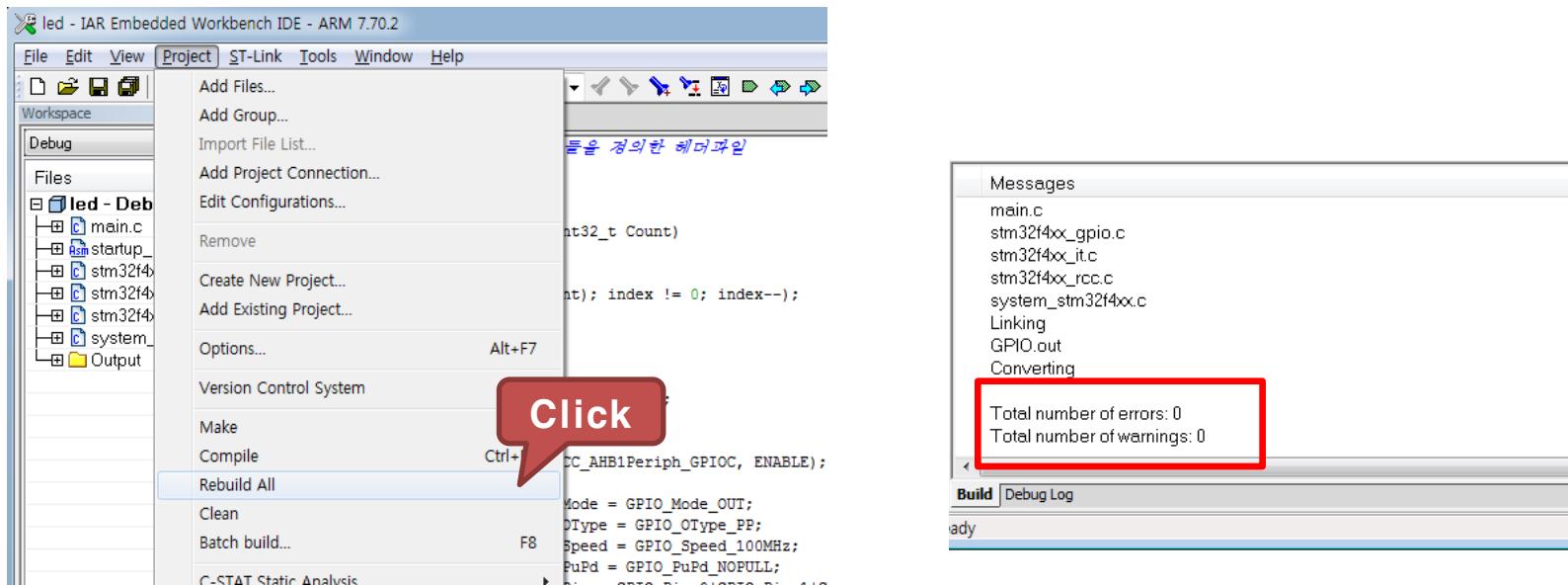
```
// TIM3_SR 레지스터의 UIF(Update interrupt flag) 비트를 클리어
TIM_ClearITPendingBit(TIM3, TIM_IT_Update);      // 인터럽트 허용
TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);        // 타이머3 동작
TIM_Cmd(TIM3,ENABLE);
while(1){}
}

void TIM3_IRQHandler(void) {
    if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) {
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
        GPIO_Write(GPIOC, LED_Data);
        LED_Data++;          // LED_Data값을 1씩 증가
        // LED_Data값이 0x0F 보다 크면 0으로 초기화
        if(LED_Data > 0x0F) LED_Data = 0;
    }
}
```

실습 1 : 타이머로 LED 점멸

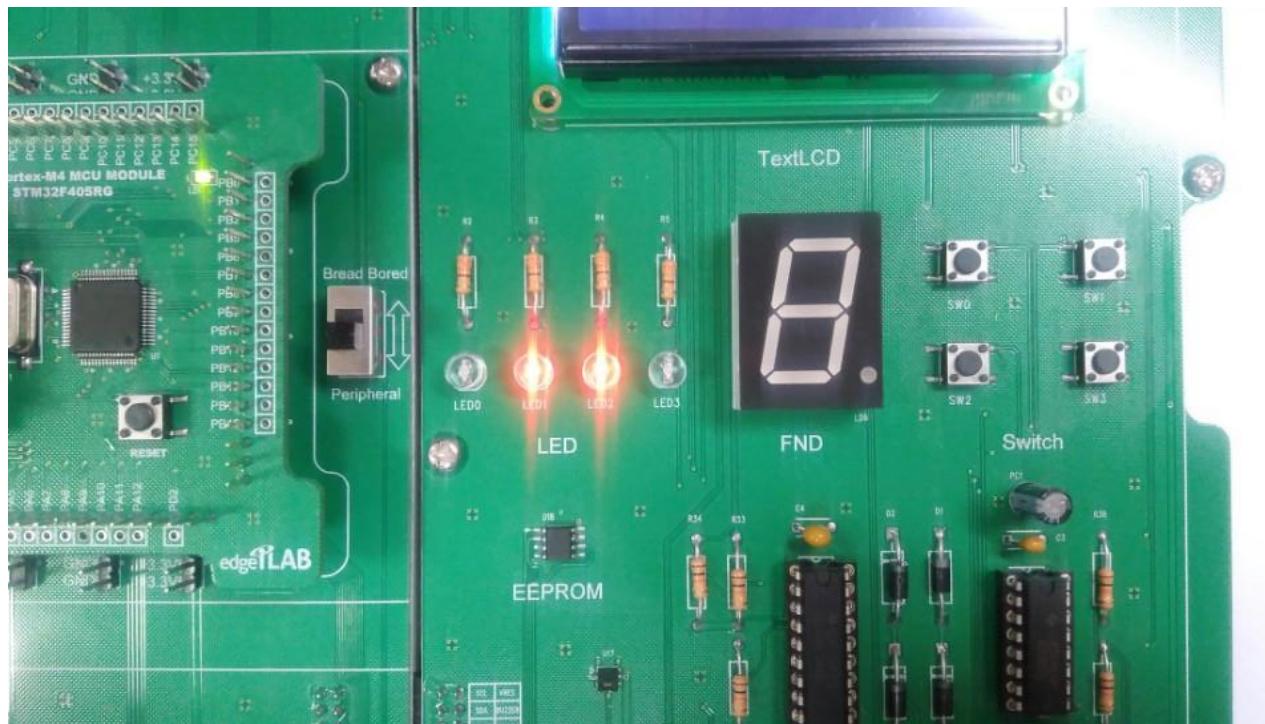
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 timerLed 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 1 : 타이머로 LED 점멸

- 실행 결과
 - LED의 불이 1초마다 순차적으로 점멸



실습 2 : 타이머를 이용한 디지털 시계

- 실습 개요
 - 타이머를 이용하여 디지털 시계의 기능을 설계
 - Array-FND 모듈에 마이크로 컨트롤러 출력 포트를 연결하고, 클럭을 이용하여 일정 카운트 기능을 수행
 - 타이머3의 일반 모드 동작을 사용
 - 시계는 초만 표시
- 실습 목표
 - 타이머/카운터 활용 방법의 습득(관련 레지스터 이해)
 - 디지털 시계(초/분) 구현 방법 이해

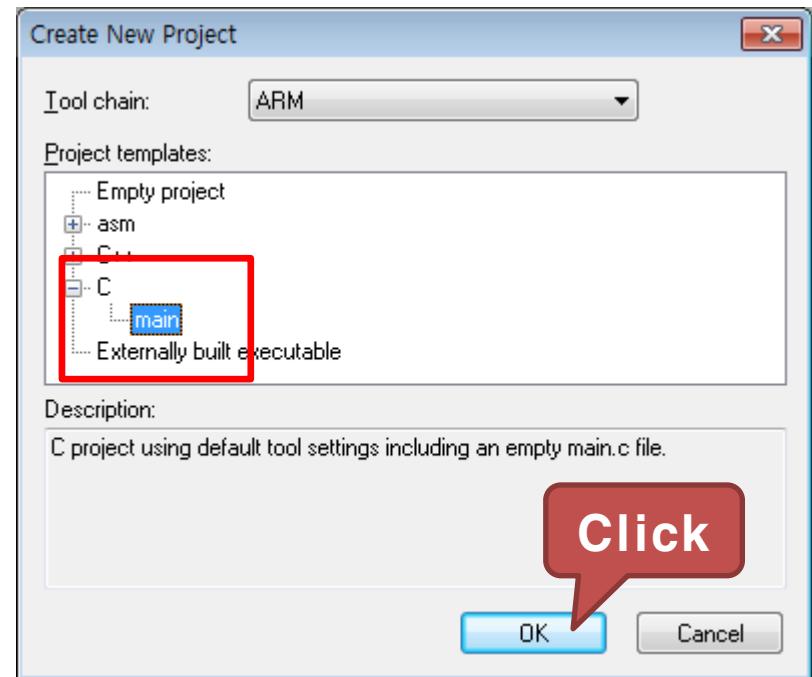
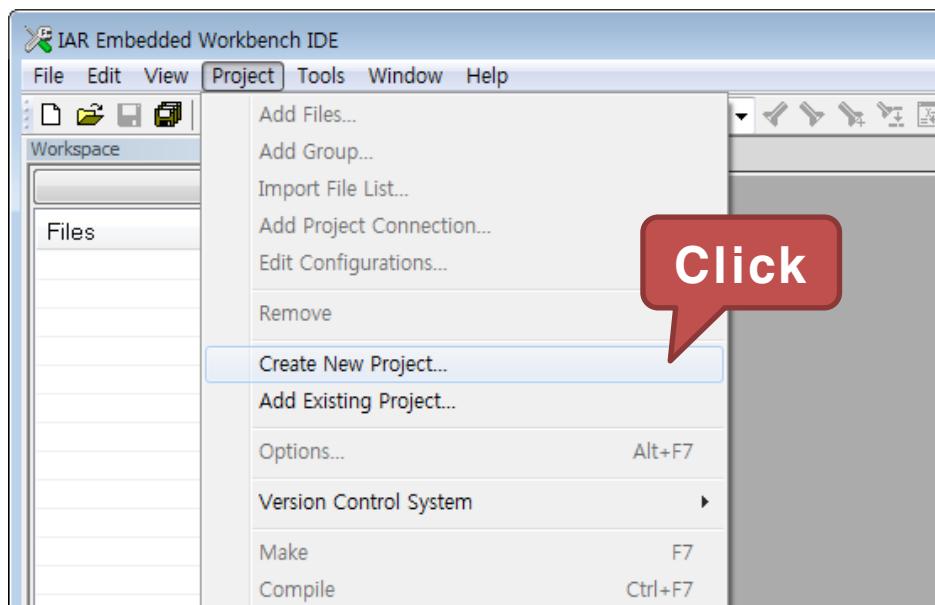
실습 2 : 타이머를 이용한 디지털 시계

- 구동 프로그램 : 사전 지식
 - 타이머를 이용하여 디지털 시계
 - 타이머로 LED 점멸하기 예제와 거의 유사
 - 시계 표시를 위해 Array-FND를 사용
 - 타이머를 이용하여 정확히 1초를 카운트

실습 2 : 타이머를 이용한 디지털 시계

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch6” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “timerClock”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : 타이머를 이용한 디지털 시계

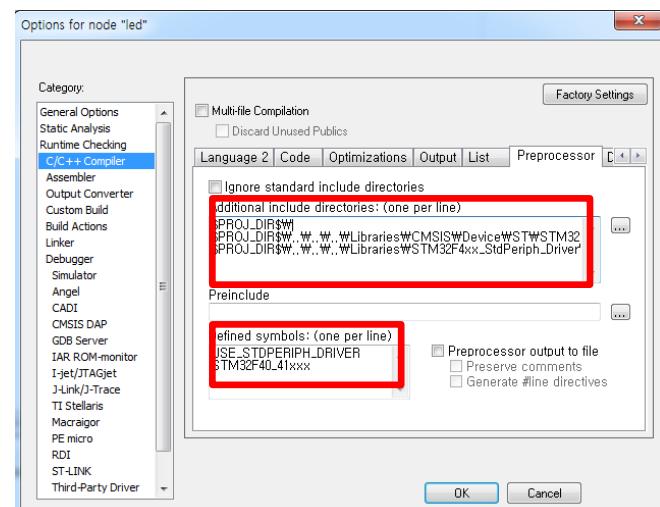
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch6\timerClock	system_stm32f4xx.c
Cortex_Example\Projects\ch6\timerClock	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_misc.c

실습 2 : 타이머를 이용한 디지털 시계

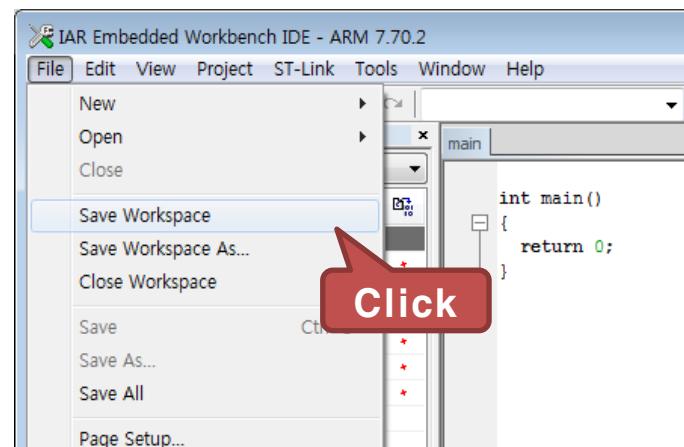
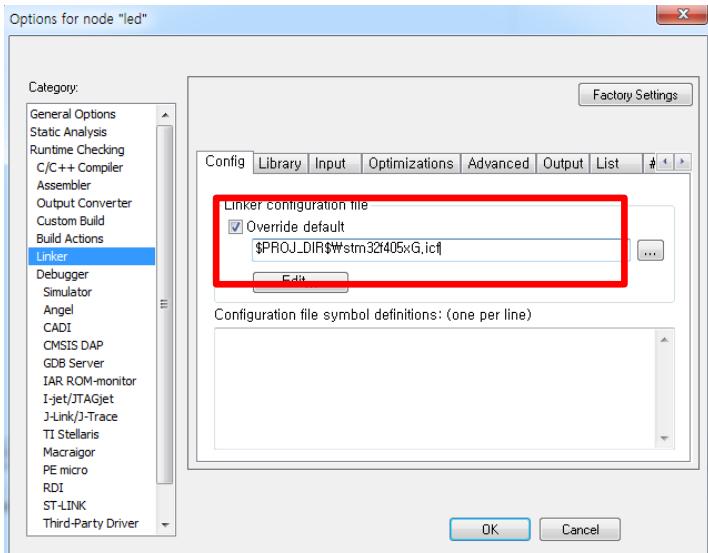
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : 타이머를 이용한 디지털 시계

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : 타이머를 이용한 디지털 시계

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

// 7-Segment에 표시할 글자의 입력 데이터를 저장
unsigned int FND_DATA_TBL [] = {0x3F00, 0X0600, 0X5B00, 0X4F00,
                                0X6600, 0X6D00, 0X7C00, 0X0700, 0X7F00, 0X6700, 0X7700,
                                0X7C00, 0X3900, 0X5E00, 0X7900, 0X7100, 0X0800, 0X8000};
unsigned char time_s=0; // 초를 세는 변수

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    NVIC_InitTypeDef  NVIC_InitStructure;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

실습 2 : 타이머를 이용한 디지털 시계

- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  
GPIO_InitStructure.GPIO_Pin =  
    GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11|  
    GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;  
GPIO_Init(GPIOC, &GPIO_InitStructure);  
  
// (168Mhz/2)/8400 = 10KHz(1ms)  
TIM_TimeBaseStructure.TIM_Prescaler = 8400-1;  
TIM_TimeBaseStructure.TIM_Period = 10000-1;  
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;  
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
```

실습 2 : 타이머를 이용한 디지털 시계

- 구동 프로그램
 - main.c 코드 작성

```
// 타이머 3 인터럽트의 우선순위를 설정
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
NVIC_InitStructure.NVIC IRQChannel = TIM3_IRQn;
NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 0x01;
NVIC_InitStructure.NVIC IRQChannelSubPriority = 0x01;
NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
TIM_Cmd(TIM3, ENABLE);
while(1)
{
}
```

실습 2 : 타이머를 이용한 디지털 시계

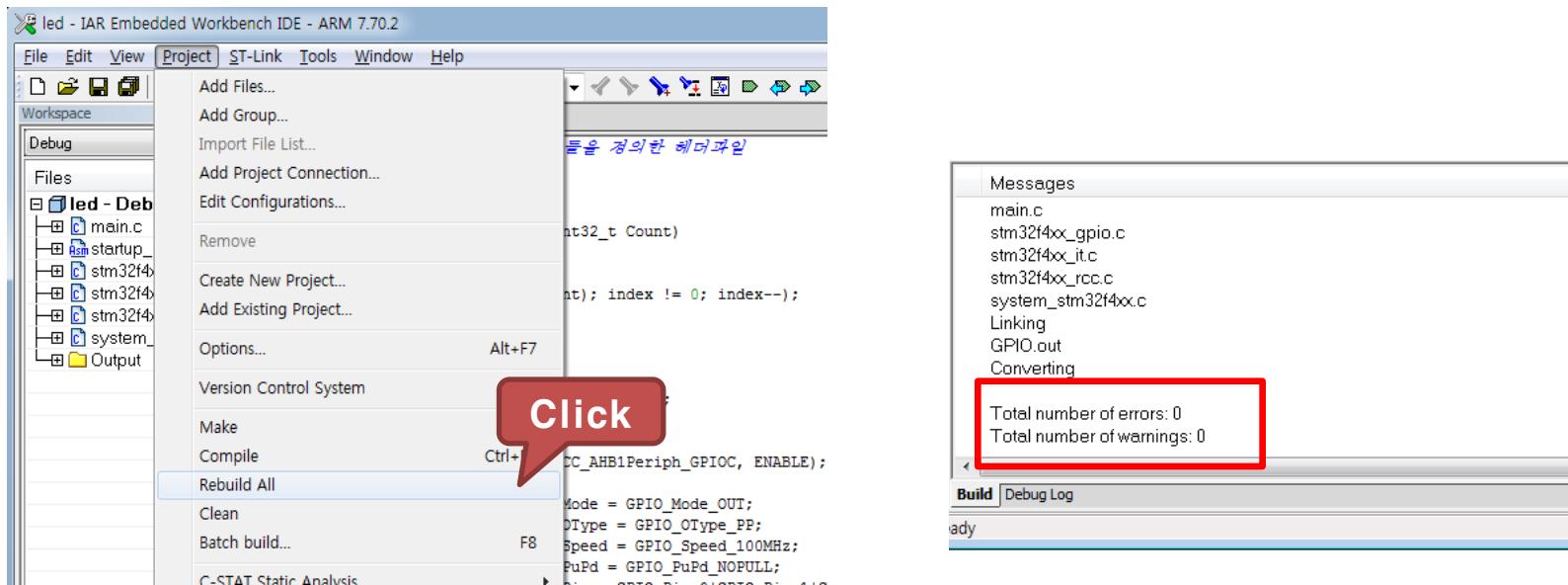
- 구동 프로그램
 - main.c 코드 작성

```
void TIM3_IRQHandler(void) {
    if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) {
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
        // 포트 C에 FND Table 값을 출력
        GPIO_Write(GPIOC, FND_DATA_TBL[time_s]);
        time_s++;                      // time_s 변수는 9까지만 증가
        if(time_s>=10) time_s=0;       // 10되면 0으로 초기화
    }
}
```

실습 2 : 타이머를 이용한 디지털 시계

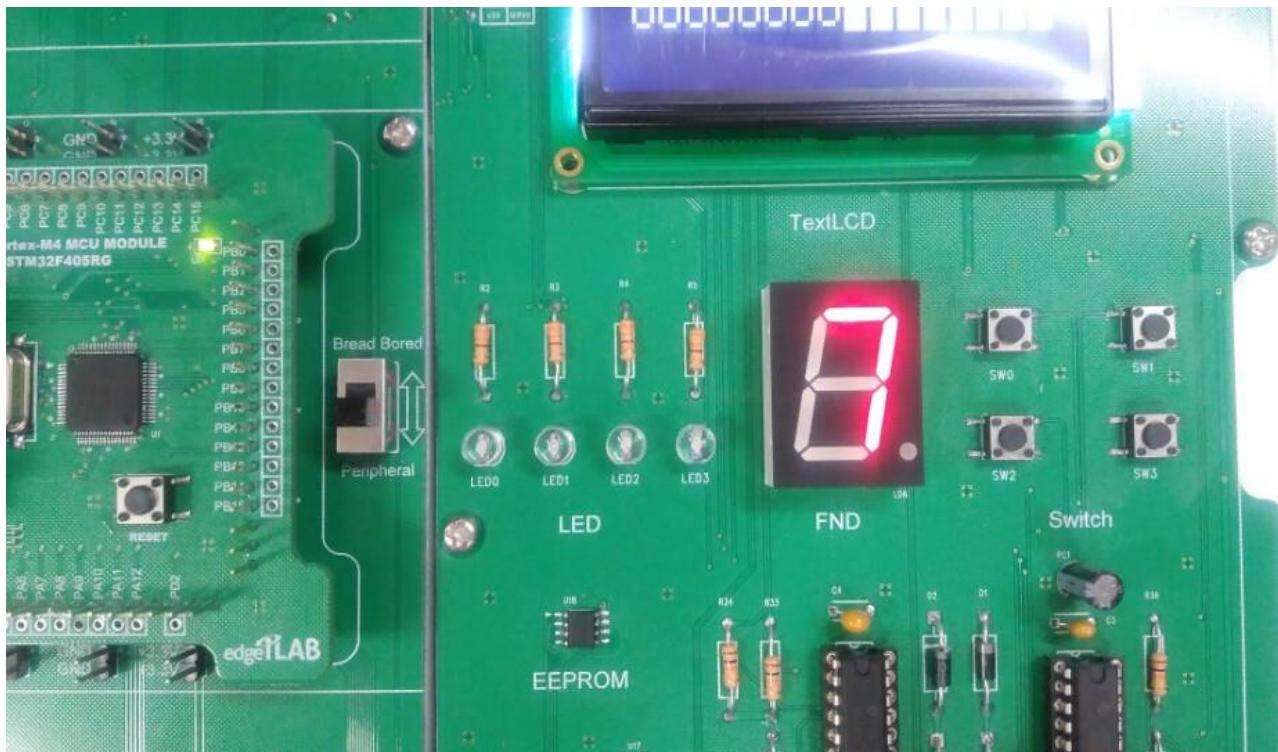
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 timerClock 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 2 : 타이머를 이용한 디지털 시계

- 실행 결과
 - 1초마다 FND의 숫자가 증가



타이머와 PWM

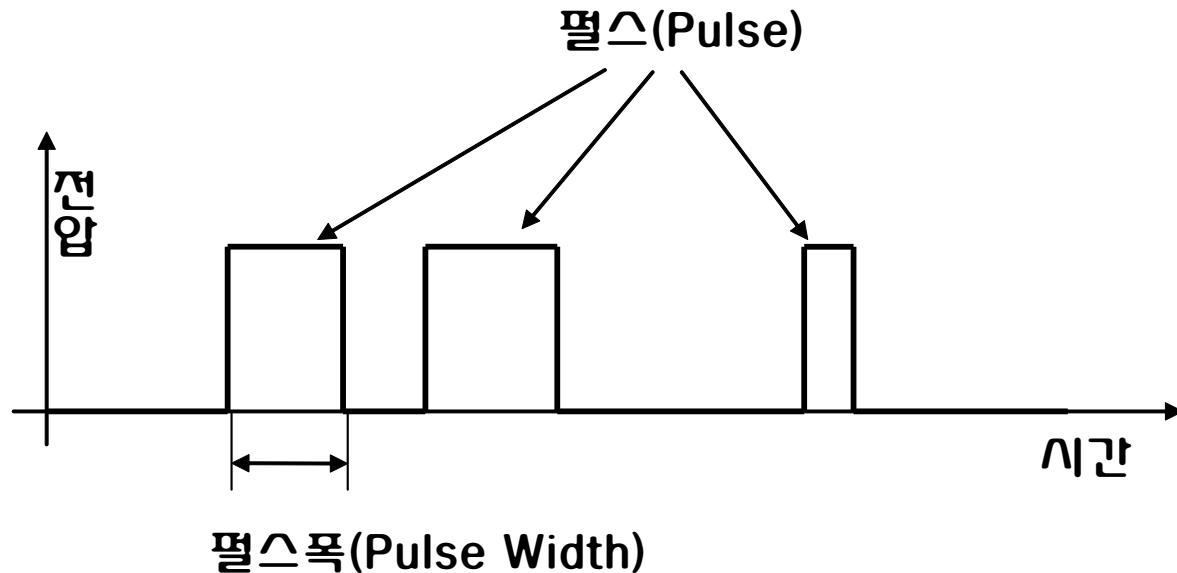
1. **PWM(Pulse Width Modulation)**
2. **타이머의 동작모드**
3. **PWM으로 LED 밝기 조절하기**
4. **PWM으로 버저 울리기**



엣지아이랩

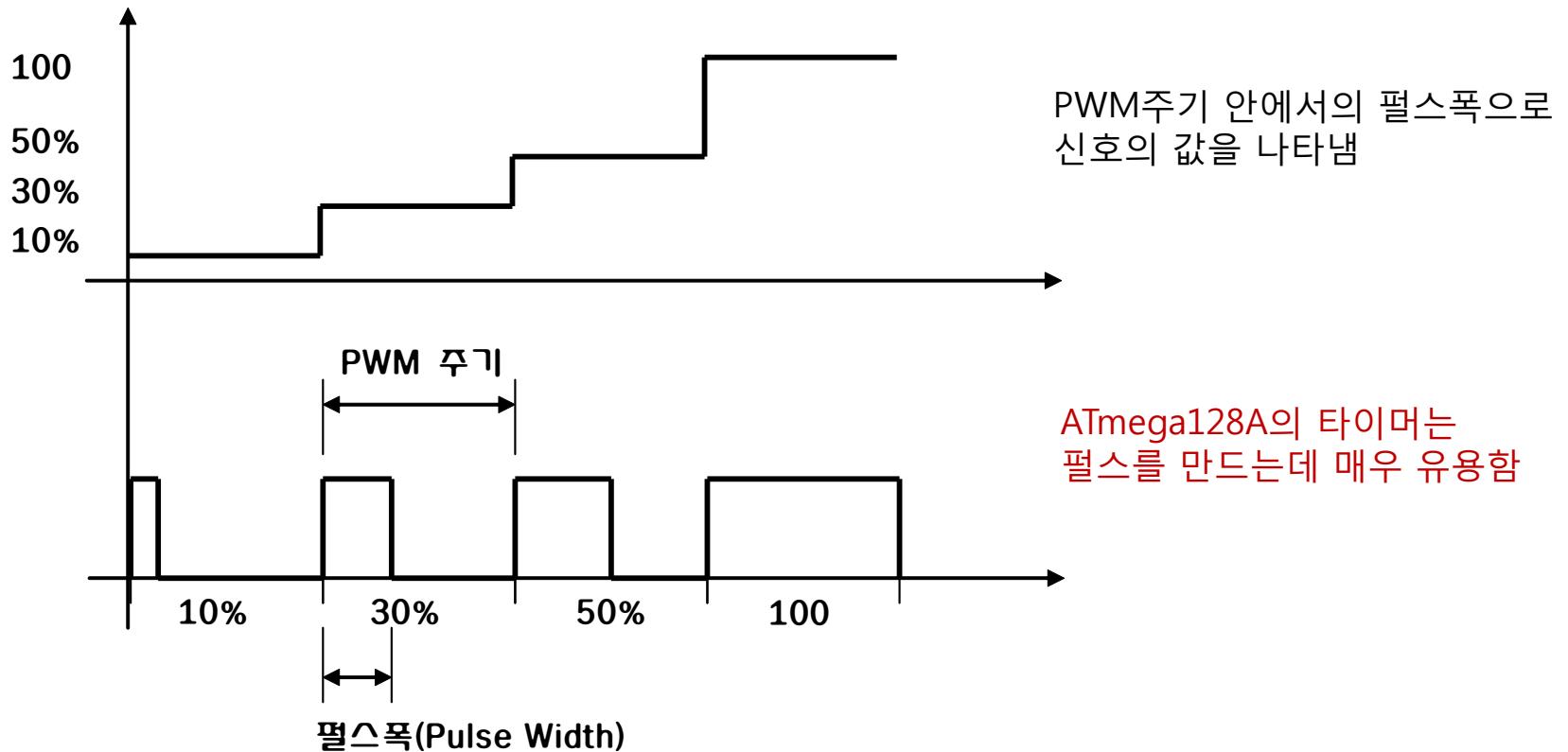
PWM(Pulse Width Modulation)

- 펄스(Pulse)와 펄스폭(Pulse Width)
 - 펄스 : 짧은 시간동안 생기는 진동 현상
 - 펄스폭 : 하나의 펄스가 가지는 폭



PWM(Pulse Width Modulation)

- PWM(펄스폭 변조)
 - 펄스 폭을 전송하고자 하는 신호에 따라 변화시키는 변조 방식
 - 모터 제어나 전압제어 등에 널리 사용



타이머/카운터의 동작모드

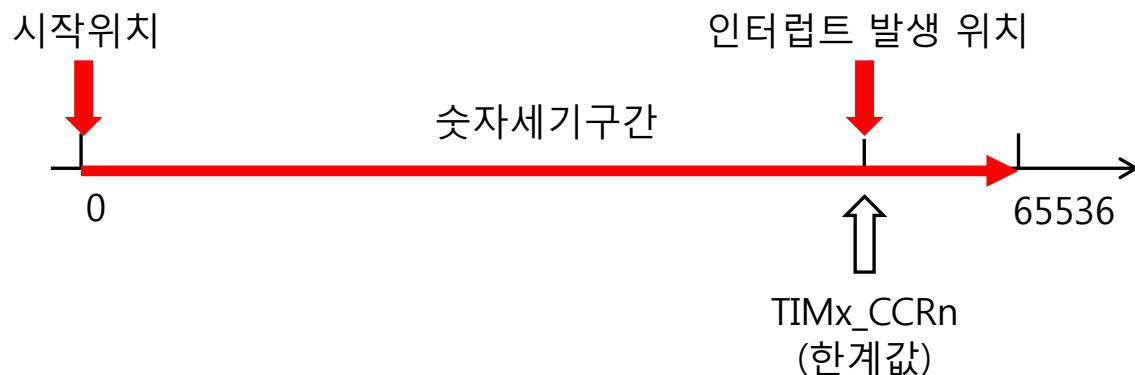
- Normal Mode(일반 동작모드)
 - 업/다운/중앙정렬 카운터로 동작
 - 업데이트 이벤트는 TIMx_EGR레지스터의 UG비트가 설정되어 있어야 발생
 - 업카운팅 모드는 0에서 auto-reload value까지 카운트하여, 0이 되는 순간 오버플로우가 발생
 - 다운카운팅모드는 TIMx_ARR 레지스터값부터 0까지 카운트하며, 다시 카운터가 TIMx_ARR 값으로 설정되는 순간 언더플로우가 발생
 - 중앙정렬모드는 오버/언더플로우 이벤트 2개 다 발생



타이머/카운터의 동작모드

- PWM Mode

- TIMx_CCRx레지스터로 정의한 듀티비와 TIMx_ARR 레지스터에 의해 정의된 주파수 신호를 발생
- TIMx_CCMRx레지스터의 OCxM비트들을 설정하여 각 채널을 독립적으로 선택 가능
- 반드시 preload 레지스터(TIMx-CCMRx의 OCxPE비트)와 auto-preload 레지스터 (TIMx_CR1의 ARPE비트)를 Enable해야 함
- Preload 레지스터(TIMx_ARR레지스터의 값을 복사하고 있음)의 값은 오직 update 이벤트가 발생했을 때만 shadow레지스터로 전송
- 카운터를 시작하기 전에는 반드시 TIMx_EGR의 UG비트를 설정하여 모든 레지스터를 초기화
- 출력은 TIMx_CCER레지스터의 CCxE비트로 enable됨
- PWM모드에서 TIMx_CNT와 TIMx_CCRx레지스터는 항상 서로의 값을 비교하게 되며 Output compare 모드가 PWM모드일 경우 OCREF 신호가 동작하게 됨



실습 1 : PWM으로 LED 밝기 조절 하기

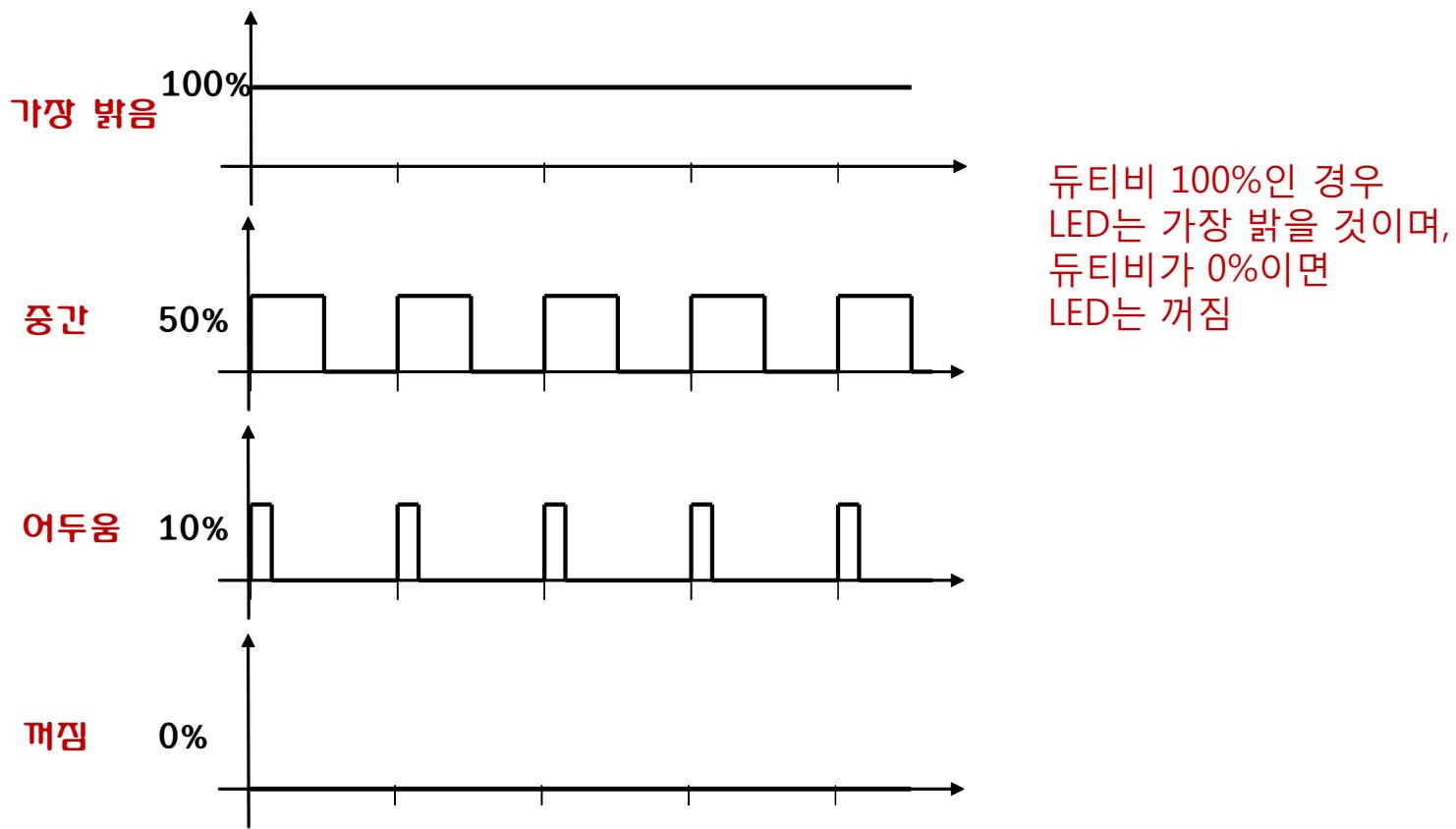
- 실습 개요
 - 타이머 2의 PWM 동작 모드를 이용하여 LED의 밝기를 조절하기
 - PWM 동작 모드에서 TIMx_CHx 핀을 통해 PWM신호를 만들어 출력함으로써, LED의 밝기를 조절하도록 함
 - 밝기는 PWM 신호의 듀티비(Pulse Duty)에 의해 좌우됨
- 실습 목표
 - 타이머2의 PWM 기능 동작원리 이해
 - 타이머2의 PWM 모드 제어 방법의 습득(관련 레지스터 이해)
 - PWM 신호 출력 제어 방법 습득

실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램 : 사전 지식

- PWM 신호에 의한 LED의 밝기 조절 방법

- 출력 비교용 레지스터(CCR)값을 조절하여 TIMx_CHx로 출력되는 PWM 신호의 듀티비를 원하는 대로 변경

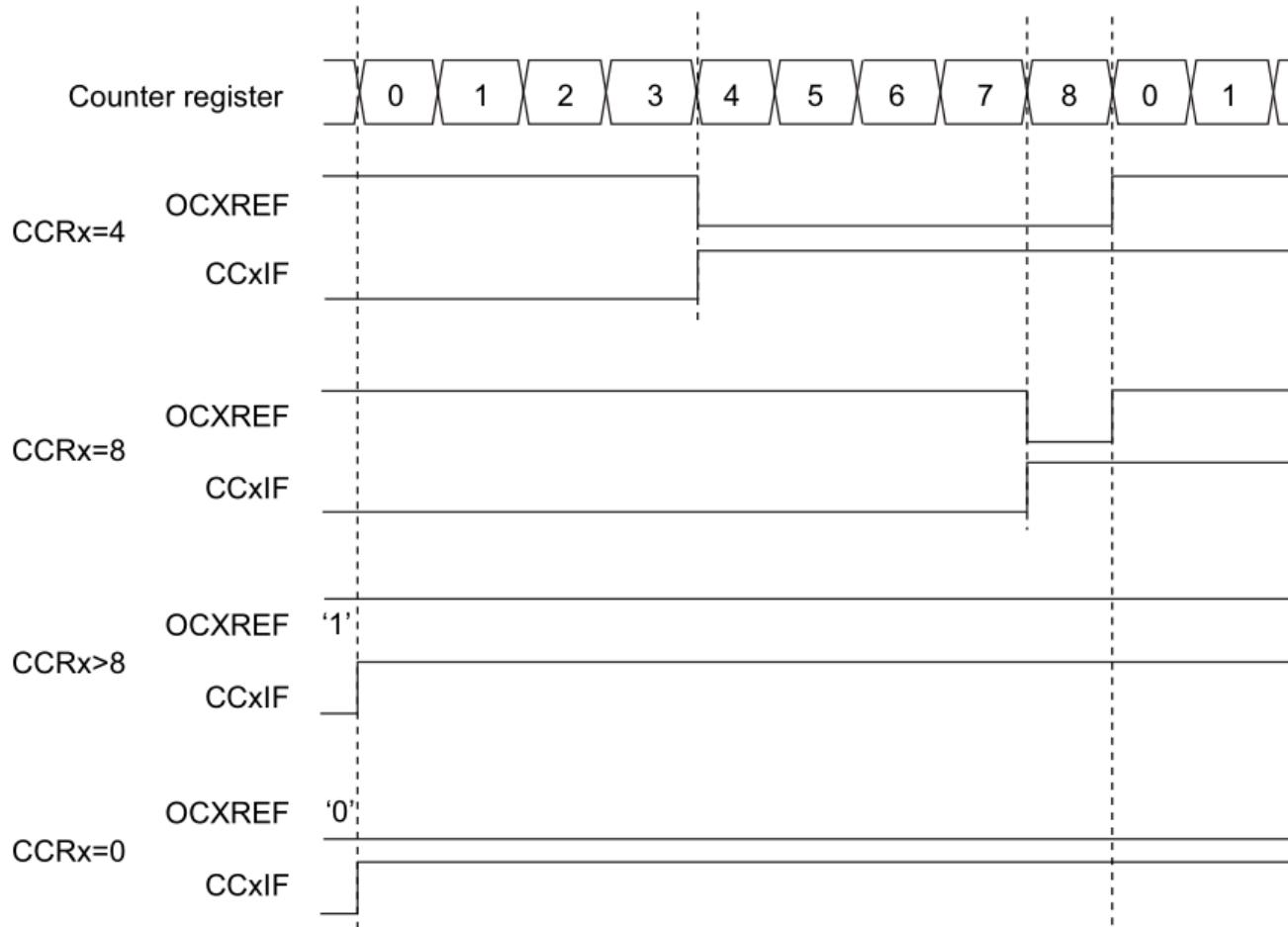


실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램 : PWM 동작 모드 설정
 - 사용할 타이머/카운터 결정
 - 여기서는 타이머/카운터 2를 사용
 - 동작 모드 결정
 - 여기서는 PWM 모드 사용
 - 카운트 방향을 업카운트(0 부터 TIMx-ARR까지)로 선택
 - TIM2_CR1레지스터의 DIR을 0으로 클리어
 - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
 - 여기서는 클럭(84MHz)을 사용
 - 타이머 2의 클럭은 메인 클럭(168Mhz)의 $\frac{1}{2}$ 을 사용
 - 샘플링 클럭의 분주비는 1:1로 설정
 - 타이머 클럭 프리스케일러는 83으로 설정(TIM2_CR1/CKD:00, TIM2_PSC: 83)
 - 84MHz 클럭의 1분주비와 83 프리스케일러로 나누어 지므로 1MHz(1us)로 동작
 - PWM 주파수 결정
 - 10KHz를 만들기 위해 0.1ms을 타이머 주기로 결정
 - TIM2_ARR레지스터의 값을 100으로 설정
 - 듀티비는 TIM2_CCR1 에 1~100 까지 입력하여 변화(TIM2_CCR1 최대값은 TIM2_ARR값 까지)

실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램 : 사전 지식
 - CCR_x값의 변화에 따른 출력 신호 변화(TIM_x_ARR = 8)



실습 1 : PWM으로 LED 밝기 조절 하기

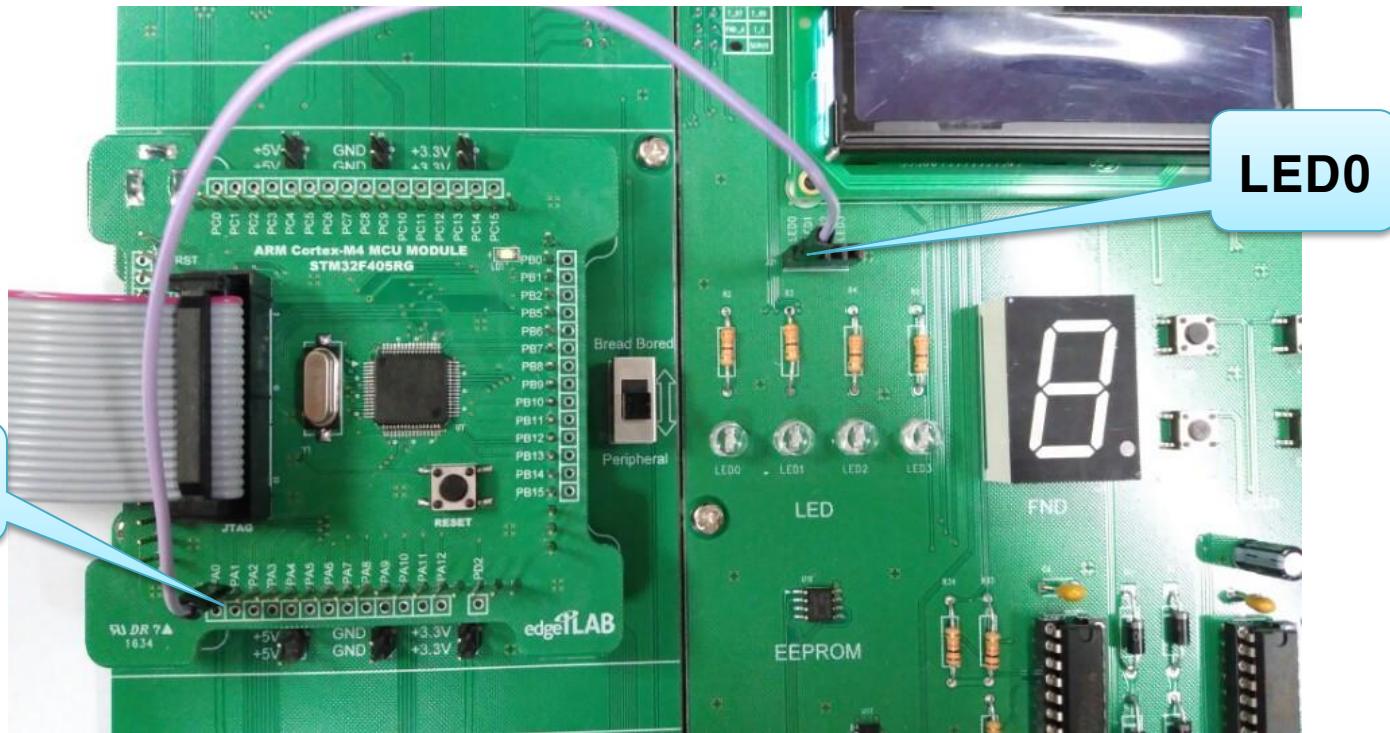
- TIM 라이브러리
 - STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
 - TIM_OCInitTypeDef 구조체

TIM_OCInitTypeDef구조체	설 명
uint16_t TIM_OCMode	Output compare 모드를 결정한다. (TIMx_CCMR1,2의 OCnM비트)
uint16_t TIM_OutputState	Capture/Compare의 입/출력을 선택한다. (TIMx_CCER의 CCnE비트)
uint16_t TIM_OutputNState	Capture/Compare의 입/출력을 선택한다. (TIMx_CCER의 CCnNE비트 : 타이머 1, 8에서만 사용)
uint16_t TIM_Pulse	TIMx_CCRn레지스터(Capture/Compare n Value) 값을 결정한다. (선택한 채널에 의해 레지스터결정.)
uint16_t TIM_OCPolarity	PWM, OCM 또는 OPM Channel의 출력극성을 결정한다. 이 비트가 0이어야 설정한 모드대로 동작한다. (TIMx_CCER의 CCnP비트)
uint16_t TIM_OCNPolarity	PWM, OCM 또는 OPM Channel의 출력극성을 결정한다. (TIMx_CCER의 CCnNP비트 : 타이머 1, 8에서만 사용)
uint16_t TIM_OCIdleState	Output Idle state(OCnN output)를 설정한다. (TIMx_CR2의 OISn, 타이머 1과 8 만 사용)
uint16_t TIM_OCNIdleState	Output Idle state(OCn output)를 설정한다. (TIMx_CR2의 OISnN, 타이머 1과 8 만 사용)

실습 1 : PWM으로 LED 밝기 조절 하기

- 실습 준비

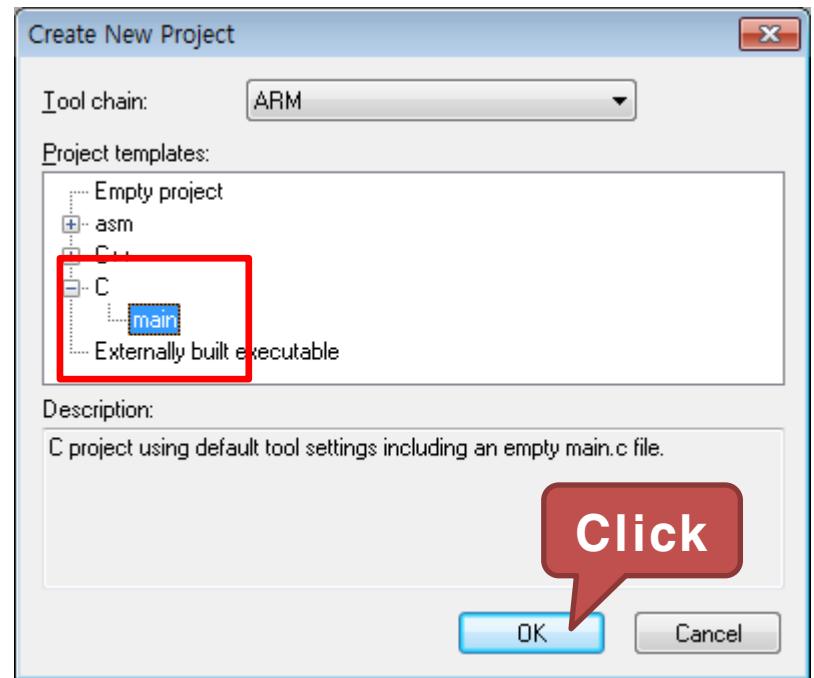
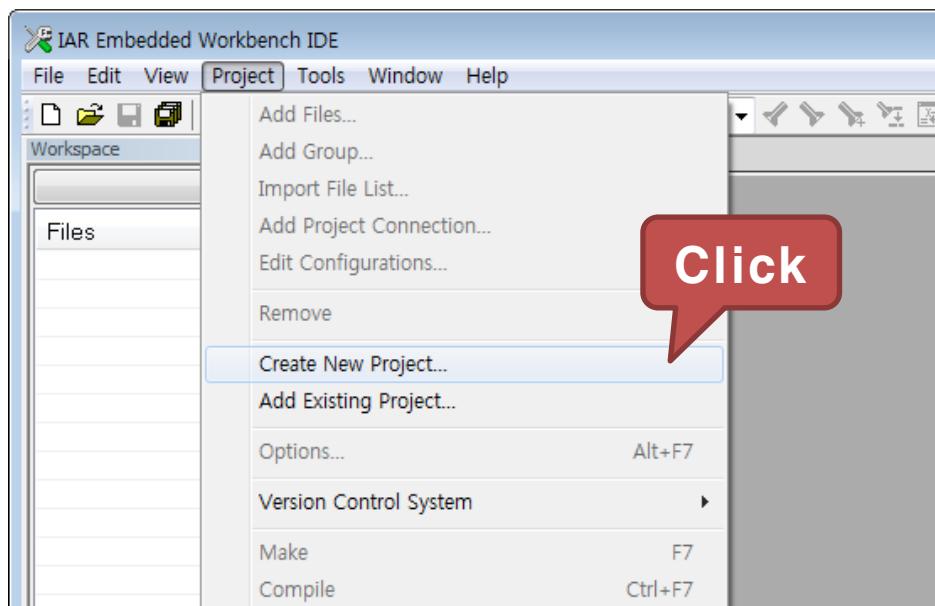
- 사용보드를 분리하여 다음 그림과 같이 연결
 - Edge-MCU보드의 PA0 → Edge-Peri 보드의 LED0(LED 상단)



실습 1 : PWM으로 LED 밝기 조절하기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch7” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “pwmLed”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1 : PWM으로 LED 밝기 조절 하기

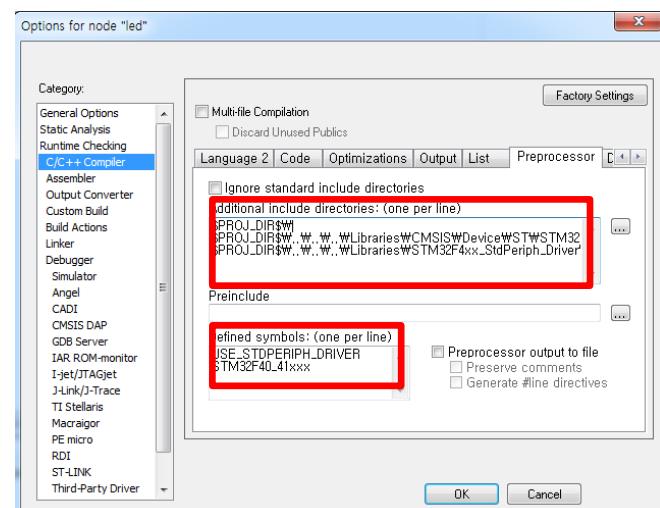
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch7\pwmLed	system_stm32f4xx.c
Cortex_Example\Projects\ch7\pwmLed	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c

실습 1 : PWM으로 LED 밝기 조절 하기

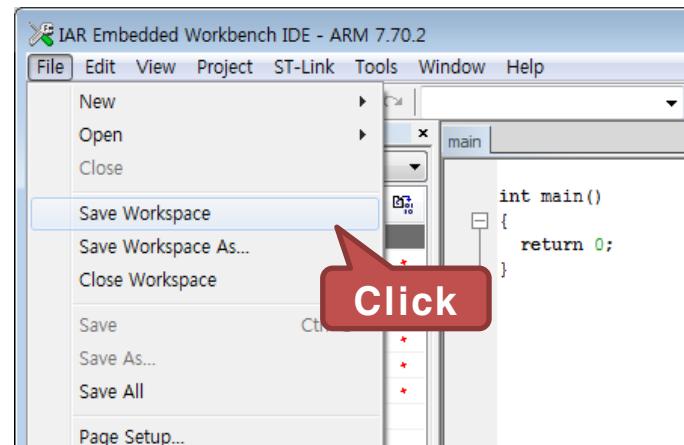
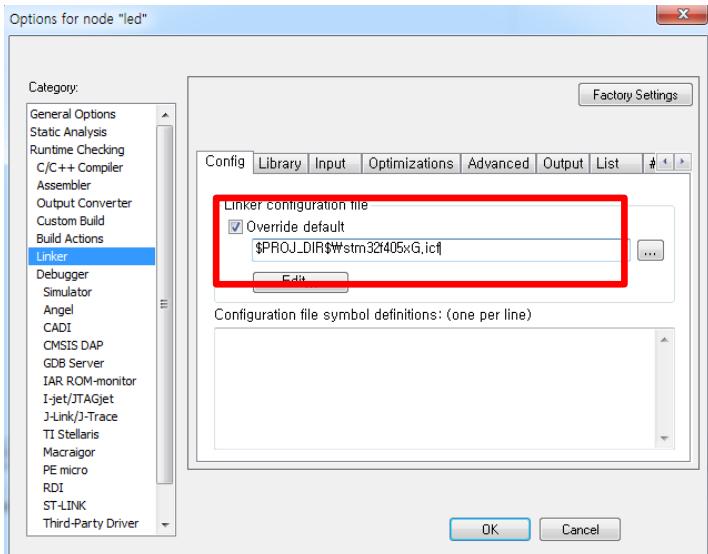
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1 : PWM으로 LED 밝기 조절하기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램
 - main.c 코드 작성

```
#include "stm32f4xx.h"
static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}
unsigned int CCR1_Val = 0;                         // TIM2_CCR1 레지스터 값 결정

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCIInitTypeDef  TIM_OCIInitStructure;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
```

실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;           // TIM2_CH1  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource0, GPIO_AF_TIM2);  
  
// (168Mhz/2)/84 = 1MHz(1us)  
TIM_TimeBaseStructure.TIM_Prescaler = 84-1;  
TIM_TimeBaseStructure.TIM_Period = 100-1;  
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;  
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
```

실습 1 : PWM으로 LED 밝기 조절 하기

- 구동 프로그램
 - main.c 코드 작성

```
//...  
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;  
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;  
TIM_OCInitStructure.TIM_Pulse = CCR1_Val;  
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;  
TIM_OC1Init(TIM2, &TIM_OCInitStructure);  
  
// 타이머2를 동작  
TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Enable);  
TIM_ARRPreloadConfig(TIM2, ENABLE);  
TIM_Cmd(TIM2, ENABLE);
```

실습 1 : PWM으로 LED 밝기 조절 하기

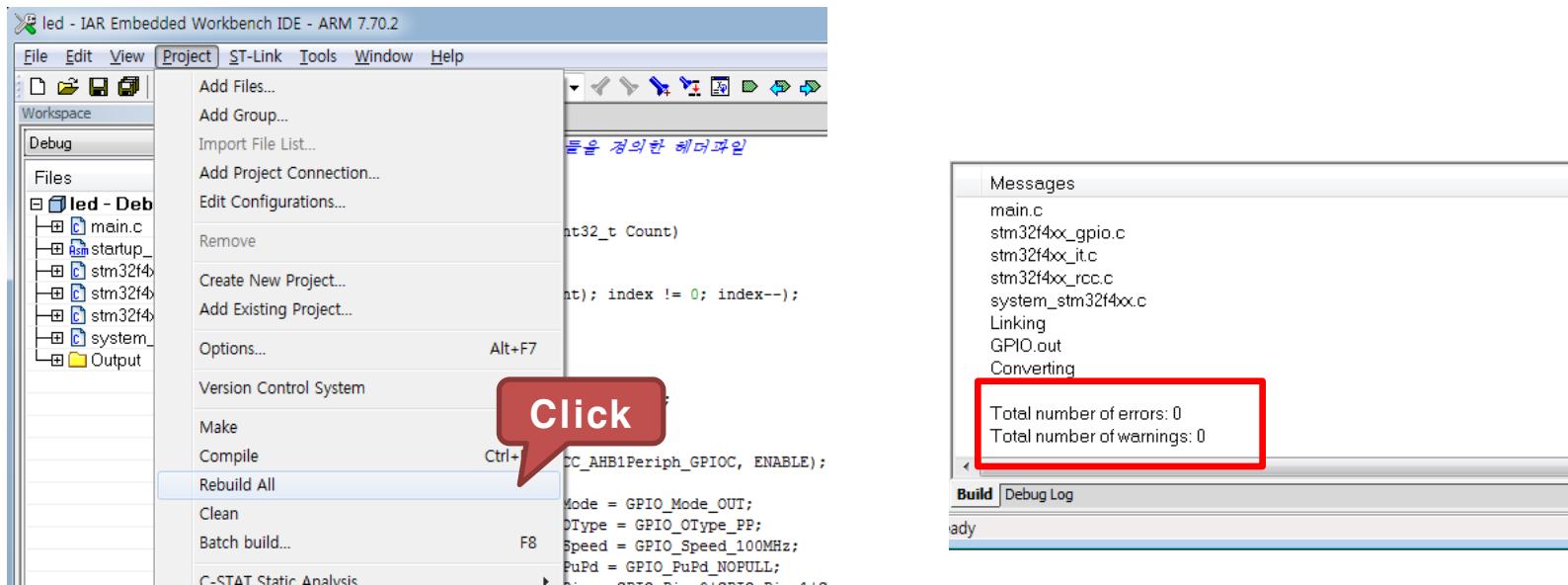
- 구동 프로그램
 - main.c 코드 작성

```
//...
while(1) {
    TIM_Cmd(TIM2, DISABLE);
    TIM_SetCompare1(TIM2, CCR1_Val);
    TIM_Cmd(TIM2, ENABLE);
    CCR1_Val++;
    if(CCR1_Val>100) CCR1_Val = 0;
    Delay(10);
}
```

실습 1 : PWM으로 LED 밝기 조절하기

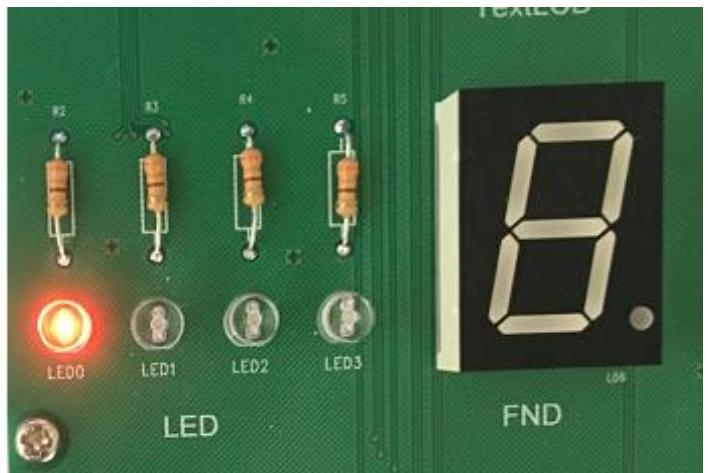
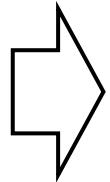
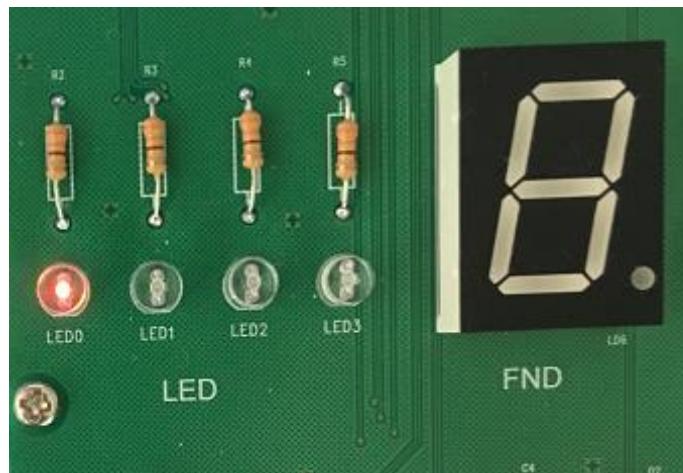
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 pwmLed 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 1 : PWM으로 LED 밝기 조절 하기

- 실행 결과
 - LED가 어두워졌다가 밝아짐



실습 2 : PWM으로 PIEZO 울리기

- 실습 개요

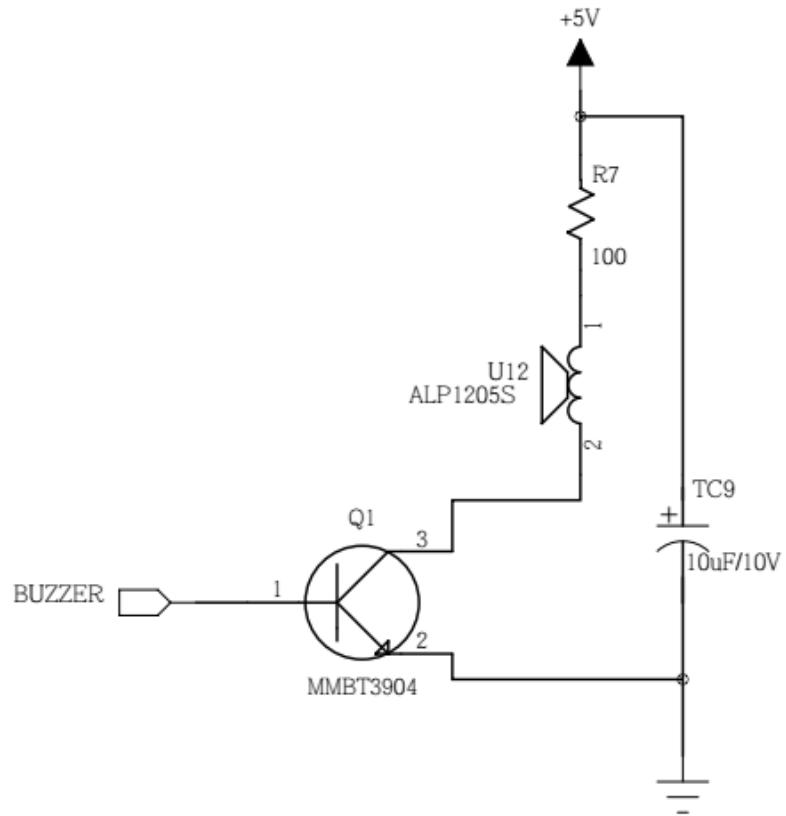
- 타이머 2의 PWM 동작 모드를 원하는 주파수의 신호를 만들고, 이를 Piezo에 입력하여 여러가지 소리를 내도록 설계
- PWM 동작 모드에서 TIMx_CHx 핀을 통해 PWM신호를 만들어 출력하고, PWM의 주파수를 조절하도록 함

- 실습 목표

- 타이머2의 PWM 기능 동작원리 이해
- 타이머2의 PWM 모드 제어 방법의 습득(관련 레지스터 이해)
- Piezo의 동작원리 이해

실습 2 : PWM으로 PIEZO 울리기

- 사용 모듈 : Audio 모듈 회로
 - 압전 Piezo 관련 회로

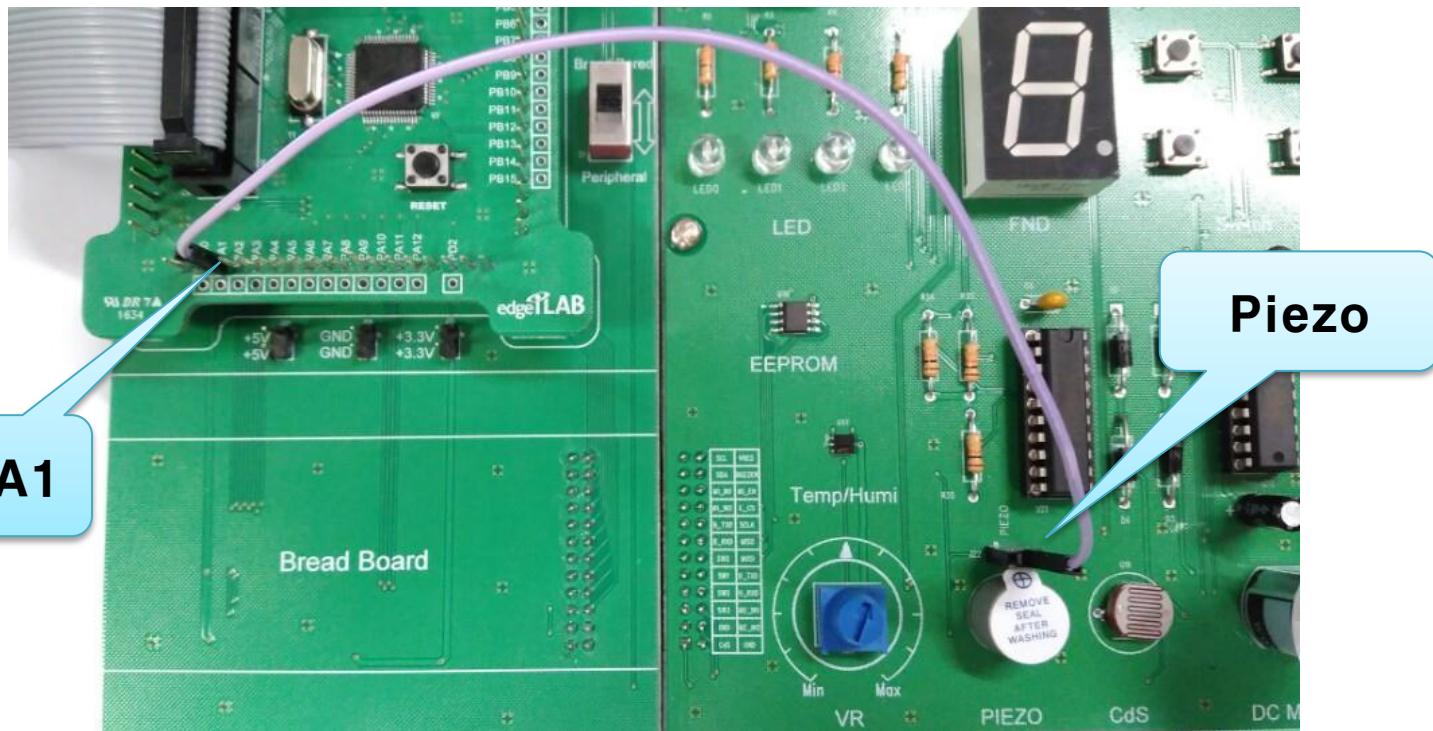


실습 2 : PWM으로 PIEZO 울리기

- 구동 프로그램 : PWM 동작 모드 설정
 - 사용할 타이머/카운터 결정
 - 여기서는 타이머/카운터 2를 사용
 - 동작 모드 결정
 - 여기서는 PWM 모드 사용
 - 카운트 방향을 업카운트(0 부터 TIMx-ARR까지)로 선택
 - TIM2_CR1레지스터의 DIR을 0으로 클리어
 - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
 - 여기서는 클럭(84MHz)을 사용
 - 타이머 2의 클럭은 메인 클럭(168Mhz)의 $\frac{1}{2}$ 을 사용
 - 샘플링 클럭의 분주비는 1:1로 설정
 - 타이머 클럭 프리스케일러는 0으로 설정(TIM2_CR1/CKD:00, TIM2_PSC: 0)
 - 84MHz 클럭의 1분주비와 0 프리스케일러로 나누어 지므로 84MHz로 동작
 - PWM 주파수 결정
 - 원하는 주파수를 만들기 위해 TIM2_ARR레지스터의 값을 84000000/pwmfreq 으로 설정
 - 듀티비는 TIM2_CCR1 에 84000000/pwmfreq/2 값을 입력하여 듀티비 50%로 만듦

실습 2 : PWM으로 PIEZO 울리기

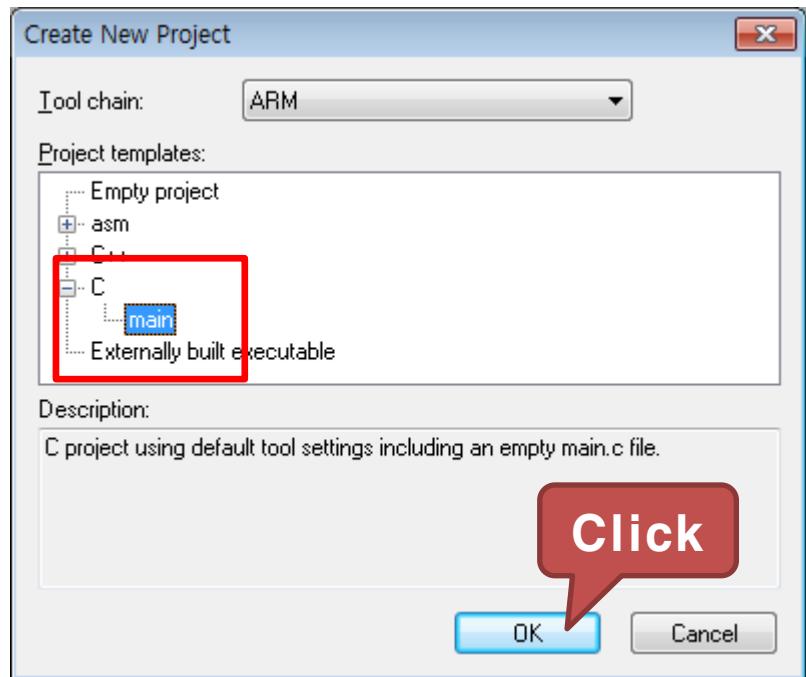
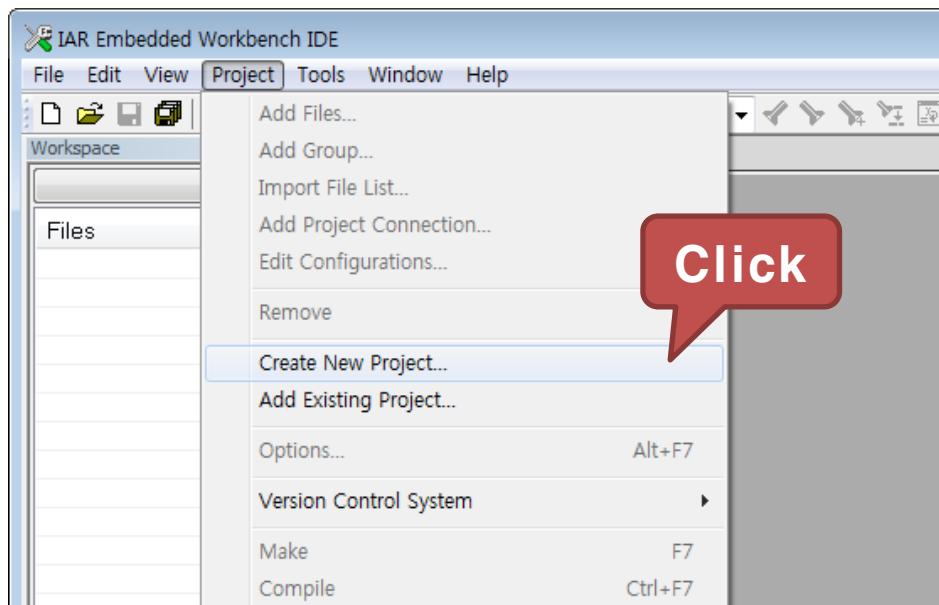
- 실습 준비
 - 사용보드를 분리하여 다음 그림과 같이 연결
 - Edge-MCU보드의 PA0 → Edge-Peri 보드의 PIEZO(PIEZO상단우측)



실습 2 : PWM으로 PIEZO 울리기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch7” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “pwmPiezo”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : PWM으로 PIEZO 울리기

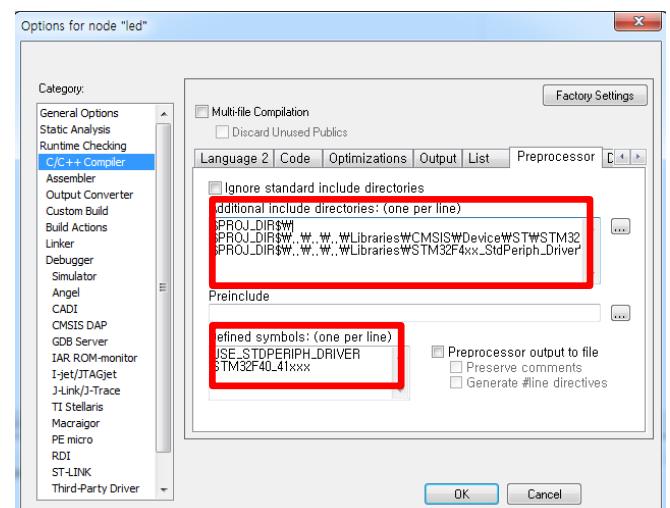
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch7\pwmPiezo	system_stm32f4xx.c
Cortex_Example\Projects\ch7\pwmPiezo	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c

실습 2 : PWM으로 PIEZO 울리기

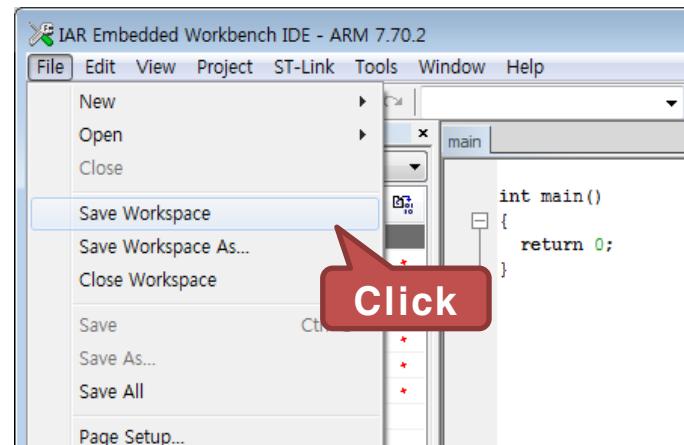
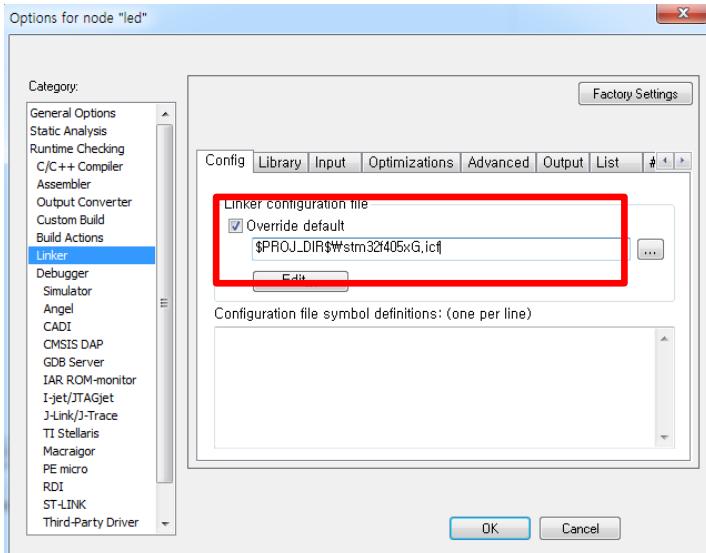
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : PWM으로 PIEZO 울리기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : PWM으로 PIEZO 울리기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"
static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}
// 피아노 음계에 해당하는 PWM 주파수
unsigned int DoReMi[8] = {523,587, 659, 698, 783,880, 987, 1046};
unsigned long pwmfreq;   // 실제 주파수 계산에 사용되는 변수
unsigned char piano=0;  // Piezo로 출력할 음계를 저장하는 변수

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCIInitTypeDef  TIM_OCIInitStructure;
```

실습 2 : PWM으로 PIEZO 울리기

- 구동 프로그램

- main.c 코드 작성

```
//...  
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);  
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;           // TIM2_CH2  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource1, GPIO_AF_TIM2);  
  
TIM_TimeBaseStructure.TIM_Prescaler = 0; // (168Mhz/2)/1 = 84MHz  
TIM_TimeBaseStructure.TIM_Period = 0;  
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;  
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
```

실습 2 : PWM으로 PIEZO 울리기

- 구동 프로그램
 - main.c 코드 작성

```
//...  
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;  
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;  
TIM_OCInitStructure.TIM_Pulse = 0;  
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;  
TIM_OC2Init(TIM2, &TIM_OCInitStructure);  
  
// 타이머2를 동작  
TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Enable);  
TIM_ARRPreloadConfig(TIM2, ENABLE);  
  
TIM_Cmd(TIM2, ENABLE);
```

실습 2 : PWM으로 PIEZO 울리기

- 구동 프로그램

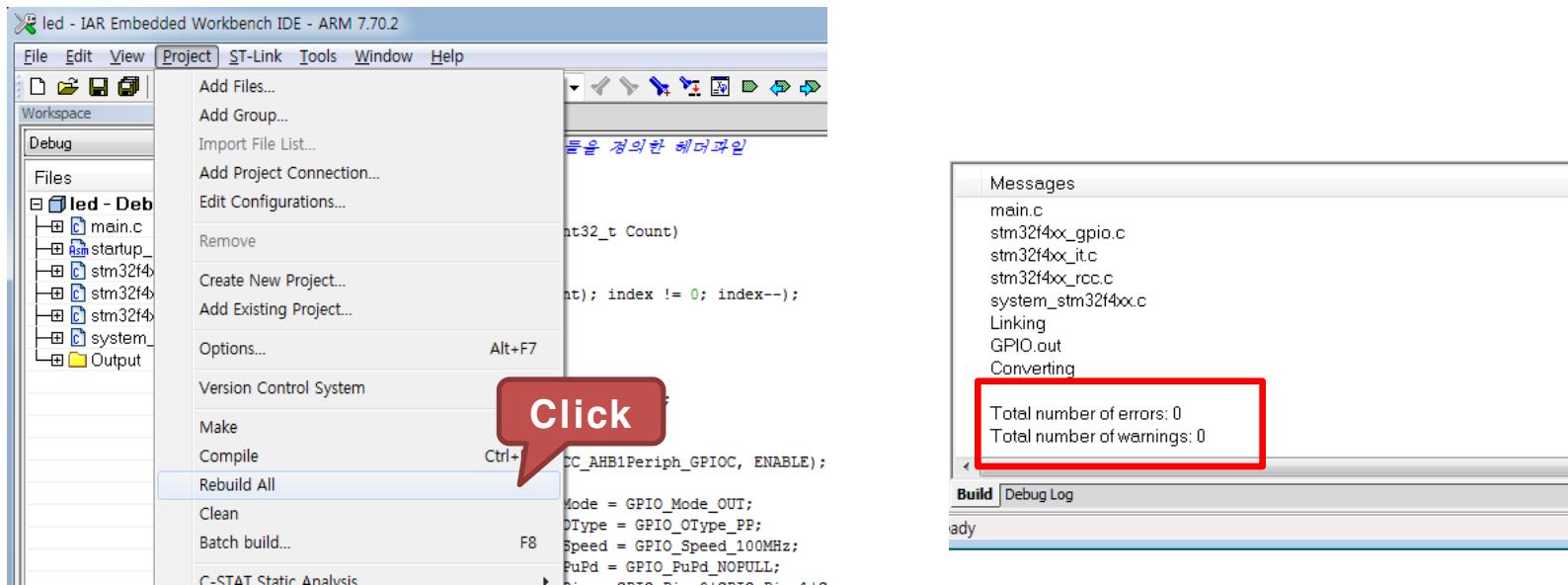
- main.c 코드 작성

```
//...
while(1){
    TIM_Cmd(TIM2, DISABLE);
    pwmfreq = 84000000/DoReMi[piano]; // 음계에 맞는 주파수를 계산
    // 84MHz / (84000000/pwmFreq) = pwmfreq = pwmfreq Hz
    TIM_TimeBaseStructure.TIM_Period = pwmfreq - 1;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    TIM_SetCompare2(TIM2,pwmfreq/2);      // 50% 듀티비
    TIM_Cmd(TIM2, ENABLE);
    piano++;                            // piano 변수 1증가
    if(8 < piano) piano = 0;           // piano가 9가 되면 초기화
    Delay(1000);
}
}
```

실습 2 : PWM으로 PIEZO 울리기

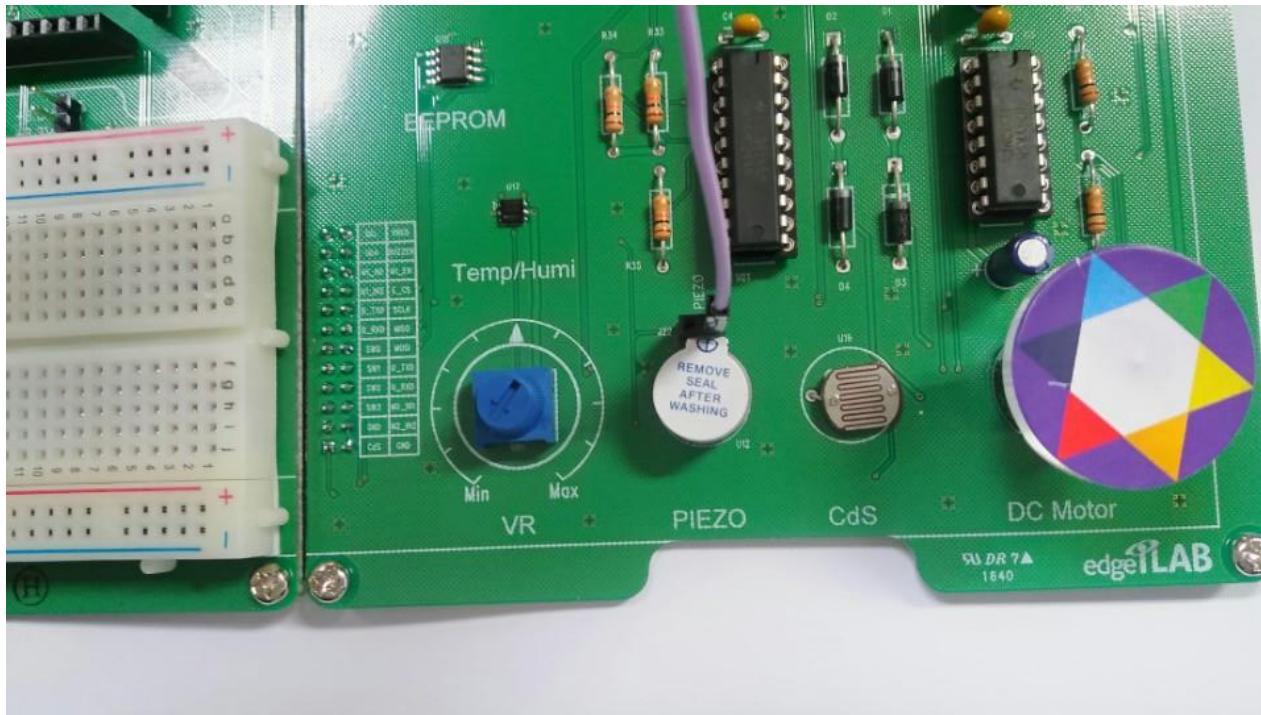
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 pwmPiezo 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 2 : PWM으로 PIEZO 울리기

- 실행 결과
 - Piezo에서 도레미파솔라시도 음계가 출력





UART

- **UART와 RS232 개요**
- **STM32F405의 USART 포트**
- **UART로 Hello 보내기**
- **UART로 PC와 데이터 주고받기**



엣지아이랩

UART와 RS232 개요

- UART(Universal Asynchronous Receiver/Transmitter)
 - 시리얼 기반의 통신 방식으로 일반적으로 RS232 프로토콜을 통해 원격지와 통신을 지원하는 방식
 - UART는 컴퓨터에게 RS-232C DTE 인터페이스를 제공함으로써, 모뎀이나 기타 다른 직렬장치들과 통신하거나 데이터를 주고받을 수 있게 함
 - UART의 동작
 - 직렬 비트 스트림을 컴퓨터가 처리할 수 있도록 바이트로 변환
 - 패리티 비트 처리
 - 시작 비트와 정지 비트 처리
 - 키보드나 마우스로부터 들어오는 인터럽트 처리

UART와 RS232 개요

- RS-232C
 - 직렬전송을 위한 규격
 - 1969년 미국의 EIA (Electric Industries Association)에 의해서 정해진 표준 인터페이스
 - "직렬 2진 데이터의 교환을 하는 데이터 터미널 장비(DTE)와 데이터 통신장비(DCE)간의 인터페이스의 제반을 규정하는 것"
 - RS-232C의 동작
 - 병렬을 직렬로 직렬을 병렬로 바꾸어 주는 작업
 - 스타트 비트와 스톱비트 포함하여 10비트를 1바이트로 보냄
 - RXD, TXD 라인을 통해 신호를 송수신
 - RS232 Transceiver를 통해 전송 전압을 끌어 올려 보다 먼 거리까지 전송

UART와 RS232 개요

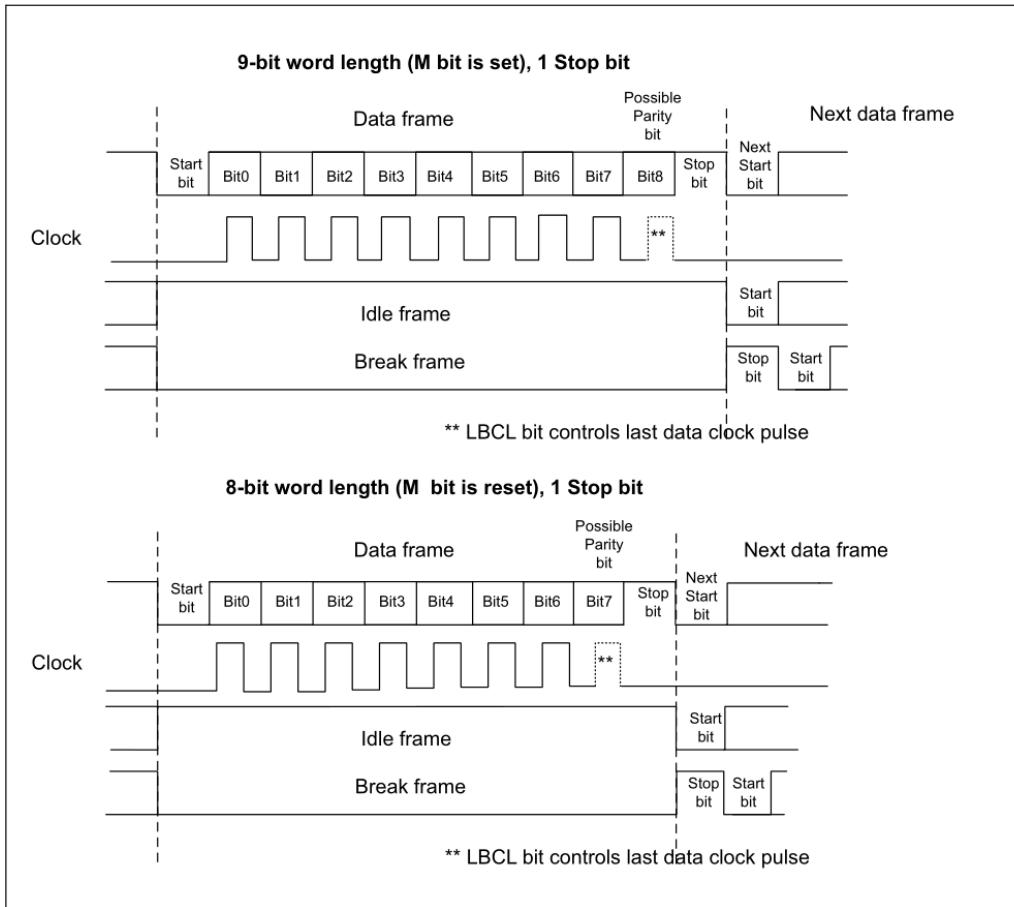
- RS-232C를 이용한 비동기식 전송시 규약
 - 통신속도
 - 시간당 데이터를 전송할 수 있는 양
 - baud rate : 1초당 전송되는 변조된 신호의 수
 - 스탶비트
 - 데이터의 시작과 끝을 알리는 스탶트와 스탶비트를 사용
 - 전송을 시작할 경우 1을 내보내고 8비트를 전송한 후 스탶비트를 전송
 - 스탶트 비트는 고정/ 스탶비트는 1과 1.5, 2비트중 하나를 선택
 - 패리티
 - 오류 검출을 위해 사용
 - 패리티의 종류는 짹수 및 훌수 방식과 사용하지 않는 경우가 있다
 - 데이터 길이가 7인 경우에 8번째 비트를 패리티 비트로 이용
 - 자료 길이
 - 하나의 데이터를 전송하는데 필요한 데이터 길이(비트 수)
 - 보통 7과 8 비트

STM32F405의 USART 포트

- STM32F405의 직렬통신 포트
 - 직렬통신포트 USART(Universal Synchronous and Asynchronous Receive and Transmitter) 6개 내장
 - USART1~3, USART6
 - USART4, 5
 - 완전 이중방식(Full-Duplex)
 - 동기 및 비동기 전송 가능
 - 멀티 프로세서 통신 모드로 동작 가능
 - 높은 정밀도의 보레이트 발생기 내장
 - 인터럽트
 - 송신 완료(TX Complete)
 - 송신 데이터 레지스터 준비완료(TX Data Register Empty)
 - 수신완료(RX Complete)
 - 시리얼 통신 외에도 Smartcard 제어, IrDA 통신, LIN(Local Interconnection Network) 통신 지원

STM32F405의 USART 포트

- STM32F405 USART 데이터 프레임 포맷
 - 최소 7비트 최대 13비트로 구성
 - (1 비트의 스타트 비트) + (8,9 비트의 데이터 비트)
 - + (0, 1 비트의 패리티비트) + (0.5, 1, 1.5, 2 비트의 스톱비트) 프레임



STM32F405의 USART 포트

- STM32F405 USART 데이터 프레임 포맷
 - 스타트 비트
 - 1비트로 이루어 졌으며 항상 0레벨, 송신시에 자동적으로 생성
 - 데이터 비트
 - 8, 9비트 가능
 - 패리티 비트
 - 패리티를 사용하지 않을 수도 있고 사용하는 경우 홀수 혹은 짝수 패리티 1비트를 사용
 - 스톱 비트
 - 0.5, 1, 1.5, 2개의 비트가 가능하며 항상 1레벨, 송신시에 자동적으로 생성

STM32F405의 USART 포트

- STM32F405 USART 레지스터
 - USART_SR(Status Register)
 - 상태 레지스터
 - USART_DR(Data Register)
 - 데이터 레지스터
 - USART_BRR (Baud Rate Register)
 - 보우레이트 레지스터
 - USART_CR1(Control Register1)
 - 제어 레지스터 1
 - USART_CR2(Control Register 2)
 - 제어 레지스터 2
 - USART_CR3(Control Register 3)
 - 제어 레지스터 3
 - USART_GTPR(Guard time and prescaler register)
 - Guard 타임 과 프리스케일러 설정 레지스터

STM32F405 USART 레지스터

- USART_SR (Status register)

- USART 상태 레지스터
- CTS (CTS flag)

- CTS입력이 toggles될 때, 하드웨어에 의해 1로 설정
- CTSE비트가 1로 설정되면, 이는 소프트웨어에 의해(0을 입력함으로써) 클리어
- UART_CR3의 CTSIE이 1일 때, 인터럽트가 발생
 - 0 : CTS status line을 Disable
 - 1 : CTS status line을 Enable
- 주의 : 이 비트는 UART4 와 UART5에서는 사용할 수 없음

- LBD (LIN break detection flag) LINBreakDetectionFlag(Statusflag)
 - 0 : Disable LIN Break Detect
 - 1 : Enable LIN Break Detect
 - 주의 : Noise error 인터럽트는 LBD = 1, LBDIE = 1 일 때 발생

31~16																	
Reserved																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reserved								CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE

STM32F405 USART 레지스터

- USART_SR (Status register)

- TXE (Transmit data register empty)

- 데이터 전송이 가능할 때, 하드웨어에서 1로 설정
 - UART_CR1의 TXEIE가 1이면, 인터럽트가 발생되며, USART_DR레지스터에 데이터를 입력할 때, 0으로 클리어

- 0 : 데이터 전송 불가
 - 1 : 데이터 전송 가능

- TC (Transmission complete)

- 데이터를 포함한 프레임이 전송되었을 때, 하드웨어에서 1로 설정
 - Transmission Complete 인터럽트는 USART_CR1의 TCIE비트가 1일 때 발생되며, USART_DR레지스터에 데이터를 입력한 후, USART_SR레지스터를 통해 전송상태를 보고 있을 때, 0으로 클리어

31~16																	
Reserved																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reserved								CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE

STM32F405 USART 레지스터

- USART_SR (Status register)

- RXNE (Read data register not empty)

- RDR(Receive Data Register) 시프트 레지스터의 데이터가 USART_DR 레지스터로 전송되어지면, 하드웨어에서 1로 설정
 - 이 인터럽트는 USART_CR1레지스터의 RXNEIE비트가 1일 때, 발생
 - USART_DR레지스터를 읽어 들이면, 0으로 클리어
 - 0 : Data is not received
 - 1 : Received data is ready to be read.

- IDLE (IDLE line detected)

- Idle Line이 검출되면 하드웨어에서 1로 설정
 - 이 인터럽트는 USART_CR1레지스터의 IDLEIE비트가 1일 때, 발생
 - USART_DR레지스터에 데이터를 입력한 후, USART_SR레지스터를 통해 전송상태를 보고 있을 때, 0으로 클리어
 - 0 : No Idle Line is detected
 - 1 : Idle Line is detected

31~16																	
Reserved																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reserved								CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE

STM32F405 USART 레지스터

- USART_SR (Status register)

- ORE (Overrun error)

- RXNE비트가 1로 설정되어 있는 동안, RDR레지스터에 새로운 데이터가 전송되려고 할 때, 하드웨어에서 1로 설정
 - 이 인터럽트는 USAER_CR1레지스터의 RXNEIE비트가 1일 때, 발생
 - USART_DR레지스터에 데이터를 입력한 후, USART_SR레지스터를 통해 전송상태를 보고 있을 때, 0으로 클리어
 - 0 : No Overrun error
 - 1 : Overrun error is detected

- NE (Noise error flag)

- 전송받은 프레임에 노이즈가 발견되었을 때, 하드웨어에서 1로 설정
 - 소프트웨어 순차처리에 의해 0으로 클리어
 - 0 : No noise is detected
 - 1 : Noise is detected

31~16																	
Reserved																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reserved								CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE

STM32F405 USART 레지스터

- USART_SR (Status register)

- FE (Framing error)

- 동기화되지 못하거나 지나친 노이즈, 깨진 문자가 검출되었을 때, 하드웨어에서 1로 설정
 - 소프트웨어 순차처리에 의해 0으로 클리어
 - 0 : No Framing error is detected
 - 1 : Framing error or break character is detected

- PE (Parity error)

- 수신모드에서 패리티 에러가 발생되었을 때, 하드웨어에서 1로 설정
 - 소프트웨어 순차처리에 의해 0으로 클리어
 - 이 인터럽트는 USART_CR1레지스터의 PEIE비트가 1일 때, 발생
 - 0 : No parity error
 - 1 : Parity error

31~16																	
Reserved																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reserved								CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE

STM32F405 USART 레지스터

- USART_DR (Data register)
 - USART 데이터 레지스터
 - USART 모듈의 송수신 데이터를 저장하는 레지스터

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DR[8:0]							

STM32F405 USART 레지스터

- USART_BRR (Baud rate register)
 - USART 보우레이트 레지스터
 - DIV_Mantissa[11:0]와 DIV_Fraction[3:0]을 이용해 USART 통신의 보우레이트를 결정
 - 공식 :
$$\frac{f_{clk}}{8 \times (2 - OVER8) \times USARTDIV}$$
 - USART1을 사용할 때는 fpclk 는 PCLK2(최대 84MHz)가 되며, 그 외의 USART는 PCLK1(최대 42MHz)를 사용
 - USARTDIV는 DIV_Mantissa[11:0] + (DIV_Fraction[3:0]/16)으로 구함
 - 예) USART_BRR = 0x1BC,
 - DIV_Mantissa = 0xD27, DIV_Fraction = 0D12,
 - USARTDIV = 27(0x1B) + (12(0xC0)/16) = 0D27.75

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			

STM32F405 USART 레지스터

- USART_BRR (Baud rate register)

Baud Rate (kbps)	fpclk = 42MHz			fpclk = 84MHz		
	Actual(kbps)	USARTDIV	Error	Actual(kbps)	USARTDIV	Error
1.2	1.200	2187.5	0.0%	1.2	4375	0.0%
2.4	2.400	1093.75	0.0%	2.4	2187.5	0.0%
9.6	9.600	273.4375	0.0%	9.6	546.875	0.0%
19.2	19.195	136.75	0.02%	19.2	273.4375	0.0%
38.4	38.391	38.375	0.02%	38.391	136.75	0.02%
57.6	57.613	45.5625	0.02%	57.613	91.125	0.02%
115.2	115.068	22.8125	0.11%	115.226	45.5625	0.02%
230.4	230.769	11.375	0.16%	230.137	22.8125	0.11%
460.8	461.538	5.6875	0.16%	461.538	11.375	0.16%
921.6	913.043	2.875	0.93%	923.076	5.6875	0.93%
1792	1826	1.4375	1.9%	1787	2.9375	0.27%
18432	1826	1.4375	0.93%	1826	2.875	0.93%
3584	N/A	N/A	N/A	3652	1.4375	1.9%
36864	N/A	N/A	N/A	3652	1.4375	0.93%

STM32F405 USART 레지스터

- USART_CR1 (Control register 1)
 - USART 제어 레지스터 1
 - USART 모듈의 송수신 동작을 제어하는 기능을 수행하는 레지스터
 - OVER8 (Oversampling mode)
 - 0 : oversampling by 16
 - 1 : oversampling by 8
 - UE (USART enable)
 - 0 : USART prescaler and outputs disable
 - USART는 동작하지 않게 되며, 전력 을 줄이는 효과를 얻게 됨
 - 1 : USART enable
 - M (Word length)-USART통신의 word의 type을 결정
 - 0 : 1 Startbit, 8 Databits, n Stop bit
 - 1 : 1 Startbit, 9 Databits, n Stop bit
 - 주의 : 이 비트는 데이터가 송수신 중에는 변경되어서는 안됨

31~16																
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OVER8	Res	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK	

STM32F405 USART 레지스터

- USART_CR1 (Control register 1)

- WAKE (Wakeup method)
 - 0 : Idle Line
 - 1 : Address Mark
- PCE (Parity control enable)
 - 하드웨어레벨의 패리티제어를 결정
 - 1로 설정하면, 수신받은 데이터에서 패리티를 검출하여 MSB 포지션에 적용
 - 0 또는 1로 설정되면, 다음 송수신되는 데이터부터 적용
 - 0 : Parity control disabled
 - 1 : Parity control enabled
- PS (Parity selection)
 - PCE비트가 설정되어 있을 때, odd/even 패리티를 선택
 - 0 또는 1로 설정되면, 다음 송수신되는 데이터부터 적용
 - 0 : Even parity
 - 1 : Odd parity

31~16

Reserved

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Res	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK

STM32F405 USART 레지스터

- USART_CR1 (Control register 1)
 - PEIE (Parity error interrupt enable)
 - 소프트웨어 레벨에서 설정
 - 0 : Interrupt를 무시
 - 1 : PE비트가 1일 때, 인터럽트 enable
 - TXEIE (TXE interrupt enable)
 - 소프트웨어 레벨에서 설정
 - 0 : Interrupt를 무시
 - 1 : TXE비트가 1일 때, 인터럽트 enable
 - TCIE (Transmission complete interrupt enable)
 - 소프트웨어 레벨에서 설정
 - 0 : Interrupt를 무시
 - 1 : TC비트가 1일 때, 인터럽트 enable

31~16																
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OVER8	Res	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK	

STM32F405 USART 레지스터

- USART_CR1 (Control register 1)
 - RXNEIE (RXNE interrupt enable)
 - 소프트웨어 레벨에서 설정
 - 0 : Interrupt를 무시
 - 1 : ORE비트나 RXNE비트가 1일 때, 인터럽트 enable
 - IDLEIE (IDLE interrupt enable)
 - 소프트웨어 레벨에서 설정
 - 0 : Interrupt를 무시
 - 1 : IDLE비트 1일 때, 인터럽트 enable
 - TE (Transmitter enable)
 - 전송을 하려면 이 비트가 1로 설정되어 있어야 하며, 소프트웨어 레벨에서 설정
 - 0 : Transmitter is disabled
 - 1 : Transmitter is enabled

31~16																
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OVER8	Res	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK	

STM32F405 USART 레지스터

- USART_CR1 (Control register 1)

- RE (Receiver enable) 수신을 하려면 이 비트가 1로 설정되어 있어야 하며, 소프트웨어 레벨에서 설정
 - 0 : Receiver is disabled
 - 1 : Receiver is enabled and begins searching for a start bit
- RWU (Receiver wakeup)
 - 0 : Receiver in active mode
 - 1 : Receiver in mute mode
- SBK (Send break)
 - 0 : No break character is transmitted
 - 1 : Break character will be transmitted

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Res	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK

STM32F405 USART 레지스터

- USART_CR2 (Control register 2)
 - USART 제어 레지스터 2
 - USART 모듈의 송수신 동작을 제어하는 기능을 수행하는 레지스터
 - LINEN (LIN mode enable)
 - 0 : LIN mode disabled
 - 1 : LIN mode enabled
 - STOP (STOP bits)
 - 이 비트들로 스톱비트를 정의
 - 00 : 1 Stop bit
 - 10 : 0.5 Stop bit
 - 10 : 2 Stop bit
 - 11 : 1.5 Stop bit

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	LINEN	STOP[1:0]	CLKEN	CPOL	CPHA	LBCL	Res	LBDIE	LBDL	Res	ADD[3:0]				

STM32F405 USART 레지스터

- USART_CR2 (Control register 2)
 - CLKEN (Clock enable)
 - 0 : SCLK pin disabled
 - 1 : SCLK pin enabled
 - CPOL (Clock polarity)
 - SCLK핀(동기화모드)의 출력 클럭의 극성을 선택
 - 0 : 전송중의 SCLK핀을 항상 low로 함
 - 1 : 전송중의 SCLK핀을 항상 high로 함
 - CPHA (Clock phase)
 - SCLK핀(동기화모드)의 출력 클럭의 위상을 선택
 - 0 : 첫 번째 전송 클럭은 데이터의 처음을 의미
 - 1 : 두 번째 전송 클럭은 데이터의 처음을 의미

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res	LBDIE	LBDL	Res	ADD[3:0]			

STM32F405 USART 레지스터

- USART_CR2 (Control register 2)
 - LBCL (Last bit clock pulse)
 - SCLK핀(동기화모드에서)으로 데이터의 끝을 알리는 출력 여부를 결정
 - 0 : 마지막 데이터 비트의 클럭 펄스를 SCLK핀으로 출력하지 않음
 - 1 : 마지막 데이터 비트의 클럭 펄스를 SCLK핀으로 출력
 - LBDLE (LIN break detection interrupt enable)
 - Break interrupt mask
 - 0 : 이 인터럽트를 무시
 - 1 : USART_SR레지스터의 LBD = 1이지 않을 때, 인터럽트를 발생
 - LBDL (LIN break detection length)
 - 10비트 또는 11비트의 bit break detection을 선택
 - 0 : 10bit break detection
 - 1 : 11bit break detection

31~16																
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Res	LINEN	STOP[1:0]	CLKEN	CPOL	CPHA	LBCL	Res	LBDIE	LBDL	Res	ADD[3:0]					

STM32F405 USART 레지스터

- USART_CR2 (Control register 2)
 - ADD[3:0] (Address of the USART node)
 - USART 노드 주소를 결정
 - 어드레스(주소)를 통한 wake-up을 위한 mute모드인 동안 multiprocessor 통신에서 사용

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	LINEN	STOP[1:0]	CLKEN	CPOL	CPHA	LBCL	Res	LBDIE	LBDL	Res	ADD[3:0]				

STM32F405 USART 레지스터

- USART_CR2 (Control register 2)
 - USART 모듈의 송수신 동작을 제어하는 기능을 수행하는 레지스터
 - LINEN (LIN mode enable)
 - 0 : LINmode disabled
 - 1 : LINmode enabled
 - STOP (STOP bits)
 - 이 비트들로 스톱비트를 정의
 - 00 : 1 Stop bit
 - 10 : 0.5 Stop bit
 - 10 : 2 Stop bit
 - 11 : 1.5 Stop bit

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res	LBDIE	LBDL	Res	ADD[3:0]			

STM32F405 USART 레지스터

- USART_CR3 (Control register 3)
 - USART 제어 레지스터 3
 - USART 모듈의 송수신 동작을 제어하는 기능을 수행하는 레지스터
 - ONEBIT (One sample bit method enable)
 - 이 비트로 샘플 비트를 정의
 - 0 : Three sample bit method
 - 1 : One sample bit method
 - CTSIE (CTS interrupt enable)
 - 0 : Interrupt is inhibited
 - 1 : An interrupt is generated whenever CTS=1 in the USART_SR register
 - 주의 : 이 비트는 UART4와 UART5에서는 사용할 수 없음
 - CTSE (CTS enable)
 - 0 : CTS hardware flow control disabled
 - 1 : CTS mode enabled
 - 주의 : 이 비트는 UART4와 UART5에서는 사용할 수 없음

31~16																
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved				ONEBIT	IRLP	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE	

STM32F405 USART 레지스터

- USART_CR3 (Control register 3)
 - RTSE (RTS enable)
 - 0 : RTS hardware flow control disabled
 - 1 : RTS interrupt enabled
 - 주의 : 이 비트는 UART4와 UART5에서는 사용할 수 없음
 - DMAT (DMA enable transmitter)
 - 소프트웨어에 의해 셋/리셋
 - 0 : DMA mode is enabled for transmission
 - 1 : DMA mode is disabled for transmission
 - DMAR (DMA enable receiver)
 - 소프트웨어에 의해 셋/리셋
 - 0 : DMA mode is enabled for reception
 - 1 : DMA mode is disabled for reception

31~16																
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved				ONEBIT	IRLP	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE	

STM32F405 USART 레지스터

- USART_CR3 (Control register 3)
 - SCEN (Smartcard mode enable)
 - 스마트 카드 모드를 설정하는 비트
 - 0 : Smartcard Mode disabled
 - 1 : Smartcard Mode enabled
 - NACK (Smartcard NACK enable)
 - 0 : NACK transmission in case of parity error is disabled
 - 1 : NACK transmission during parity error is enabled
 - HDSEL (Half-duplex selection)
 - Single-wire Half-duplex mode를 설정하는 비트
 - 0 : Half duplex mode is not selected
 - 1 : Half duplex mode is selected

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			ONEBIT	IRLP	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE	

STM32F405 USART 레지스터

- USART_CR3 (Control register 3)
 - IRLP (IrDA low-power)
 - normal mode 와 low-power IrDA mode 중 선택하는 비트
 - 0 : Normal mode
 - 1 : Low-power mode
 - IREN (IrDA mode enable)
 - 소프트웨어에 의해 셋/클리어
 - 0 : IrDA disabled
 - 1 : IrDA enabled
 - EIE (Error interrupt enable)
 - 0 : Interrupt is inhibited
 - 1 : An interrupt is generated whenever DMAR=1 in the USART_CR3 register and FE=1 or ORE=1 or NF=1 in the USART_SR register

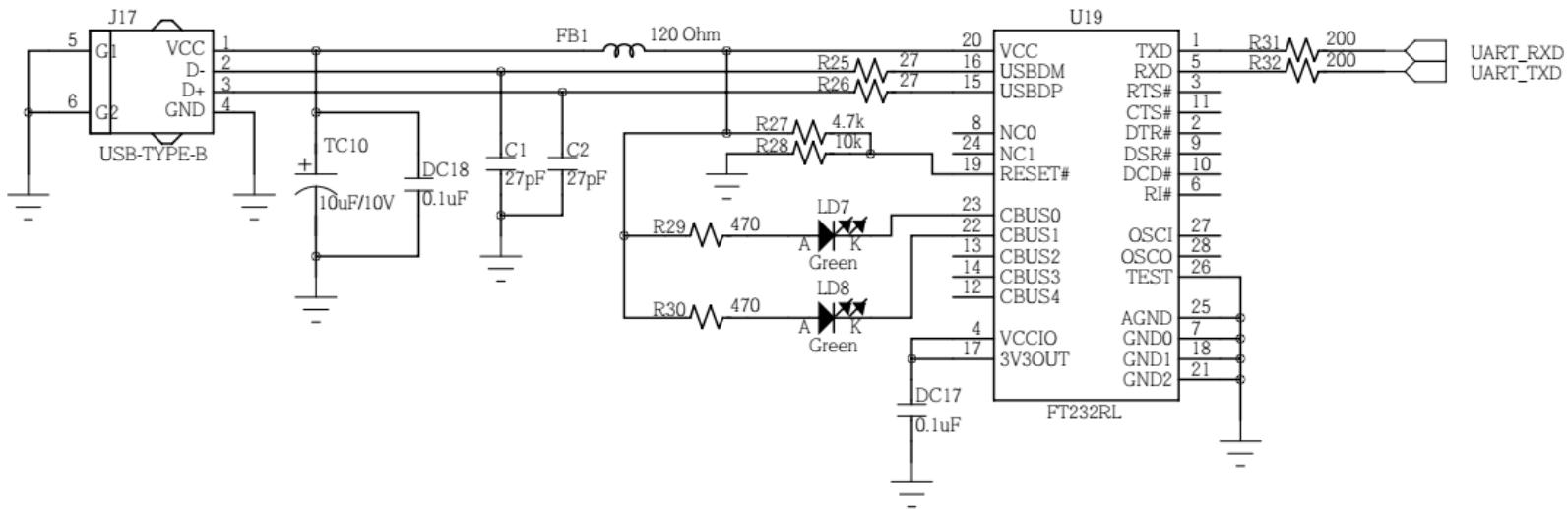
31~16																
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved				ONEBIT	IRLP	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE	

실습 1 : UART로 Hello 보내기

- 실습 개요
 - UART를 이용하여 미리 작성된 문장("Hello World")을 PC로 전송하기
 - STM32F405의 USART 포트를 입력과 출력으로 선언하고 이 포트를 UART 모듈에 연결
 - USB 케이블을 이용하여 PC와 연결
- 실습 목표
 - UART 기능 동작원리 이해
 - STM32F405의 USART 제어 방법의 습득(관련 레지스터 이해)
 - UART를 통해 PC와 통신하는 방법 습득

실습 1 : UART로 Hello 보내기

- 사용 모듈 : UART 모듈 회로



실습 1 : UART로 Hello 보내기

- 구동 프로그램 : 사전 지식
 - 미리 정해 주어야 하는 통신 규약을 결정

Baud Rate	115200
패리티	No Parity
Stop Bit	1
전송문자 데이터 비트수	8
흐름제어	없음

- USART 제어 레지스터 세팅
 - 비동기 전송 모드
 - 멀티 프로세서 통신 모드
 - USART의 RX와 TX를 Enable
- USART_SR 레지스터의 플래그를 보면서 데이터를 보낼 수 있는 상태를 기다렸다가 USART_DR 레지스터에 데이터를 넣어주면 USART로 데이터가 출력

실습 1 : UART로 Hello 보내기

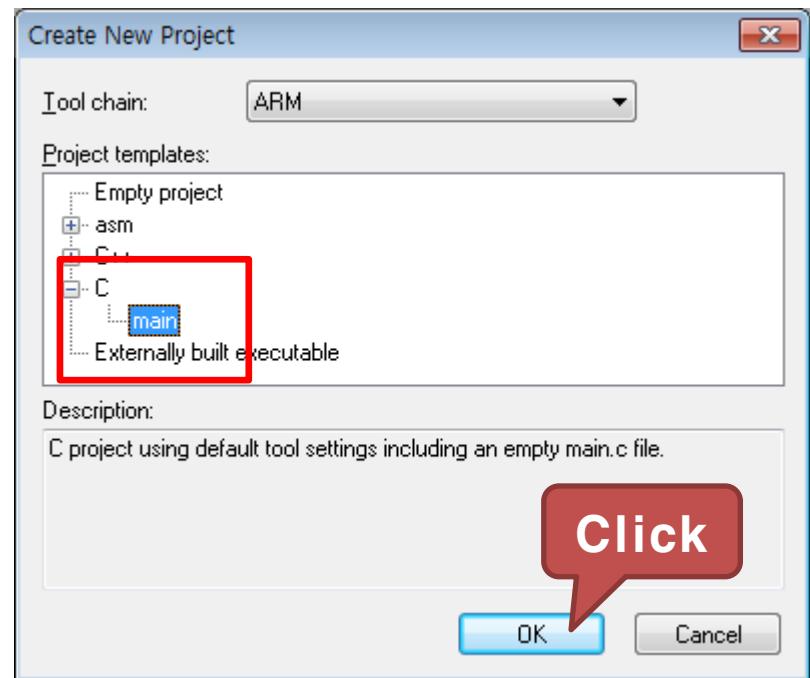
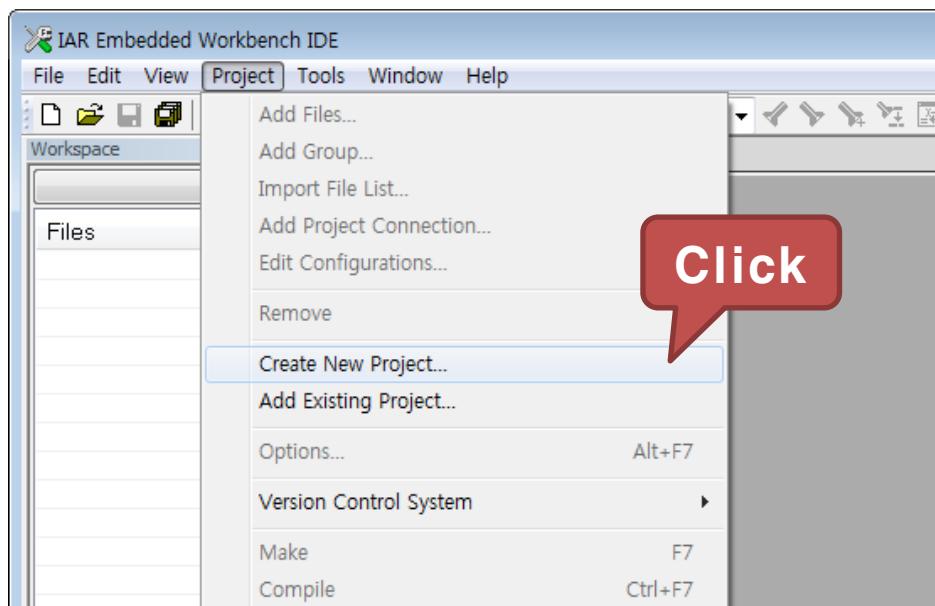
- TIM 라이브러리
 - STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
 - USART_InitTypeDef 구조체

USART_InitTypeDef구조체	설 명
uint32_t USART_BaudRate	보우레이트 설정(USART_BRR에 맞게 계산됨)
uint16_t USART_WordLength	word 길이 설정(USART_CR1의 M비트)
uint16_t USART_StopBits	stop 비트 설정(USART_CR2의 STOP비트)
uint16_t USART_Parity	패리티 비트 설정(USART_CR1의 PCE, PS)
uint16_t USART_Mode;	USART 통신모드 설정(USART_CR1의 TE, RE비트)
uint16_t USART_HardwareFlowControl;	하드웨어 흐름제어 설정(USART_CR3의 CTSE,RTSE)

실습 1 : UART로 Hello 보내기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch8” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “uartHello”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1 : UART로 Hello 보내기

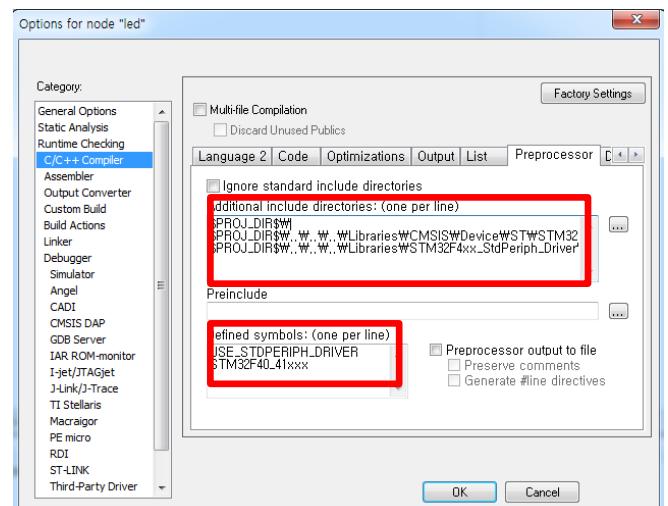
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch8\uartHello	system_stm32f4xx.c
Cortex_Example\Projects\ch8\uartHello	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_usart.c

실습 1 : UART로 Hello 보내기

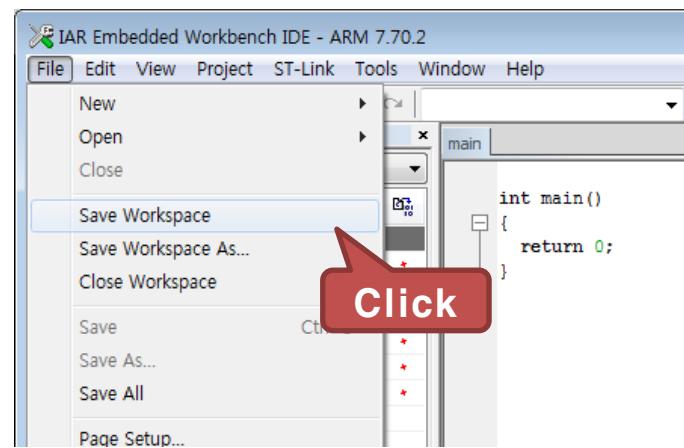
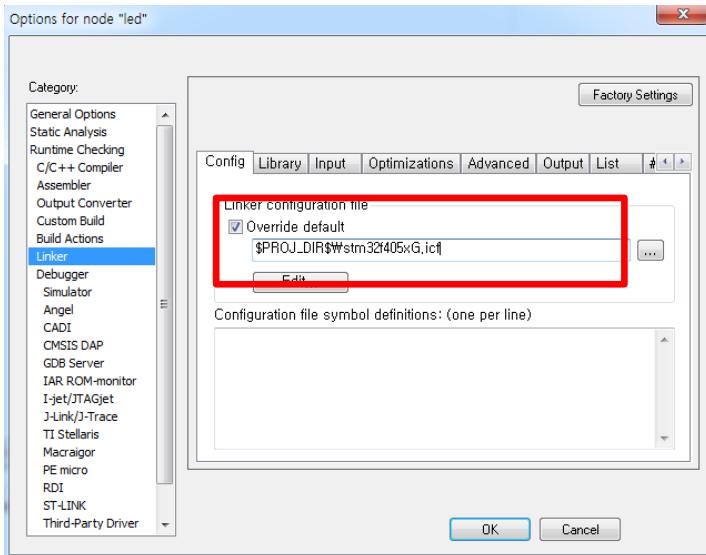
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1 : UART로 Hello 보내기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1 : UART로 Hello 보내기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

void putch(uint8_t c) { // USART으로 문자 하나를 전송하는 함수
    USART_SendData(USART1, c);
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
}

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    unsigned char text[]="Hello! World!! \r\n";
    unsigned char i=0;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
```

실습 1 : UART로 Hello 보내기

- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;  
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9|GPIO_Pin_10;  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);
```

실습 1 : UART로 Hello 보내기

- 구동 프로그램

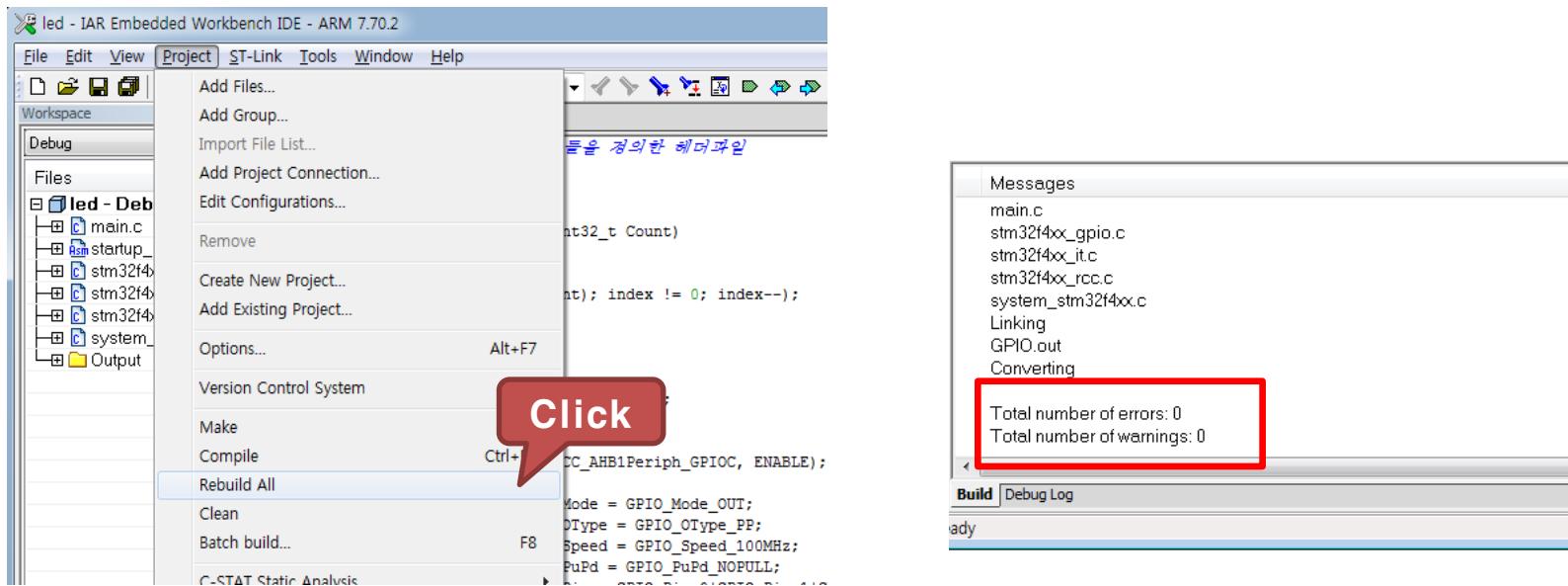
- main.c 코드 작성

```
//...  
USART_InitStructure.USART_BaudRate = 115200;  
USART_InitStructure.USART_WordLength = USART_WordLength_8b;  
USART_InitStructure.USART_StopBits = USART_StopBits_1;  
USART_InitStructure.USART_Parity = USART_Parity_No;  
USART_InitStructure.USART_HardwareFlowControl =  
    USART_HardwareFlowControl_None;  
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;  
USART_Init(USART1, &USART_InitStructure);  
USART_Cmd(USART1, ENABLE);  
  
while(text[i]!='\0') { // 데이터가 '\0' 인경우는 문자열의 끝임  
    putch(text[i++]); // 저장된 문자열을 출력  
}  
while(1){}  
}
```

실습 1 : UART로 Hello 보내기

● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 uartHello 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 1 : UART로 Hello 보내기

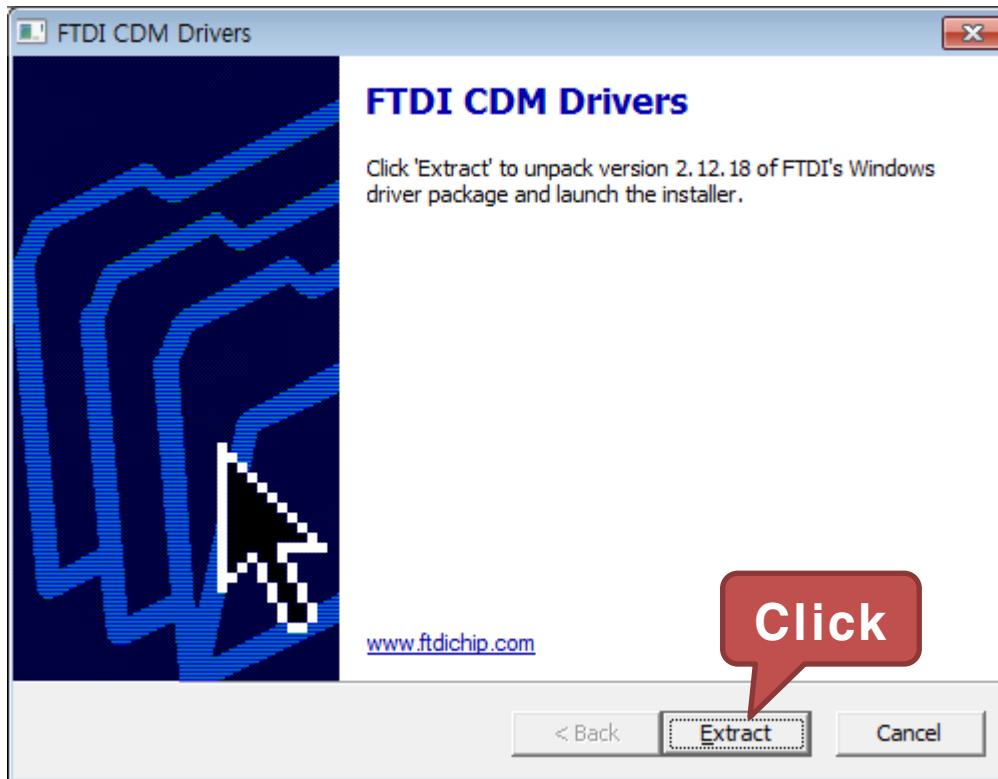
● VCP 설치

- UART 실습을 하려면 PC와 장비를 USB로 연결해야 하며, 장비의 USB-to-Serial 장치의 드라이버를 설치해야 함
- VCP 설치하기(다운로드) : <http://www.ftdichip.com/Drivers/VCP.htm>

Operating System	Release Date	Processor Architecture							Comments
		x86 (32-bit)	x64 (64-bit)	PPC	ARM	MIPSII	MIPSIV	SH4	
Windows*	2016-06-23	2.12.18	2.12.18	-	-	-	-	-	WHQL Certified. Includes VCP and P2XX. Available as a setup executable Please read the Release Notes and Installation Guides.
Linux	2009-05-14	1.5.0	1.5.0	-	-	-	-	-	All FTDI devices now supported in Ubuntu 11.10, kernel 3.0.0-19 Refer to TN-101 if you need a custom VCP VID/PID in Linux
Mac OS X 10.3 to 10.8	2012-08-10	2.2.18	2.2.18	2.2.18	-	-	-	-	Refer to TN-105 if you need a custom VCP VID/PID in MAC OS

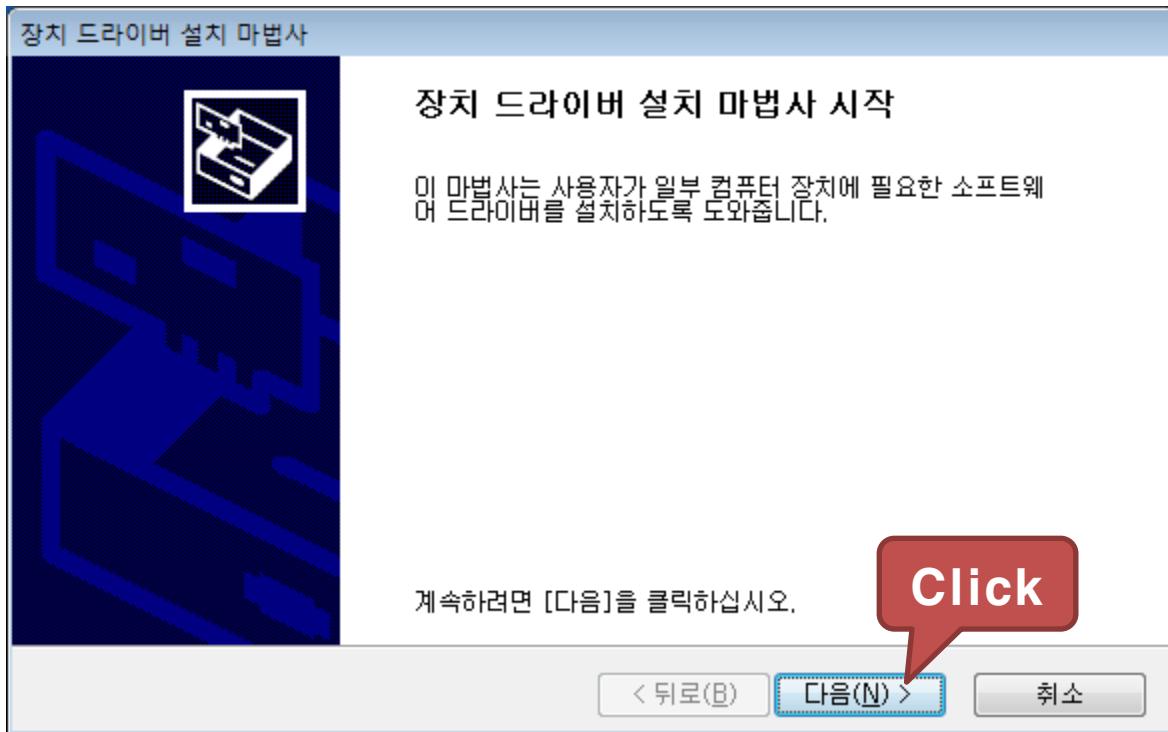
실습 1 : UART로 Hello 보내기

- VCP 설치
 - CDM21218_Setup.exe 파일을 실행



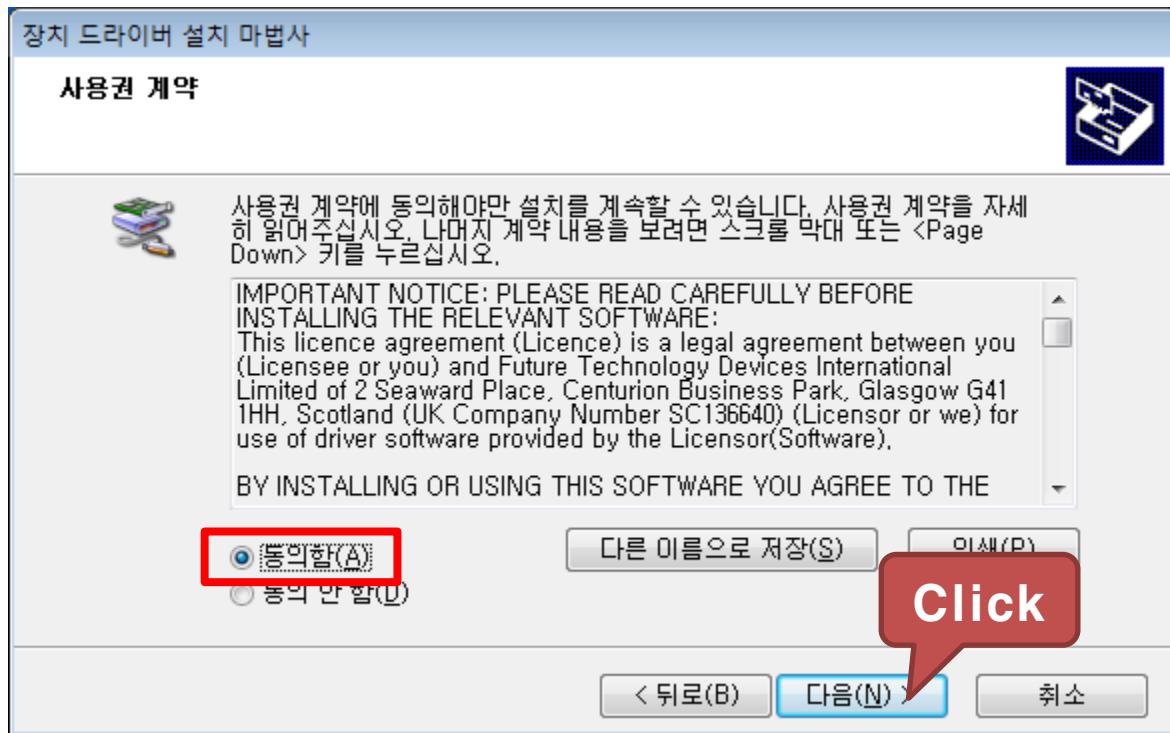
실습 1 : UART로 Hello 보내기

- VCP 설치



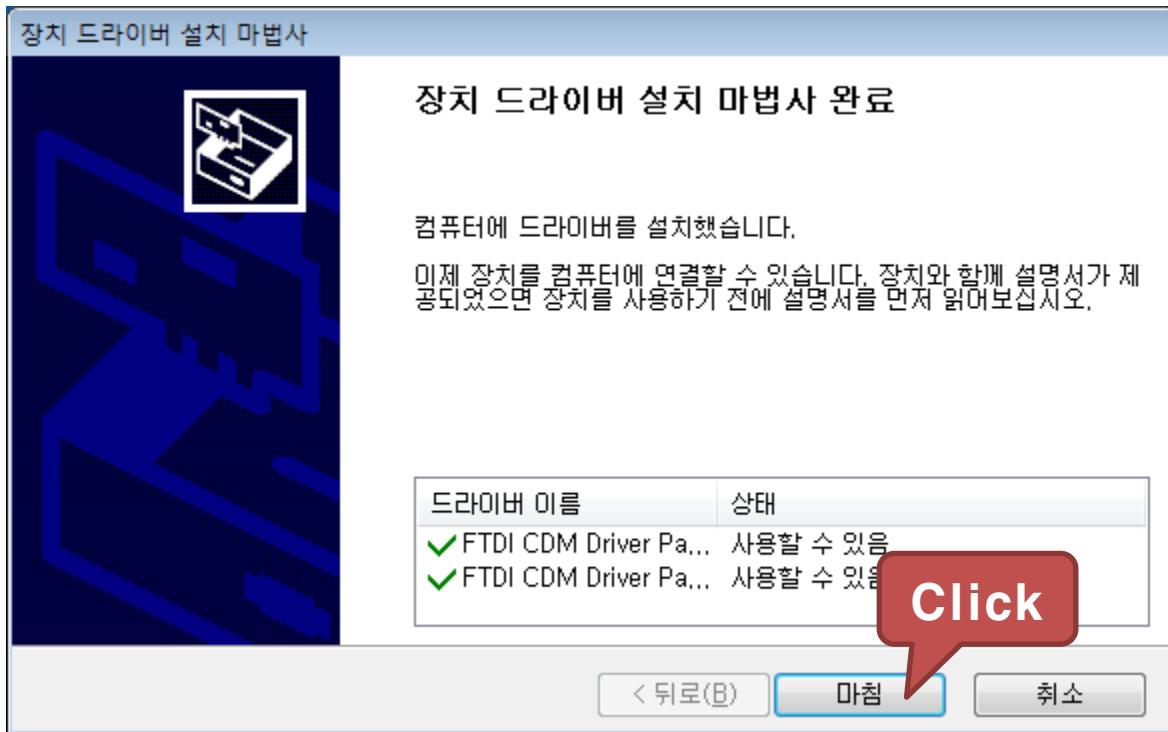
실습 1 : UART로 Hello 보내기

- VCP 설치



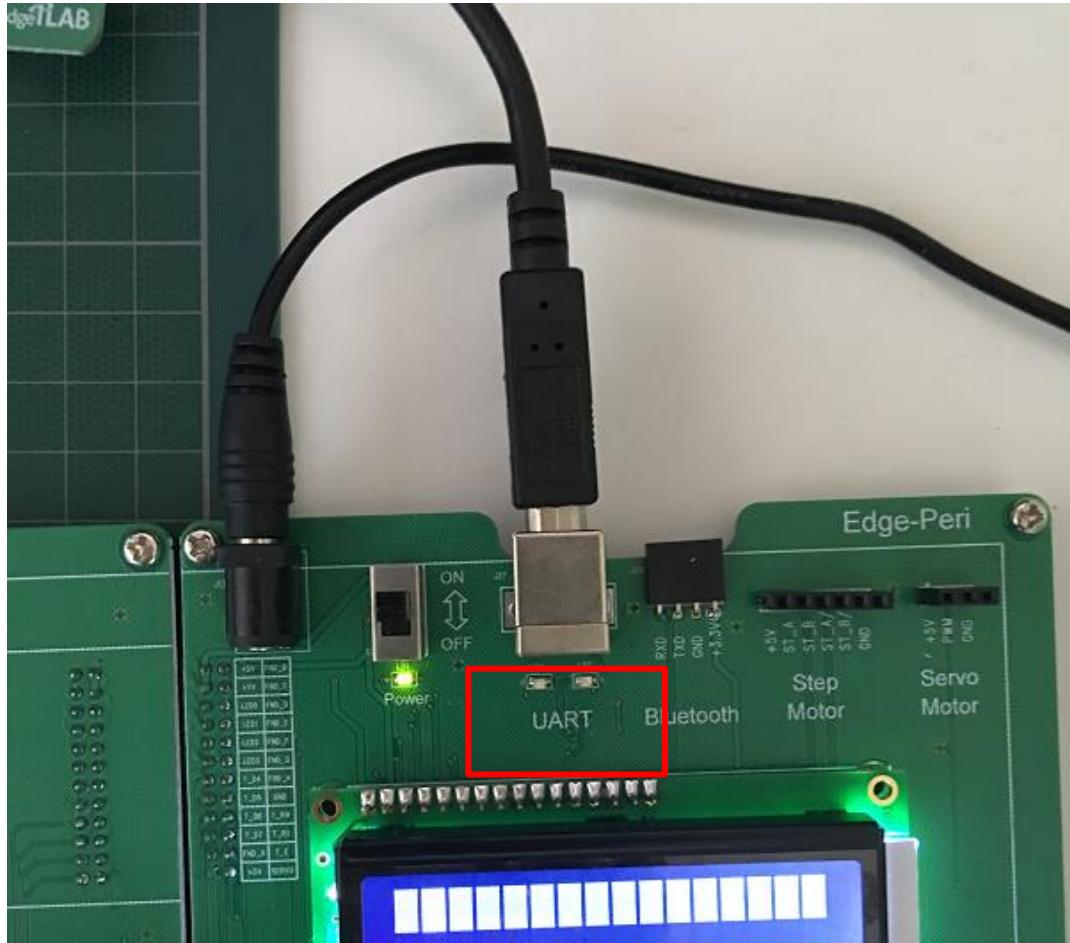
실습 1 : UART로 Hello 보내기

- VCP 설치



실습 1 : UART로 Hello 보내기

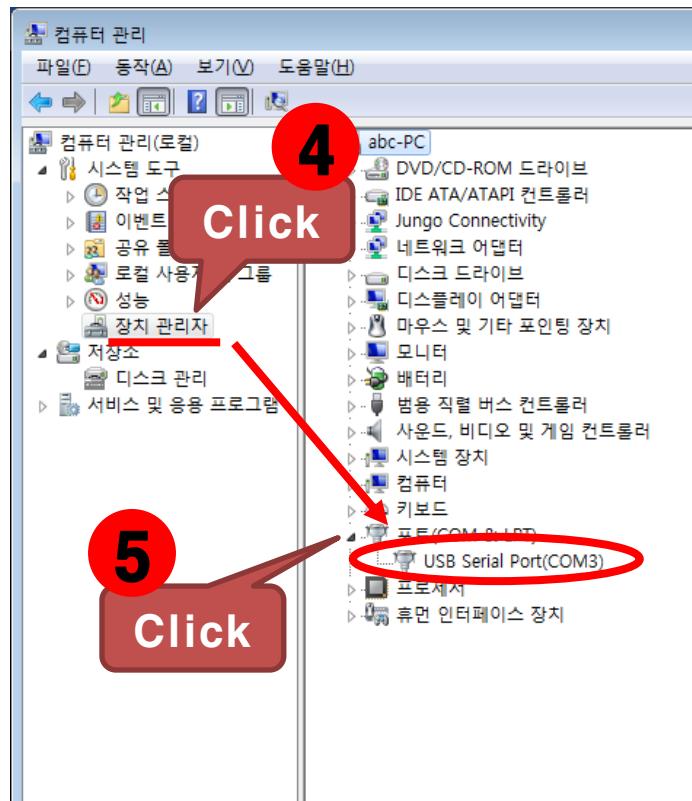
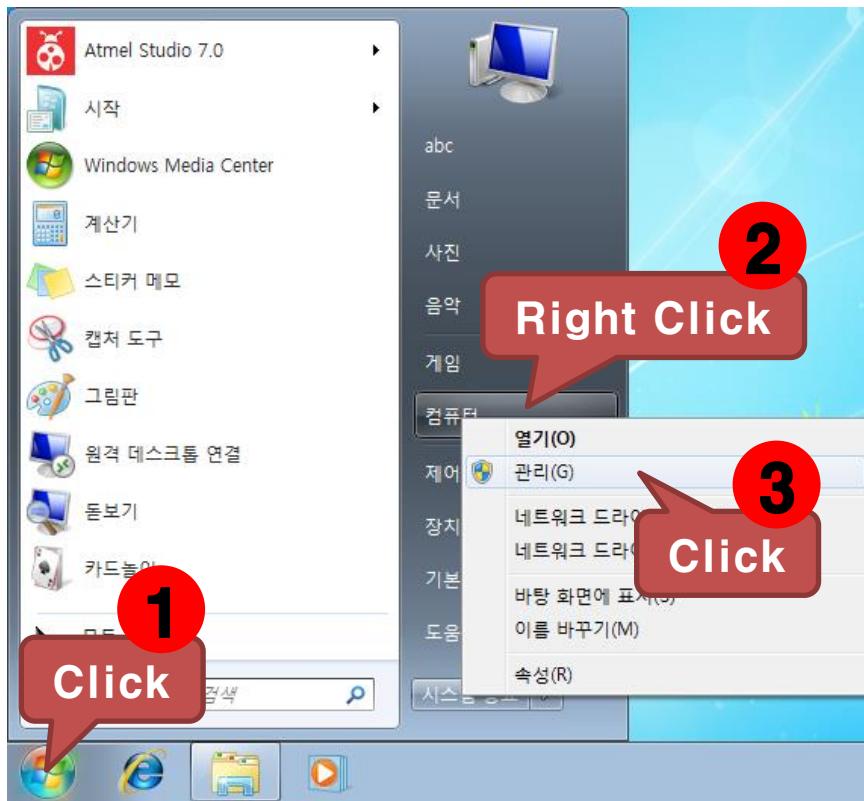
- Edge Peri 보드의 UART 커넥터에 USB 케이블을 꽂은 뒤 PC와 연결



실습 1 : UART로 Hello 보내기

- VCP 포트 확인

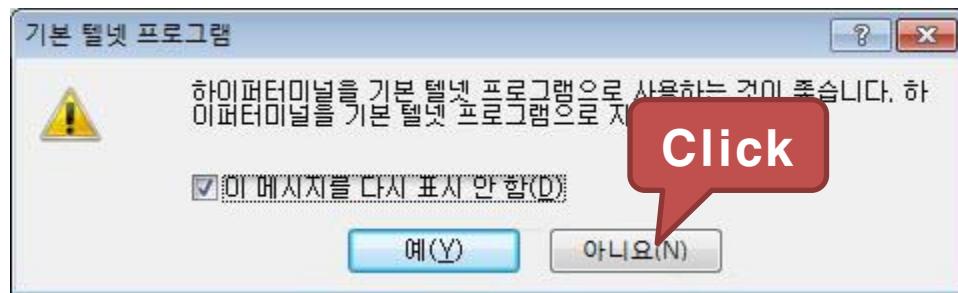
- 장치관리자를 실행해서 USB Serial Port 번호를 확인하고 기억해둠



실습 1 : UART로 Hello 보내기

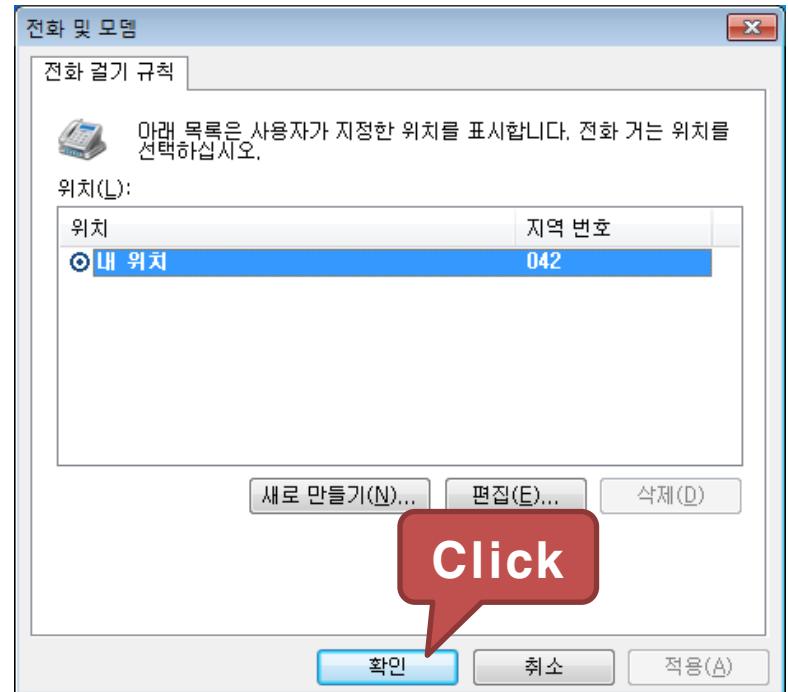
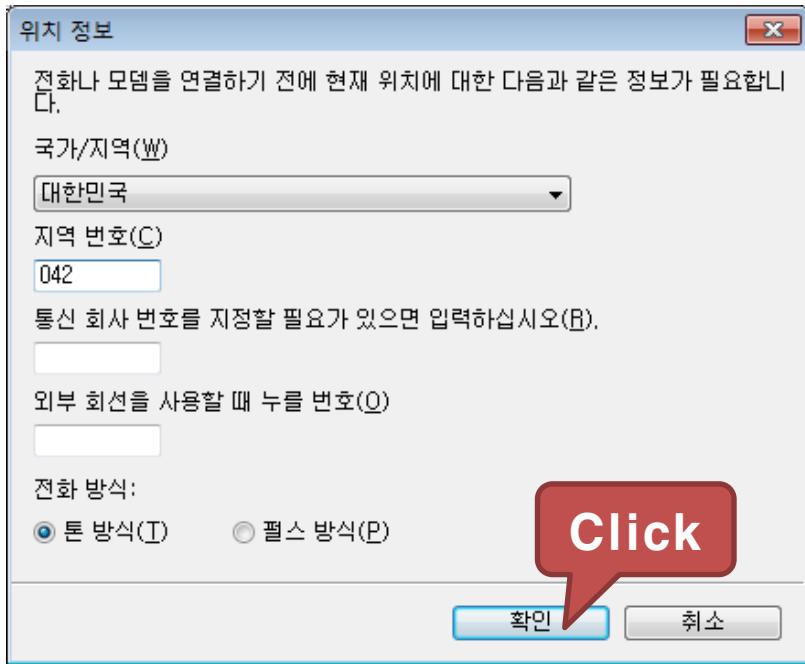
● 하이퍼터미널 실행

- 예제 확인은 하이퍼터미널을 통해서 하게 되며, 인터넷에서 하이퍼터미널을 임의의 폴더에 다운로드 후, "hypertrm.exe" 파일을 실행
- 기본 텔넷 프로그램 설정
 - "이 메시지를 다시 표시 안함(D)"를 체크하고 "아니오"를 클릭



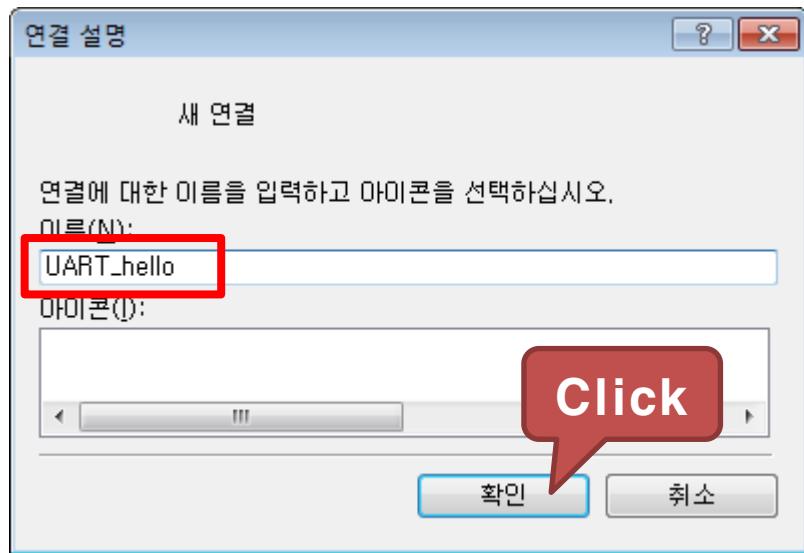
실습 1 : UART로 Hello 보내기

- 하이퍼터미널 실행
 - 위치 정보 입력창에서 아무 지역번호나 입력하고 “확인”을 클릭
 - 전화 및 모뎀창에서는 “확인”을 클릭



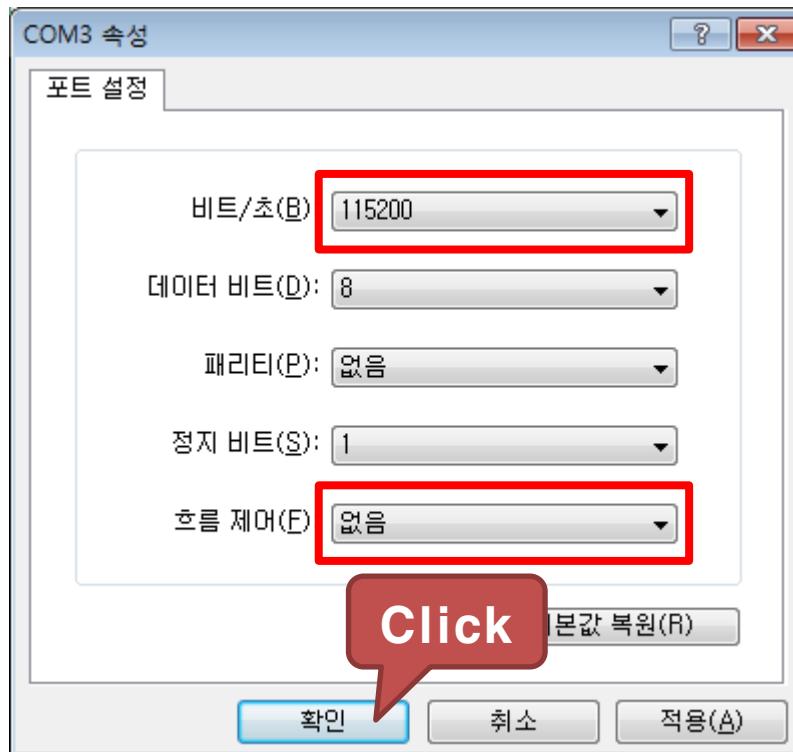
실습 1 : UART로 Hello 보내기

- 하이퍼터미널 실행
 - 연결 설명 창의 이름 란에 “UART_hello” 라고 확인을 클릭
 - 연결 대상창의 연결에 사용할 모뎀 번호를 설정하고 확인을 클릭



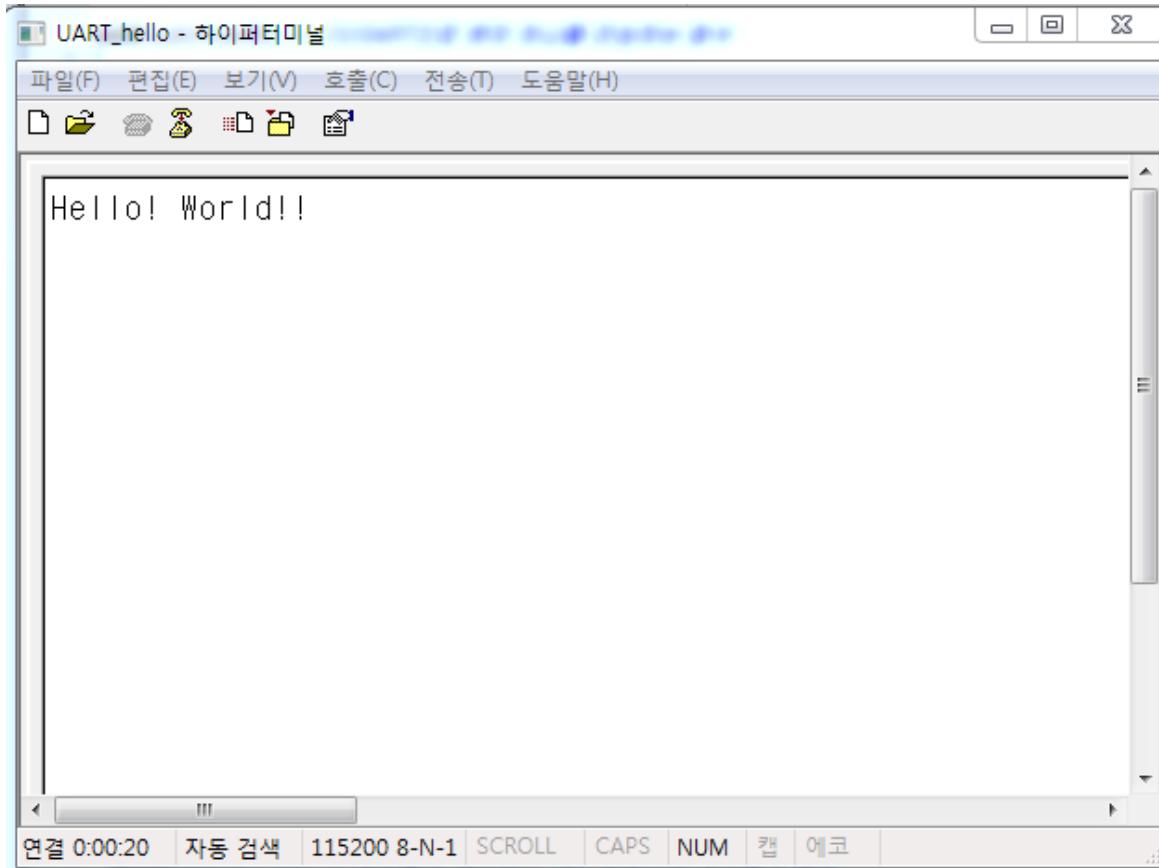
실습 1 : UART로 Hello 보내기

- 하이퍼터미널 실행
 - 통신 설정을 그림과 같이하고, 확인을 클릭



실습 1 : UART로 Hello 보내기

- 실행 결과 확인
 - 이제 MCU 모듈의 리셋 버튼을 누르면 다음 그림과 같은 결과를 확인



실습 2 : UART로 PC와 데이터 주고받기

- 실습 개요
 - PC로부터 전송되는 문자열을 받아 다시 PC로 되돌려 전송하도록 함
 - 설정은 앞의 예제와 유사
 - 프로그램의 작성된 문장을 PC 화면에 뿌려주는 역할과 키보드를 통해 입력 받은 ASCII값을 화면상에 보여주는 기능을 함
- 실습 목표
 - UART 기능 동작원리 이해
 - STM32F405의 USART 제어 방법의 습득(관련 레지스터 이해)
 - UART를 통해 PC와 통신하는 방법 습득

실습 2 : UART로 PC와 데이터 주고받기

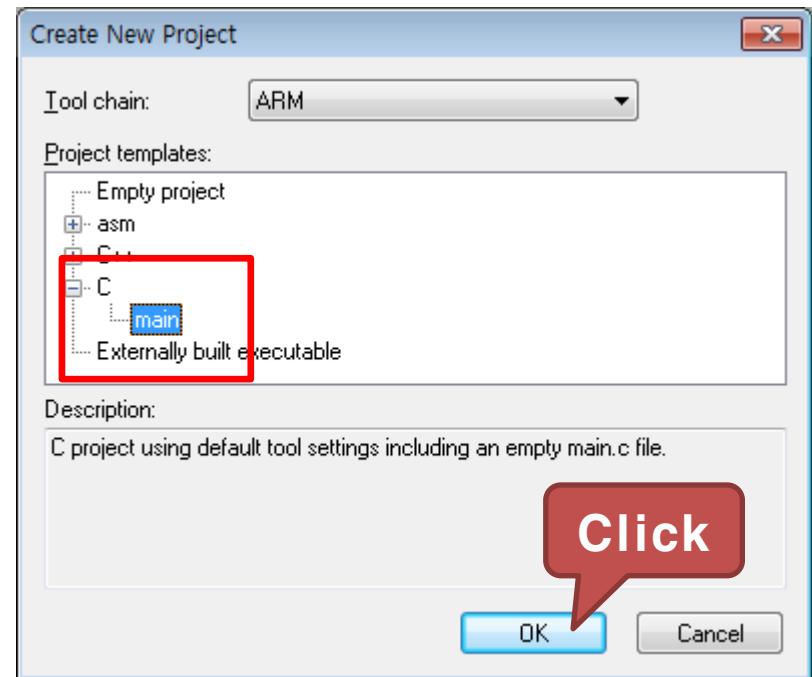
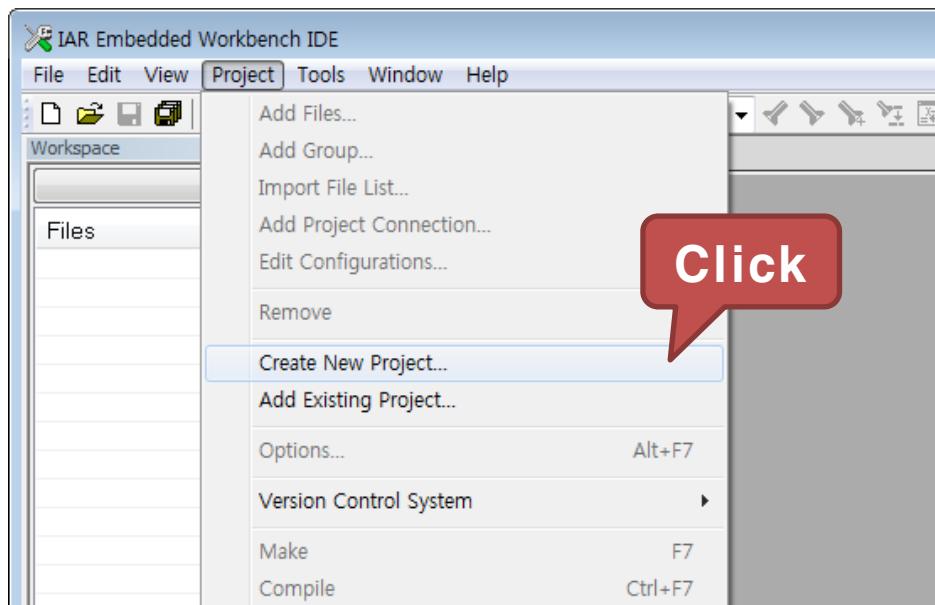
- 구동 프로그램 : 사전 지식

- STM32F405의 USART 포트를 통해서 PC와 UART 통신을 연결하고, PC로 받은 문자열을 그대로 되돌려 전송
 - 설정 : 기본적인 UART 설정은 이전 예제와 동일하게 설정
 - 데이터 수신
 - 모든 설정이 끝나면 USART_SR 레지스터의 플래그를 보면서 PC로부터 데이터가 도착했는지 살펴보고 있다가 데이터가 도착하면 USART_DR 레지스터로부터 데이터를 가져오면 됨
 - 데이터 송신
 - USART_SR 레지스터의 플래그를 보면서 데이터를 보낼 수 있는 상태를 기다렸다가 USART_DR 레지스터에 데이터를 넣어주면 UART로 데이터가 출력

실습 2 : UART로 PC와 데이터 주고받기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch8” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “uartEcho”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : UART로 PC와 데이터 주고받기

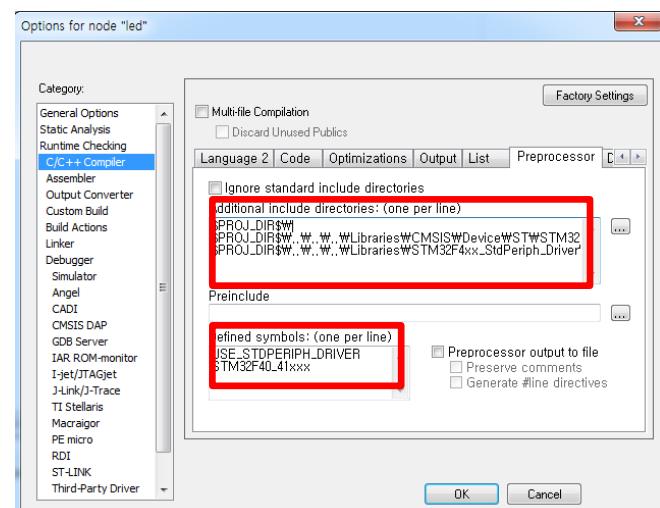
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch8\uartEcho	system_stm32f4xx.c
Cortex_Example\Projects\ch8\uartEcho	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_usart.c

실습 2 : UART로 PC와 데이터 주고받기

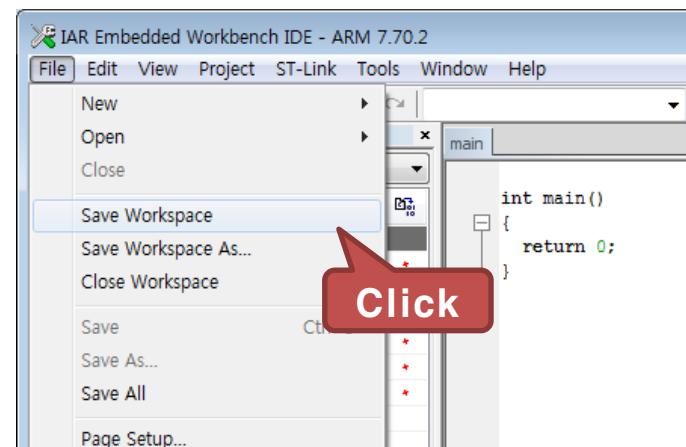
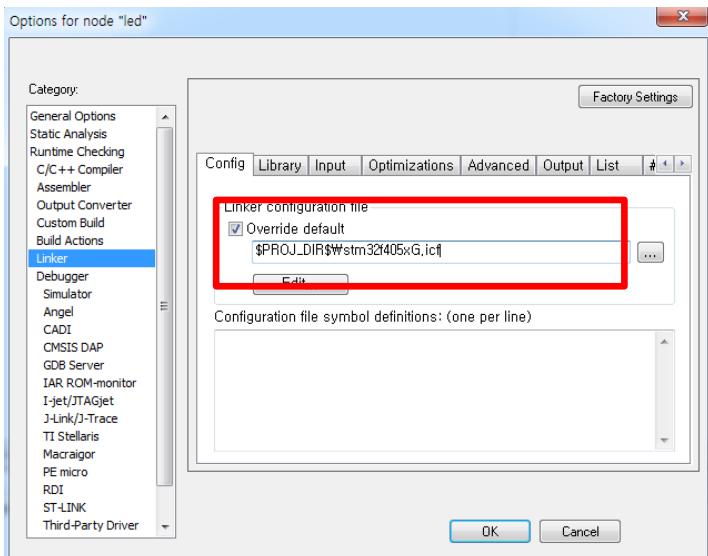
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : UART로 PC와 데이터 주고받기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : UART로 PC와 데이터 주고받기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

void putch(uint8_t c) { // USART1으로 문자 하나를 전송하는 함수
    USART_SendData(USART1, c);
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
}

uint8_t getch(void) { // USART1으로부터 한 문자를 입력받는 함수
    uint8_t key= 0;
    while (USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET);
    key = USART_ReceiveData(USART1);
    return key;
}
```

실습 2 : UART로 PC와 데이터 주고받기

- 구동 프로그램
 - main.c 코드 작성

```
int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    unsigned char text[]="USART 0 Test Program.\r\n";
    unsigned char echo[]="ECHO >> ";
    unsigned char i=0;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9|GPIO_Pin_10;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
```

실습 2 : UART로 PC와 데이터 주고받기

- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);  
  
USART_InitStructure USART_BaudRate = 115200;  
  
USART_InitStructure USART_WordLength = USART_WordLength_8b;  
USART_InitStructure USART_StopBits = USART_StopBits_1;  
USART_InitStructure USART_Parity = USART_Parity_No;  
USART_InitStructure USART_HardwareFlowControl =  
    USART_HardwareFlowControl_None;  
USART_InitStructure USART_Mode = USART_Mode_Rx | USART_Mode_Tx;  
USART_Init(USART1, &USART_InitStructure);  
USART_Cmd(USART1, ENABLE);
```

실습 2 : UART로 PC와 데이터 주고받기

- 구동 프로그램

- main.c 코드 작성

```
//...
while(text[i]!='\0') // 데이터가 '\0' 인경우는 문자열의 끝임
    putch(text[i++]); // 저장된 text 문자열을 출력

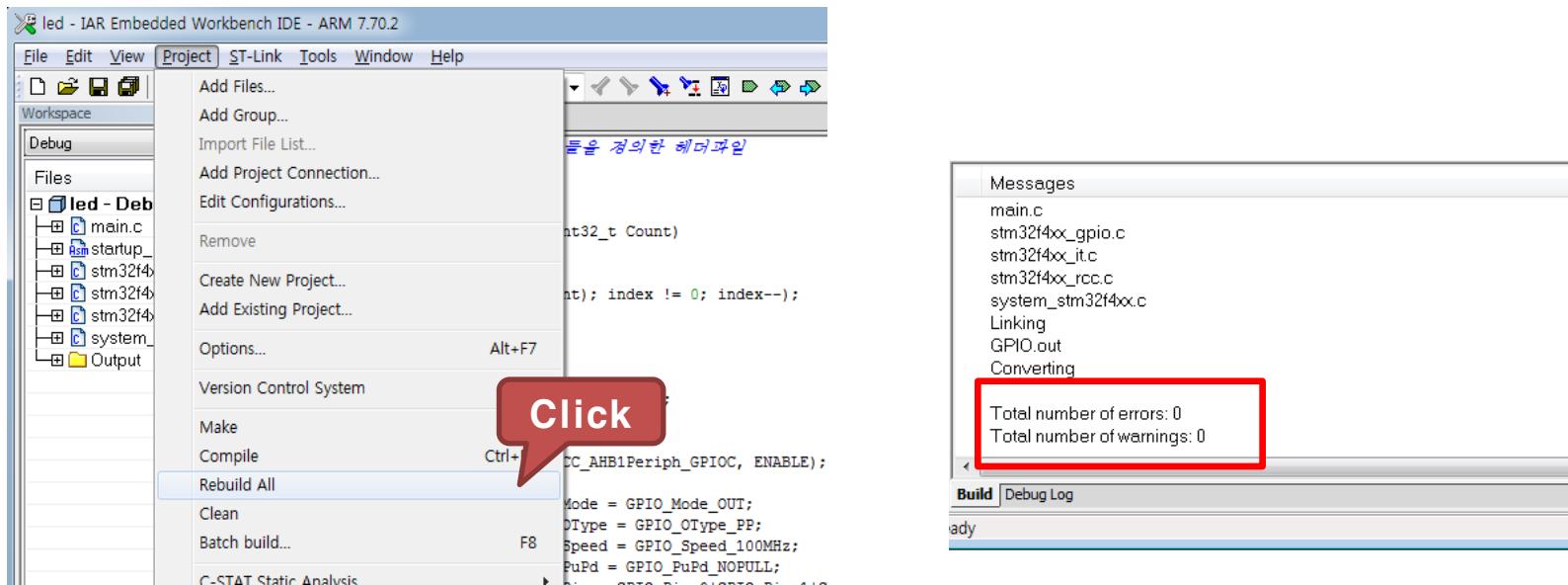
i=0; // 다음 문자열을 출력하기 위해 카운터 변수 초기화
while(echo[i]!='\0') // 데이터가 '\0' 인경우는 문자열의 끝임
    putch(echo[i++]); // 저장된 echo 문자열을 출력

while(1)
{
    putch(getch());
}
```

실습 2 : UART로 PC와 데이터 주고받기

● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 uartEcho 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭

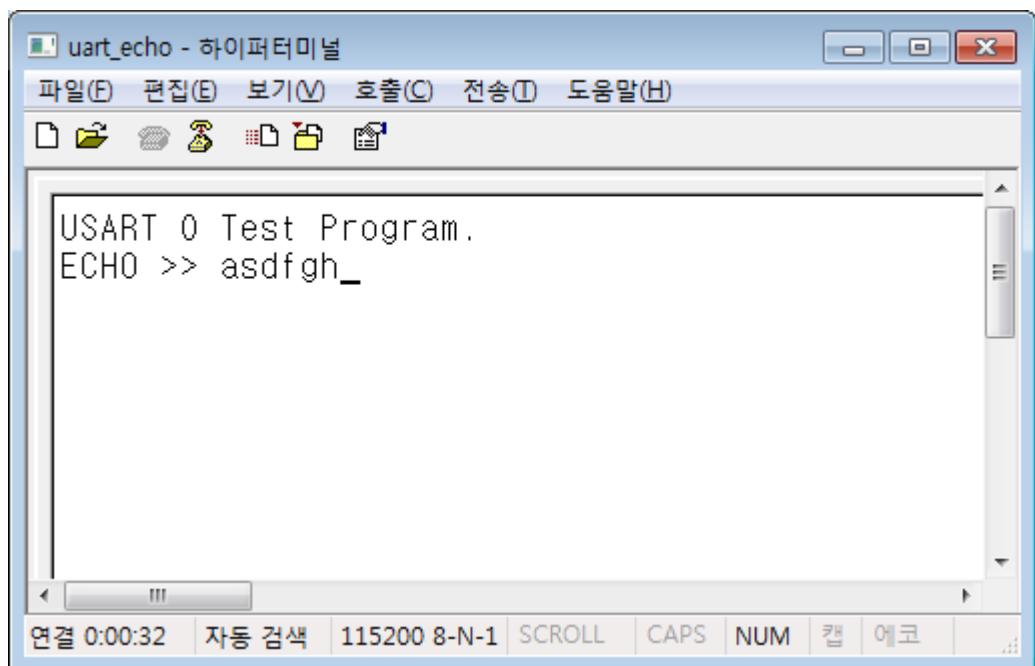


실습 2 : UART로 PC와 데이터 주고받기

● 실행 결과

- Edge Peri 보드의 UART 커넥터에 USB 케이블을 꽂은 뒤 PC와 연결
- 하이퍼터미널을 다음과 같이 설정한 다음 MCU를 리셋하여 문자열을 확인

Baud Rate	115200
패리티	No Parity
Stop Bit	1
데이터 비트수	8
흐름제어	없음





A/D 컨버터

- STM32F405의 A/D 컨버터 기능
- Text LCD
- Text LCD에 글자쓰기
- A/D 컨버터로 광센서 읽기
- A/D 컨버터로 전압값 읽기



엣지아이랩

STM32F405의 A/D 컨버터 기능

- STM32F405의 A/D 컨버터 특징
 - 최대 12비트 분해능(10-bit, 8-bit, 6-bit)
 - 인터럽트는 AD변환종료, 입력변환 종료, 아날로그 Watchdog이벤트에 의해 발생
 - 싱글/연속 변환 모드
 - 채널0~n의 자동변환을 위한 스캔모드
 - Self-calibration
 - Data alignment with in-built data coherency
 - Channel by channel programmable sampling time
 - External trigger option for both regular and injected conversion
 - 불연속적인 모드
 - Dual/Triple mode (on devices with 2 ADCs or more)
 - Configurable DMA data storage in Dual/Triple ADC mode
 - Configurable delay between conversions in Dual/Triple interleaved mode
 - ADC 공급 요건 : 2.4V to 3.6V(full speed), 1.8V(slower speed)
 - ADC 입력 범위 : $V_{ref-} \leq V_{in} \leq V_{ref+}$
 - DMA request generation during regular channel conversion

STM32F405 A/D 컨버터 레지스터

- ADC_SR (ADC status register)
 - A/D 컨버터 상태 레지스터
 - OVR (Overrun)
 - 데이터를 잃었을 때 1로 Set
 - Overrun 감지는 DMA = 1 또는 EOCS = 1 인 경우에만 활성화
 - 0 : No overrun occurred
 - 1 : Overrun has occurred
 - STRT (Regular channel start flag)
 - regular channel 변환이 시작되면 하드웨어에서 1로 설정
 - 소프트웨어에 의해 0으로 클리어
 - 0 : regular channel 변환이 시작되지 않음
 - 1 : regular channel 변환이 시작됨

31~16																	
Reserved																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reserved												OVR	STRT	JSTRT	JEOC	EOC	AWD

STM32F405 A/D 컨버터 레지스터

- ADC_SR (ADC status register)
 - JSTRT (Injected channel start flag)
 - injected channel group의 변환이 시작되면 하드웨어에서 1로 설정
 - 소프트웨어에 의해 0으로 클리어
 - 0 : injected group의 변환이 시작되지 않음
 - 1 : injected group의 변환이 시작됨
 - JEOC (Injected channel end of conversion)
 - 모든 injected channel group의 변환이 종료되면 하드웨어에서 1로 설정
 - 소프트웨어에 의해 0으로 클리어
 - 0 : 변환이 완료되지 않음
 - 1 : 변환이 완료됨

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										OVR	STRT	JSTRT	JEOC	EOC	AWD

STM32F405 A/D 컨버터 레지스터

- ADC_SR (ADC status register)
 - EOC (End of conversion)
 - 그룹채널의 변환(regular or injected)이 종료되면 하드웨어에서 1로 설정
 - 소프트웨어에 의해 0으로 클리어
 - ADC_DR레지스터를 읽어들일 때, 클리어
 - 0 : 변환이 완료되지 않음
 - 1 : 변환이 완료됨
 - AWD (Analog watchdog flag)
 - 변환된 전압이 ADC_LTR, ADC_HTR레지스터의 프로그램된 값과 서로 교차하게 되면 하드웨어에서 1로 설정
 - 0 : Analog Watchdog 이벤트가 발생되지 않음
 - 1 : Analog Watchdog 이벤트가 발생함

31~16																			
Reserved																			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Reserved														OVR	STRT	JSTRT	JEOC	EOC	AWD

STM32F405 A/D 컨버터 레지스터

- ADC_CR1 (Control register 1)
 - ADC 제어 레지스터 1
 - ADC 모듈의 동작을 설정하는 기능을 수행하는 레지스터
 - OVRIE (Overrun Interrupt enable)
 - 0 : Overrun interrupt disabled
 - 1 : Overrun interrupt enabled. An interrupt is generated when the OVR bit is set
 - RES (Resolution)
 - 분해능을 설정하는 비트들
 - 00 : 12-bit (12 ADCCLK cycles)
 - 01 : 10-bit (10 ADCCLK cycles)
 - 10 : 8-bit (11 ADCCLK cycles)
 - 11 : 6-bit (9 ADCCLK cycles)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]					

STM32F405 A/D 컨버터 레지스터

- ADC_CR1 (Control register 1)
 - AWDEN (Analog watchdog enable on regular channels)
 - 소프트웨어에 의해 클리어
 - 0 : regular channels의 Analog watchdog disable
 - 1 : regular channels의 Analog watchdog enable
 - JAWDEN (Analog watchdog enable on injected channels)
 - 소프트웨어에 의해 클리어
 - 0 : injected channels의 Analog watchdog disable
 - 1 : injected channels의 Analog watchdog enable

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				

STM32F405 A/D 컨버터 레지스터

- ADC_CR1 (Control register 1)

- DISCNUM (Discontinuous mode channel count)

- regular channels의 수를 정의하기 위해 소프트웨어에서 설정
 - regular channels은 discontinuous mode에서 변환이 될 것이며 후에, 외부 트리거를 받게 됨
 - 000 : 1 채널
 - 001 : 2채널
 - ...
 - 111 : 8 채널

- JDISCEN(Discontinuous mode on injected channels)

- injected group channels의 discontinuous mode를 enable/disable 하기 위해 소프트웨어에서 설정
 - 0 : Discontinuous mode on injected channels disabled
 - 1 : Discontinuous mode on injected channels enabled

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				

STM32F405 A/D 컨버터 레지스터

- ADC_CR1 (Control register 1)
 - DISCEN (Discontinuous mode on regular channels)
 - regular channel의 discontinuous mode를 enable/disable 하기 위해 소프트웨어에서 설정
 - 0 : Discontinuous mode on regular channels disabled
 - 1 : Discontinuous mode on regular channels enabled
 - JAUTO (Automatic Injected Group conversion)
 - automatic injected group conversion(regular group conversion 다음에 진행되는)을 enable / disable 하기 위해 소프트웨어에서 설정
 - 0 : Automatic injected group conversion disabled
 - 1 : Automatic injected group conversion enabled
 - AWDSGL (Enable the watchdog on a single channel in scan mode)
 - AWDCH[4:0] 비트에 선언된 위해 채널의 analog watchdog를 enable/disable 하기 위해 소프트웨어에서 설정
 - 0 : Analog watchdog enabled on all channels
 - 1 : Analog watchdog enabled on a single channel

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]		JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]					

STM32F405 A/D 컨버터 레지스터

- ADC_CR1 (Control register 1)
 - SCAN (Scan mode)
 - 이 스캔모드를 enable/disable 설정하기 위해 소프트웨어에서 설정
 - 스캔모드에서 ADC_SQRx or ADC_JSQRx레지스터에 선택된 입력이 컨버팅
 - 0 : Scan mode disabled
 - 1 : Scan mode enabled
 - Note : EOC또는 JEOC 인터럽트(EOICE 또는 JEOCIE비트가 설정되었을 때)는 오직 마지막채널의 변환 후 발생
 - JEOCIE (Interrupt enable for injected channels)
 - injected channels에 대한 변환 종료 인터럽트를 enable/disable 하기 위해 소프트웨어에서 설정
 - 0 : JEOC interrupt disabled
 - 1 : JEOC interrupt enabled. An interrupt is generated when the JEOC bit is set

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]					

STM32F405 A/D 컨버터 레지스터

- ADC_CR1 (Control register 1)
 - AWDIE (Analog watchdog interrupt enable)
 - analog watchdog interrupt를 enable/disable하기 위해 소프트웨어에서 설정
 - 0 : Analog Watchdog interrupt disabled
 - 1 : Analog Watchdog interrupt enabled
 - EOCIE (Interrupt enable for EOC)
 - 변환 종료 인터럽트를 enable / disable 하기 위해 소프트웨어에서 설정
 - 0 : EOC interrupt disabled
 - 1 : EOC interrupt enabled. An interrupt is generated when the EOC bit is set

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]					

STM32F405 A/D 컨버터 레지스터

- ADC_CR1 (Control register 1)

- AWDCH[4:0] (Analog watchdog channel select bits)

▪ 소프트웨어에서 설정하며, Analog Watchdog의 채널을 선택

- 00000 : ADC analog input Channel0

- 00001 : ADC analog input Channel1

...

- 01111 : ADC analog input Channel15

- 10000 : ADC analog input Channel16

- 10001 : ADC analog input Channel17

- 10010 : ADC analog input Channel18

- Other values reserved

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				

STM32F405 A/D 컨버터 레지스터

- ADC_CR2 (Control register 2)
 - ADC 제어 레지스터 2
 - ADC 모듈의 동작을 설정하는 기능을 수행하는 레지스터
 - SWSTART (Start conversion of regular channels)
 - 변환을 시작하기 위해 소프트웨어에서 설정되며, 변환이 시작될 때 하드웨어에 의해 클리어
 - 0 : Reset state
 - 1 : Starts conversion of regular channels
 - EXten (External trigger enable for regular channels)
 - 외부 트리거의 polarity 와 regular group 의 트리거를 정의하기 위해 소프트웨어에서 설정
 - 00 : Trigger detection disabled
 - 01 : Trigger detection on the rising edge
 - 10 : Trigger detection on the falling edge
 - 11 : Trigger detection on both the rising and falling edges

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	SW START	EXTEN		EXTSEL[3:0]				Res	JSW START	JEXTEN		JEXTSEL[3:0]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				ALIGN	EOCS	DDS	DMA	Reserved				CONT	ADON		

STM32F405 A/D 컨버터 레지스터

- ADC_CR2 (Control register 2)

- EXTSEL[3:0] (External Event Select for regular group)

외부트리거를 regular group의 변환의 시작점으로 사용하기 위해 선언

- | | |
|-------------------------------|--------------------------|
| - 0000 : Timer 1 CC1 event , | 0001 : Timer 1 CC2 event |
| - 0010 : Timer 1 CC3 event , | 0011 : Timer 2 CC2 event |
| - 0100 : Timer 2 CC3 event , | 0101 : Timer 2 CC4 event |
| - 0110 : Timer 2 TRGO event , | 0111 : Timer 3 CC1 event |
| - 1000 : Timer 3 TRGO event , | 1001 : Timer 4 CC4 event |
| - 1010 : Timer 5 CC1 event , | 1011 : Timer 5 CC2 event |
| - 1100 : Timer 5 CC3 event , | 1101 : Timer 8 CC1 event |
| - 1110 : Timer 8 TRGO event, | 1111 : EXTI line11 |

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	SW START	EXTEN		EXTSEL[3:0]				Res	JSW START	JEXTEN		JEXTSEL[3:0]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON

STM32F405 A/D 컨버터 레지스터

- ADC_CR2 (Control register 2)
 - JSWSTART (Start Conversion of injected channels)
 - 변환을 시작하기 위해 소프트웨어에서 설정되며, 변환이 시작될 때, 하드웨어에 의해 클리어
 - ADON 비트가 1 일때만 Set 가능(그렇지 않으면 컨버전이 시작되지 않음)
 - 0 : Reset state
 - 1 : Starts conversion of injected channels
 - JEXTEN (External trigger enable for injected channels)
 - 외부 트리거의 polarity 와 injected group 의 트리거를 정의하기 위해 소프트웨어에서 설정
 - 00 : Trigger detection disabled
 - 01 : Trigger detection on the rising edge
 - 10 : Trigger detection on the falling edge
 - 11 : Trigger detection on both the rising and falling edge

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	SW START	EXTEN		EXTSEL[3:0]				Res	JSW START	JEXTEN		JEXTSEL[3:0]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				ALIGN	EOCS	DDS	DMA	Reserved				CONT	ADON		

STM32F405 A/D 컨버터 레지스터

- ADC_CR2 (Control register 2)

- JEXTSEL[3:0] (External event select for injected group)

▪ 외부트리거를 injected group의 변환의 시작점으로 사용하기 위해 선언

- | | |
|------------------------------|---------------------------|
| - 0000 : Timer 1 CC4 event , | 0001 : Timer 1 TRGO event |
| - 0010 : Timer 2 CC1 event , | 0011 : Timer 2 TRGO event |
| - 0100 : Timer 3 CC2 event , | 0101 : Timer 3 CC4 event |
| - 0110 : Timer 4 CC1 event , | 0111 : Timer 4 CC2 event |
| - 1000 : Timer 4 CC3 event , | 1001 : Timer 4 TRGO event |
| - 1010 : Timer 5 CC4 event , | 1011 : Timer 5 TRGO event |
| - 1100 : Timer 8 CC2 event , | 1101 : Timer 8 CC3 event |
| - 1110 : Timer 8 CC4 event , | 1111 : EXTI line15 |

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	SW START	EXTEN		EXTSEL[3:0]				Res	JSW START	JEXTEN		JEXTSEL[3:0]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				ALIGN	EOCS	DDS	DMA	Reserved				CONT	ADON		

STM32F405 A/D 컨버터 레지스터

- ADC_CR2 (Control register 2)
 - ALIGN (Data Alignment)
 - ADC_DR 레지스터의 변환 값이 저장되는 정렬방식을 정의
 - 0 : 우정렬
 - 1 : 좌정렬
 - EOCS (End of conversion selection)
 - 소프트웨어에 의해 1로 Set되고 0으로 클리어
 - 0 : The EOC bit is set at the end of each sequence of regular conversions. Overrun detection is enabled only if DMA=1
 - 1 : The EOC bit is set at the end of each regular conversion. Overrun detection is enabled

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	SW START	EXTEN		EXTSEL[3:0]				Res	JSW START	JEXTEN		JEXTSEL[3:0]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				ALIGN	EOCS	DDS	DMA	Reserved				CONT	ADON		

STM32F405 A/D 컨버터 레지스터

- ADC_CR2 (Control register 2)
 - DDS (DMA disable selection)
 - 소프트웨어에 의해 1로 Set되고 0으로 클리어
 - 0 : No new DMA request is issued after the last transfer (as configured in the DMA controller)
 - 1 : DMA requests are issued as long as data are converted and DMA=1
 - DMA (Direct memory access mode)
 - DMA에 대한 내용은 데이터 시트를 참조
 - 0 : DMA mode disabled
 - 1 : DMA mode enabled

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	SW START	EXTEN		EXTSEL[3:0]				Res	JSW START	JEXTEN		JEXTSEL[3:0]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON	

STM32F405 A/D 컨버터 레지스터

- ADC_CR2 (Control register 2)
 - CONT (Continuous conversion)
 - 소프트웨어에서 1로 설정된 후, 0으로 클리어 할 때까지는 연속변환모드를 유지하며, 0으로 클리어 되면 싱글 변환 모드로 전환
 - 0 : Single conversion mode
 - 1 : Continuous conversion mode
 - ADON (A/D Converter ON/OFF)
 - A/D 컨버터를 ON/OFF
 - 0 : Disable ADC conversion and go to power down mode
 - 1 : Enable ADC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	SW START	EXTEN		EXTSEL[3:0]				Res	JSW START	JEXTEN		JEXTSEL[3:0]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				ALIGN	EOCS	DDS	DMA	Reserved				CONT	ADON		

STM32F405 A/D 컨버터 레지스터

- ADC_SMPR1 (sample time register 1)
 - ADC 샘플 타임 설정 레지스터 1
 - SMPx[2:0] (Channel x Sample time selection)
 - 각각의 채널에 대한 Sample time 선택하기 위해 소프트웨어로 설정
 - 샘플링 중에는 이 값은 변경되지 않음
 - 000 : 3 cycles
 - 001 : 15 cycles
 - 010 : 28 cycles
 - 011 : 56 cycles
 - 100 : 84 cycles
 - 101 : 112 cycles
 - 110 : 144 cycles
 - 111 : 480 cycles

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					SMP18[2:0]			SMP17[2:0]			SMP16[2:0]			SMP15[2:1]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP15 [0]	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		

STM32F405 A/D 컨버터 레지스터

- ADC_SMPR2 (sample time register 2)
 - ADC 샘플 타임 설정 레지스터 2
 - SMPx[2:0] (Channel x Sample time selection)
 - 각각의 채널에 대한 Sample time 선택하기 위해 소프트웨어로 설정
 - 샘플링 중에는 이 값은 변경되지 않음
 - 000 : 3 cycles
 - 001 : 15 cycles
 - 010 : 28 cycles
 - 011 : 56 cycles
 - 100 : 84 cycles
 - 101 : 112 cycles
 - 110 : 144 cycles
 - 111 : 480 cycles

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:1]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP6[0]	SMP5[2:0]			SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]		

STM32F405 A/D 컨버터 레지스터

- ADC_SQR1 (regular sequence register 1)
 - ADC regular sequence 설정 레지스터 1
 - L[3:0] (Regular channel sequence length)
 - regular channel 연속 전환을 해야 하는 컨버전 횟수를 설정
 - 0000 : 1 conversion
 - 0001 : 2 conversions
 - ...
 - 1111 : 16 conversions
 - SQn[4:0] (nth conversion in regular sequenc)
 - n번째 컨버팅 차례를 가지는 채널 넘버를 이 비트에 소프트웨어로 선언
 - SQn[4:0] (nth conversion in regular sequenc) nth conversion in regular sequence

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								L[3:0]				SQ16[4:1]			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ16[0]	SQ15[4:0]				SQ14[4:0]				SQ13[4:0]						

STM32F405 A/D 컨버터 레지스터

- ADC_SQR2 (regular sequence register 2)
 - ADC regular sequence 설정 레지스터 2
 - SQn[4:0] (nth conversion in regular sequenc)
 - n번째 컨버팅 차례를 가지는 채널 넘버를 이 비트에 소프트웨어로 선언
 - SQn[4:0] (nth conversion in regular sequenc) nth conversion in regular sequence

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ12[4:0]				SQ11[4:0]				SQ10[4:1]					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ10[0]	SQ9[4:0]				SQ8[4:0]				SQ7[4:0]						

STM32F405 A/D 컨버터 레지스터

- ADC_SQR3 (regular sequence register 3)
 - ADC regular sequence 설정 레지스터 3
 - SQn[4:0] (nth conversion in regular sequenc)
 - n번째 컨버팅 차례를 가지는 채널 넘버를 이 비트에 소프트웨어로 선언
 - SQn[4:0] (nth conversion in regular sequenc) nth conversion in regular sequence

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		SQ6[4:0]				SQ5[4:0]				SQ4[4:1]					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SQ4[0]	SQ3[4:0]				SQ2[4:0]				SQ1[4:0]						

STM32F405 A/D 컨버터 레지스터

- ADC_DR (regular data register) : ADC 데이터 레지스터
 - DATA[15:0] (Regular data)
 - 읽기전용이며, regular channels의 변환결과가 저장
- 이 외에도 ADC_JOFRx레지스터, ADC_HTR, ACD_LTR레지스터, ADC_SQR1, ADC_SQR2, ADC_SQR3, ADC_SQR4, ADC_JSQR, ADC_JDRx(x=1..4), ADC_CSR, ADC_CCR, ADC_CDR 레지스터가 존재(나머지는 데이터 시트를 참조)

31~16															
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															

STM32F405의 A/D 컨버터 기능

- STM32F405 A/D 컨버터의 동작 설정

- ADC_CR1, 2레지스터, ADC_SQR1레지스터로 A/D관련 모드 및 A/D 신호를 입력받을 채널수 선택, 입력 데이터 정렬 방법 등을 설정
- ADC_SQRn, ADC_SMPRn레지스터로 사용할 채널설정 및 샘플시간을 설정하고, 변환 랭크(순서)를 결정
- ADC_CR2레지스터의 ADON비트를 세팅해서 ADC를 Enable
- 변환을 하기 전에, Calibration을 초기화 한다. (ADC_CR2의 CAL, RSTCAL)
- ADC_CR2의 SWSTART비트를 체크해서 컨버전을 시작
- 상태레지스터에서 변환이 완료되면, ADC_DR레지스터에서 ADC 값을 가져옴
- ADC_CR1레지스터의 EOCIE를 체크해서 인터럽트를 발생할 수 있음

Text LCD

- Text LCD(Character LCD)
 - LCD화면에 정해진 형태의 문자를 정해진 개수만큼 표시할 수 있도록 만들어진 LCD 디스플레이 장치
 - 각종 임베디드 장치들에서 널리 사용
 - 대부분의 마이크로컨트롤러들과 TEXT LCD는 그대로 연결할 수 있도록 신호를 제공
 - 마이크로 컨트롤러 개발 환경에서도 TEXT LCD 관련 함수를 제공
- 실습에 사용되는 Text LCD
 - 16문자 * 2라인의 표시부
 - Backlight 기능 포함

Text LCD

- 인터페이스 커넥터 핀 기능

핀	Signal Name	기 능
1	VSS	전원 GND
2	VDD	전원 +5VDC
3	VEE	Contrast 제어 전압레벨 (VDD-VEE = 13.5 ~ 0V)
4	RS	Register Select (0 = instruction, 1 = data)
5	R/W	Read/Write (0 = FPGA -> LCD, 1 : FPGA <- LCD)
6	E	Enable Signal for read/write LCD
7	DB0 (LSB)	DATA
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7 (MSB)	
15	A	+LED (backlight LED용 전원 +4.4 ~ 4.7V)
16	K	-LED (backlight LED용 전원 GND)

Text LCD

- Text LCD(Character LCD) 구조
 - TEXT LCD는 보통 LCD 표시부와 LCD 제어부를 하나로 하여 LCD 모듈로 시판
 - LCD제어기 구성
 - 명령(Instruction)과 데이터(Data)를 위한 2개의 레지스터
 - BF (Busy Flag)
 - AC(Address Counter)
 - 문자발생램(CGRAM)
 - 문자발생롬(CGROM)
 - 데이터표시램(DDRAM)

Text LCD

- Text LCD(Character LCD) 구조
 - 내장 레지스터
 - 명령레지스터(IR)
 - DDRAM과 CGRAM에 대한 어드레스와 클리어, 커서시프트 등 제어 명령을 보유
 - 데이터레지스터(DR)
 - DDRAM과 CGRAM에 쓴 데이터나 읽은 데이터를 일시적으로 저장
 - 레지스터들은 RS(4번핀)과 R/W(5번핀)을 사용하여 선택

RS, R/W 신호에 따른 TEXT LCD 제어기 레지스터 선택 방법

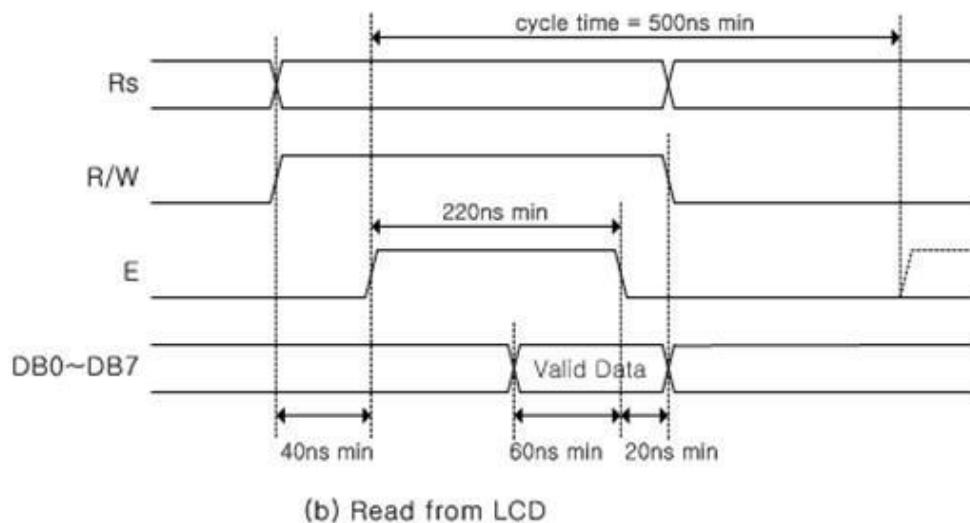
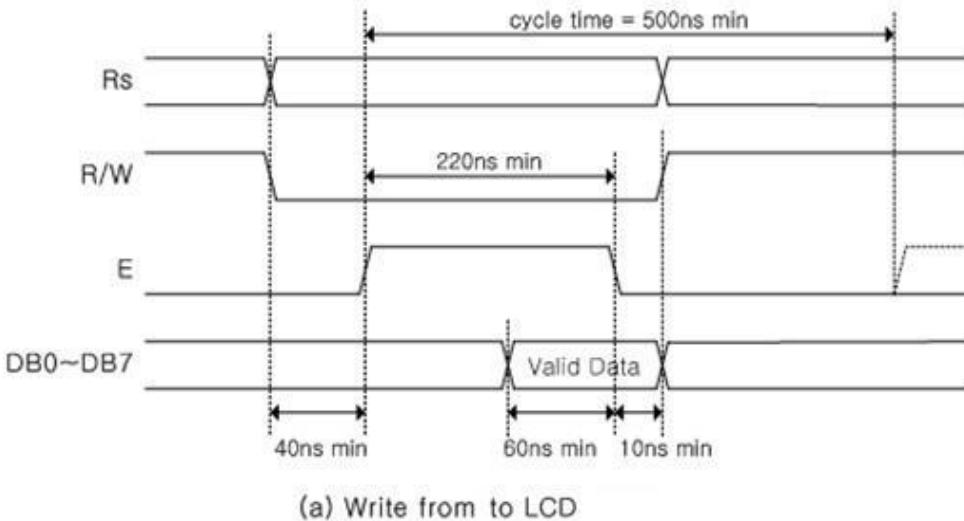
RS	R/W	레지스터 접근
0	0	IR쓰기(각종 제어 명령 쓰기)
0	1	BF읽기, AC읽기
1	0	DR쓰기
1	1	DR읽기

Text LCD

- Text LCD(Character LCD) 구조
 - BF (Busy Flag)
 - 1이면 LCD의 콘트롤러가 동작 중으로 명령 수행 불능
 - 0이면 다음 명령 수행 가능 표시
 - AC(Address Counter)
 - DDRAM과 CGRAM의 주소를 지정할 때 사용
 - IR에 주소 정보를 쓰면 주소 정보가 AC로 전송
 - DDRAM/DDROM에 데이터를 쓰면 AC는 자동으로 +1 혹은 -1이 됨
 - 문자발생램(CGRAM)
 - 사용자가 자유로이 문자를 만들 때 사용하는 램
 - 5x7은 8개, 5x10은 4개 만들어 저장 가능
 - 문자발생롬(CGROM)
 - 5x7, 5x10 도트의 문자를 내장
 - 데이터표시램(DDRAM)
 - 80x8비트 용량으로 80개의 8비트 아스키(ASCII)코드를 저장
 - 0x00-0F 주소가 LCD 1행의 1-16번째, 0x40-4F 주소가 LCD 2행의 1-16번째 문자로 표시

Text LCD

- Text LCD 제어 신호 동작 타이밍



Text LCD

- Text LCD 제어 명령

LCD 모듈 표시 제어 명령

기능	제어신호		제어명령							
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Home	0	0	0	0	0	0	0	0	1	0
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	0	0
Function Set	0	0	0	0	1	DL	N	F	0	0
Set CG RAM address	0	0	0	1	CG RAM address					
Set DD RAM address	0	0	1	DD RAM address						
Read busy flag and address	0	1	BF	Address Counter						
Data write to CG RAM or DD RAM	1	0	Write address							
Data read from CG RAM or DD RAM	1	1	Read address							

Text LCD

- Text LCD 제어 명령
 - CLEAR DISPLAY
 - 디스플레이 상태를 소거하고 커서를 Home 위치로
 - Return Home
 - DD RAM의 내용은 변경하지 않고 커서만을 Home 위치로
 - Entry Mode SET
 - 데이터를 Read하거나 Write 할 경우에 커서의 위치를 증가시킬 것인가 (I/D=1) 감소시킬 것인가 (I/D=0)를 결정
 - 이때 화면을 이동 할 것인지 (S=1) 아닌지 (S=0)를 결정
 - Display ON/OFF Control
 - 화면 표시를 ON/OFF 하거나(D) 커서를 ON/OFF 하거나(C) 커서를 깜박이게 할 것인지(B)의 여부를 지정
 - Cursor or Display Shift
 - 화면(S/C=1) 또는 커서(S/C=0)를 오른쪽(R/L=1) 또는 왼쪽(R/L=0)으로 이동

Text LCD

- Text LCD 제어 명령
 - Function SET
 - 인터페이스에서 데이터의 길이를 8비트(DL=1) 또는 4비트(DL=0)로 지정
 - 화면 표시 행수를 2행(N=1) 또는 1행(N=0)으로 지정
 - 문자의 폰트를 5x10 도트(F=1) 또는 5x7 도트(F=0)로 지정
 - Set CG RAM Address
 - Character Generator RAM의 어드레스를 지정
 - 지정 이후에 송수신 하는 데이터는 CG RAM의 데이터
 - Set DD RAM Address
 - Display Data RAM의 어드레스를 지정
 - 지정 이후에 송수신하는 데이터는 DD RAM의 데이터
 - Read Busy Flag & Address
 - LCD 모듈이 내부 동작중임을 나타내는 Busy Flag(BF) 및 어드레스 카운터의 내용을 read

Text LCD

- Text LCD 제어 명령

- Text LCD DDRAM Address

- DDRAM은 표시될 각 문자의 ASCII 코드 데이터가 저장되어 있는 메모리
 - 모두 80개의 번지가 있으며, 화면의 각 행과 열의 위치에는 고유한 어드레스 값이 부여되어 있음
 - 각 행과 행 사이의 어드레스가 연속하여 있지 않으므로 주의해야 함

표시 문자의 위치에 대한 DD RAM의 어드레스

구분	1	2	3	4	13	14	15	16
Line1	00	01	02	03	0D	0E	0F	10
Line2	40	41	42	43	4D	4E	4F	50

Text LCD

ASCII 도형문자 종류 및 코드 값

구분	00H	10H	20H	30H	40H	50H	60H	70H	80H	90H
0	사용자 정의 영역	미사용 영역		0	@	P	`	p	미사용 영역	
1			!	1	A	Q	a	q		
2			"	2	B	R	b	r		
3			#	3	C	S	c	s		
4			\$	4	D	T	d	t		
5			%	5	E	U	e	u		
6			&	6	F	V	f	v		
7			'	7	G	W	g	w		
8			(8	H	X	h	x		
9)	9	I	Y	l	y		
A			*	:	J	Z	j	z		
B			+	;	K	[k	{		
C			,	<	L	¥	l			
D			-	=	M]	m	}		
E			.	>	N	^	n	→		
F			/	?	O	_	o	←		

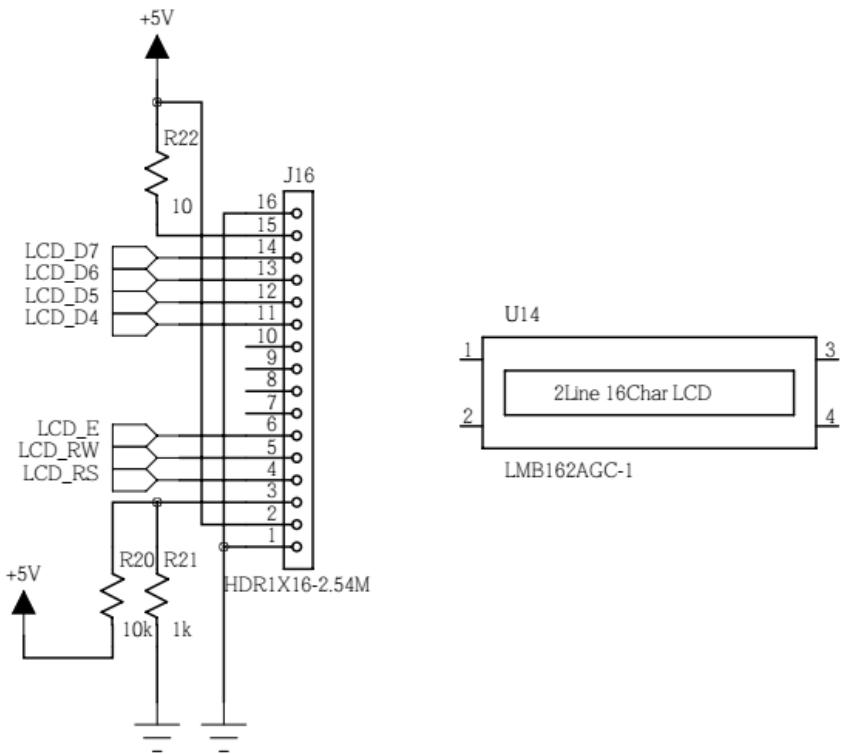
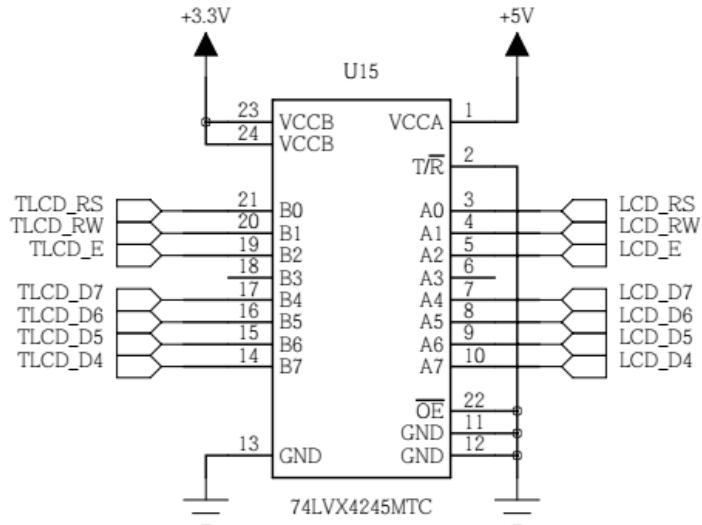
실습 1 : Text LCD에 글자 쓰기

- 실습 개요
 - STM32F405의 GPIO에 TEXT LCD를 연결하고, LCD 화면에 미리 작성된 문장 ("Hello! MCU World !!")을 표시
 - AVR 개발 환경에서 제공하는 TEXT LCD 관련 함수를 이해하면 쉽게 프로그램을 작성할 수 있음
- 실습 목표
 - TEXT LCD의 동작 원리 이해
 - AVR 개발환경에서 제공하는 TEXT LCD 관련 함수 이해
 - TEXT LCD 제어 프로그램 방법 습득

실습 1 : Text LCD에 글자 쓰기

- 사용 모듈

- TEXT LCD 모듈 회로



실습 1 : Text LCD에 글자 쓰기

- 구동 프로그램 : 사전지식
 - Cortex-M4 개발 환경에서 TEXT LCD 관련 라이브러리 함수를 제공
 - lcd.c 라는 파일에 포함
 - EWARM에서 프로젝트를 Build할 때 이 lcd.c 파일, lcd.h 파일, 그리고 lcdconf.h 파일을 함께 포함시켜야 함
 - TEXT LCD용 라이브러리 함수
 - lcdInit : TEXT LCD 초기화
 - lcdGotoXY(unsigned char x, unsigned char y) : TEXT LCD의 커서를 원하는 위치로 이동
 - lcdDataWrite(unsigned char data) : TEXT LCD에 하나의 문자 출력
 - lcdPrintData(char* data, unsigned char nBytes) : TEXT LCD에 nBytes 길이의 문자열을 출력
 - lcdClear() : TEXT LCD의 화면을 지움

실습 1 : Text LCD에 글자 쓰기

- 구동 프로그램 : 사용 포트와 핀 선언
 - 사용할 포트를 선언하는 헤더화일 : lcdconf.h
 - 이 예제에서는 MCU 모듈의 B 포트와 C 포트를 사용
 - 다음은 lcdconf.h 파일의 일부분

Text LCD
제어포트 선언

```
#ifndef LCD_CTRL_PORT
    #define LCD_CTRL_PORT_CLK    RCC_AHB1Periph_GPIOB
    #define LCD_CTRL_PORT        GPIOB
    #define LCD_CTRL_RW          GPIO_Pin_0
    #define LCD_CTRL_RS          GPIO_Pin_1
    #define LCD_CTRL_E           GPIO_Pin_2
#endif

#ifndef LCD_DATA_POUT
    #define LCD_DATA_PORT_CLK    RCC_AHB1Periph_GPIOC
    #define LCD_DATA_PORT        GPIOC
    #define LCD_DATA_4BIT
    #ifdef LCD_DATA_4BIT
        #define LCD_DATA_BUS        0x00F0
    #else
        #define LCD_DATA_BUS        0x00FF
    #endif
#endif
```

Text LCD
데이터포트 선언

실습 1 : Text LCD에 글자 쓰기

- 구동 프로그램 : 사용 포트와 핀 선언
 - 사용할 포트를 선언하는 헤더파일 : lcdconf.h
 - Text LCD 제어 포트 선언

매크로 선언	매크로 상수 의미	예
#define LCD_CTRL_PORT_CLK	포트 클럭	RCC_AHB1Periph_GPIOB
#define LCD_CTRL_PORT	포트 이름	GPIOB
#define LCD_CTRL_RS	TextLCD RS핀	0 ~15 의 값
#define LCD_CTRL_RW	TextLCD RW핀	0 ~ 15 의 값
#define LCD_CTRL_E	TextLCD E핀	0 ~ 15 의 값

- Text LCD 데이터 포트 선언

매크로 선언	매크로 상수 의미	사용 예
#define LCD_DATA_PORT_CLK	포트 클럭	RCC_AHB1Periph_GPIOC
#define LCD_DATA_PORT	포트 이름	GPIOC
#define LCD_DATA_BUS	데이터 버스	0x00F0 or 0x00FF

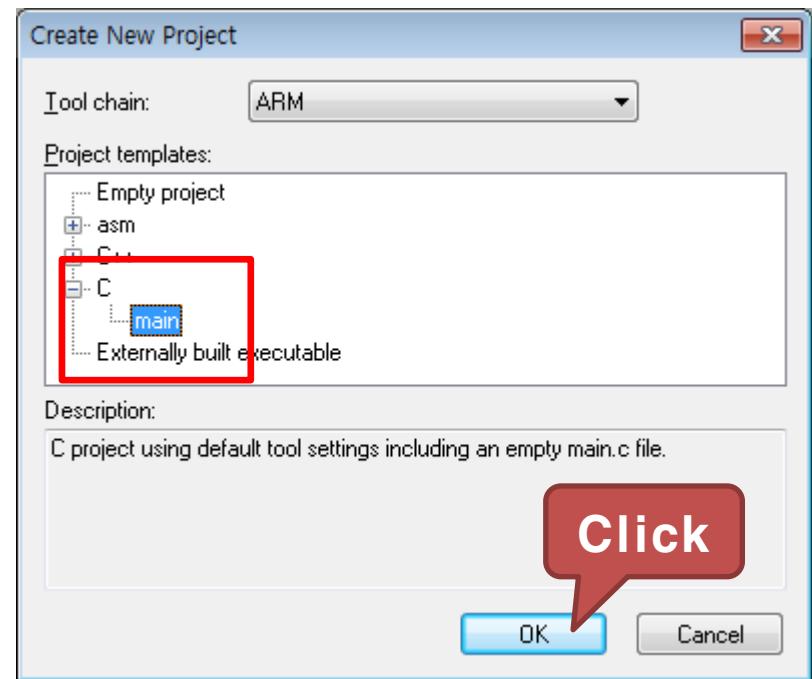
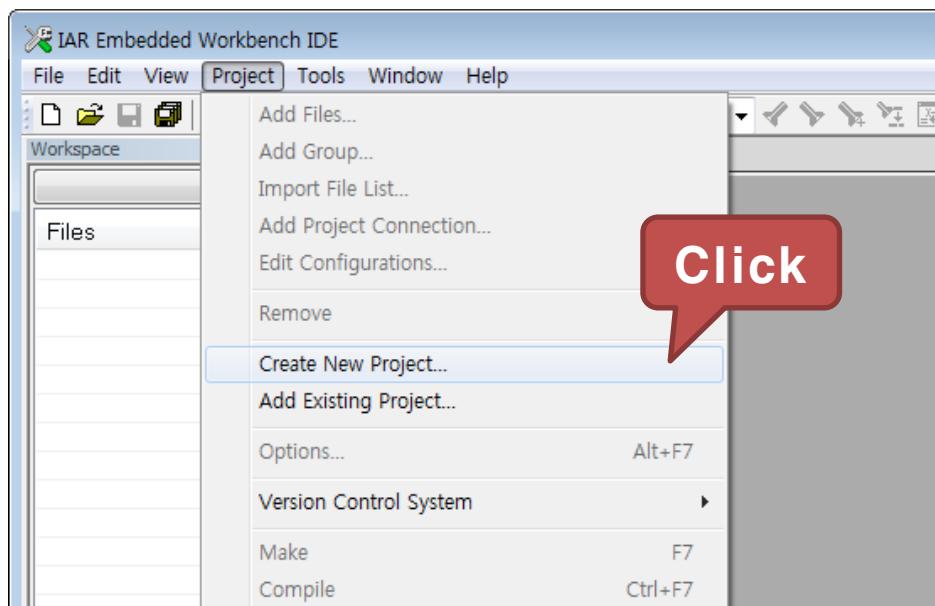
- Text LCD 데이터 포트 4비트 사용시 선언

매크로 선언	매크로 상수 의미	사용 예
#define LCD_DATA_4BIT	데이터 제어 비트 설정	4비트 사용시에만 선언

실습 1 : Text LCD에 글자 쓰기

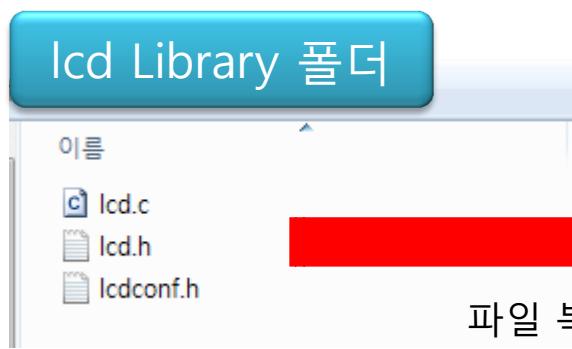
● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch9” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “textLcd”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1 : Text LCD에 글자 쓰기

- 구동 프로그램 : EWARM에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - lcd.c, lcd.h, lcdconf.h 파일은 “Cortex Example\Projects\library\lcd” 폴더에 존재
 - 라이브러리 파일은 EWARM에서 새 프로젝트를 생성한 후 생성된 폴더에 복사
 - 예를 들어 textlcd라는 프로젝트를 생성하면 “textlcd”라는 프로젝트용 폴더가 생성되고 이 폴더안에 소스 코드 및 빌드 결과물이 저장
 - 즉, lcd.c, lcd.h, lcdconf.h 파일을 “textlcd” 폴더에 복사해 넣으면 됨 (라이브러리 파일은 main.c 파일과 같은 경로에 있어야 함)



파일 복사

textLCD 프로젝트 폴더

이름	수정한 날짜	유형
lcd.c	2016-10-05 오전...	C Source File
lcd.h	2016-10-05 오전...	H 파일
lcdconf.h	2016-10-05 오전...	H 파일
stm32f4xx_conf.h	2016-09-28 오전...	H 파일
stm32f4xx_it.c	2016-09-28 오전...	C Source File
stm32f4xx_it.h	2016-09-28 오전...	H 파일
stm32f405xG.icf	2013-06-26 오후...	ICF 파일
system_stm32f4xx.c	2016-09-28 오전...	C Source File

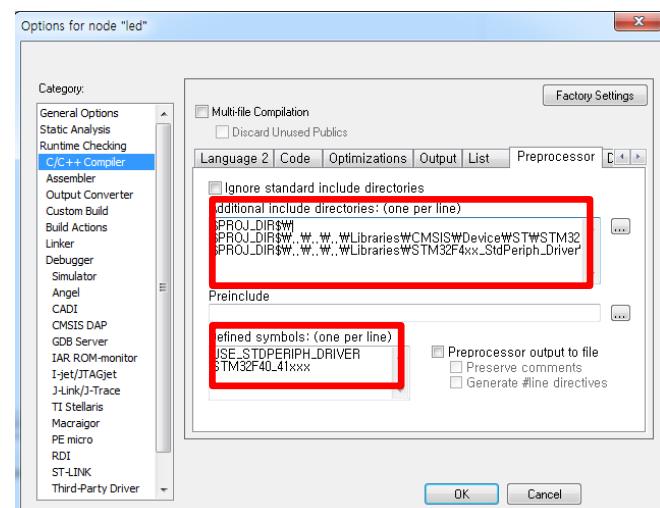
실습 1 : Text LCD에 글자 쓰기

- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch9\textLcd	system_stm32f4xx.c
Cortex_Example\Projects\ch9\textLcd	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Projects\ch9\textLcd	lcd.c

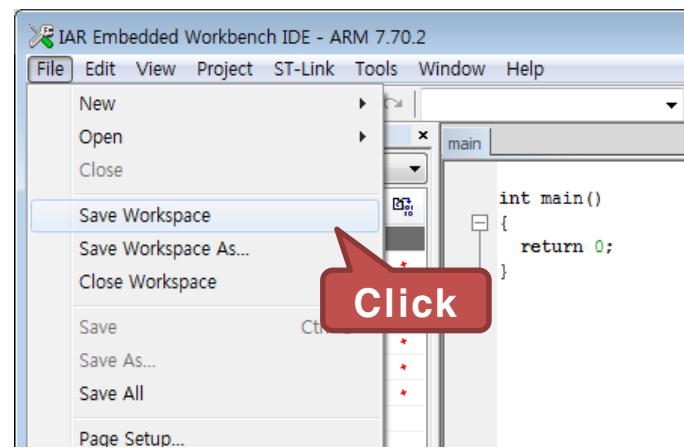
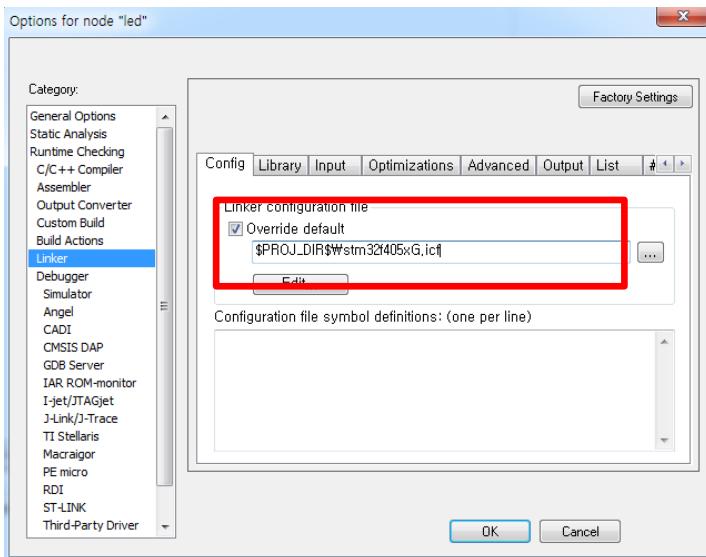
실습 1 : Text LCD에 글자 쓰기

- 프로젝트 옵션 설정
 - “Projects” 탭에서 “Options...” 선택(Alt + F7)
 - General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
 - C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1 : Text LCD에 글자 쓰기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1 : Text LCD에 글자 쓰기

- 구동 프로그램

- main.c 코드 작성

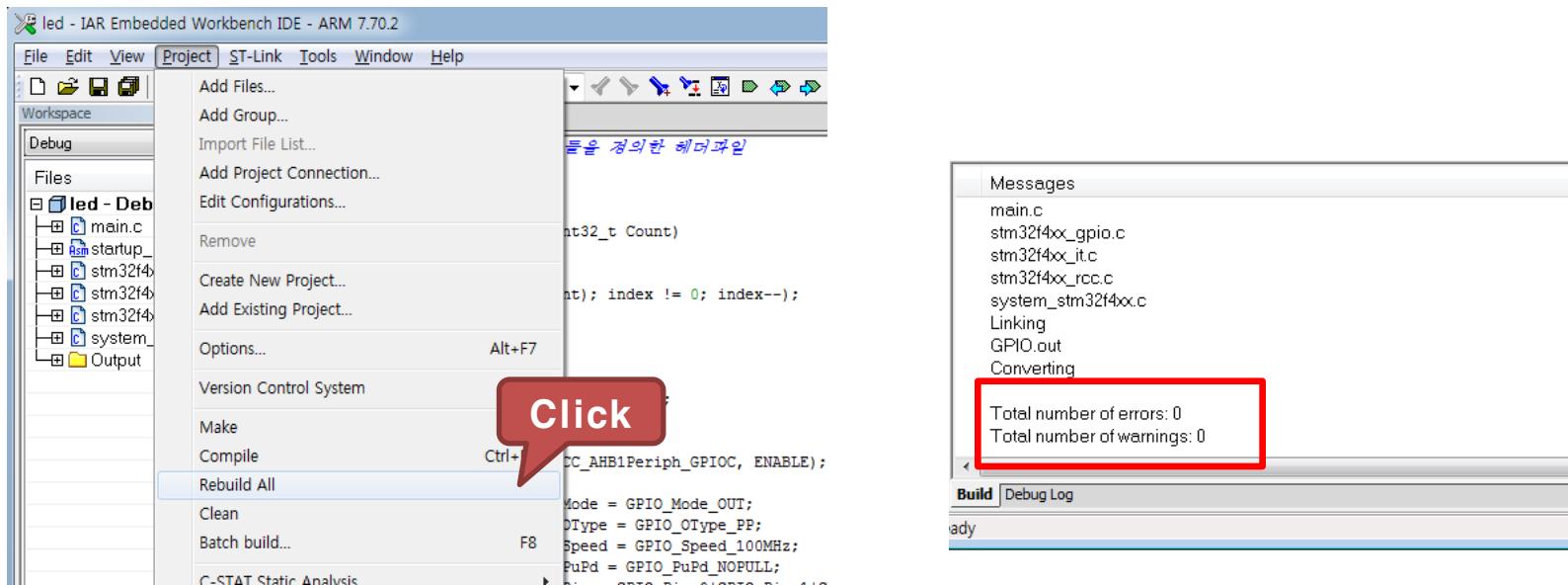
```
#include "stm32f4xx.h"
#include "lcd.h"          // Text LCD를 사용하기 위한 헤더 파일

int main(void) {
    lcdInit();           // Text LCD를 초기화
    lcdGotoXY(0,0);      // 커서위치를 첫번째 줄 첫번째 칸으로 이동
    // 첫번째 매개변수는 행을, 두번째 매개변수는 열을 의미
    lcdDataWrite('H');   // 'H'를 출력
    lcdDataWrite('e');   // 'e'를 출력
    lcdDataWrite('l');   // 'l'를 출력
    lcdDataWrite('l');   // 'l'를 출력
    lcdDataWrite('o');   // 'o'를 출력
    lcdGotoXY(3,1);      // 커서위치를 두번째 줄 네번째 칸으로 이동
    lcdPrintData("MCU World !!",12); // 문자열을 출력
    while (1) {}
}
```

실습 1 : Text LCD에 글자 쓰기

● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 textLcd 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 1 : Text LCD에 글자 쓰기

- 실행 결과
 - Text LCD 모듈에 "Hello MCU World!!" 문자열이 출력
 - 단, 처음 구동되기까지 약간의 시간(5초~1분)이 소요될 수 있음



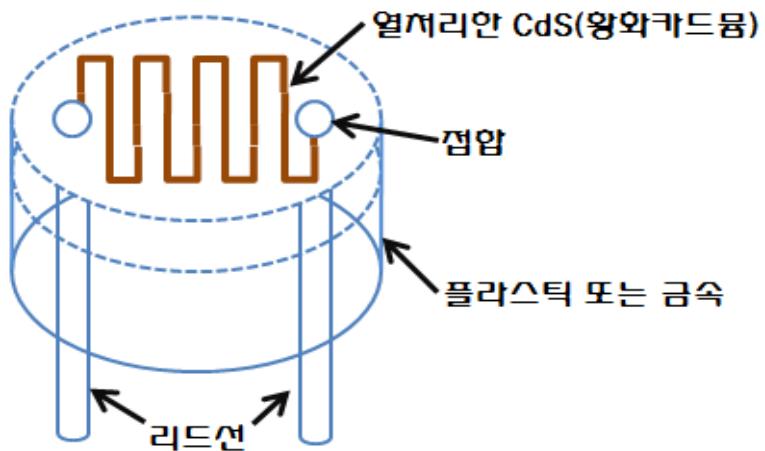
실습 2 : A/D 컨버터로 광센서 읽기

- 실습 개요
 - STM32F405 의 A/D 컨버터 기능을 이용하여 광 센서(CdS)로부터 밝기 정보를 읽어내어 Text LCD에 출력
 - CdS : 빛의 세기에 따라 내부 저항이 변하는데 밝은 곳에서는 내부 저항이 작아지는 광 가변 저항기
- 실습 목표
 - STM32F405 A/D 컨버터의 동작 원리 이해
 - A/D 컨버터 제어 방법 습득(레지스터 설정)
 - CdS 동작 원리 이해

실습 2 : A/D 컨버터로 광센서 읽기

- CdS

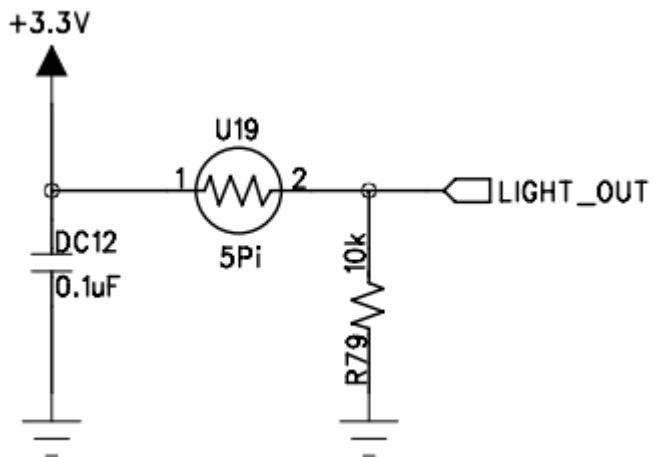
- 빛의 세기에 따라 내부 저항이 변하는데 밝은 곳에서는 내부 저항이 작아지는 광 가변 저항
- 저항은 옴의 법칙에 따라 전압에 비례하고 전류에 반비례($R = V/I$)
- 밝은 곳에서는 저항이 작아져 전류가 증가하고 어두운 곳에서는 전류가 감소



실습 2 : A/D 컨버터로 광센서 읽기

● 사용 모듈

- 센서 모듈의 광센서 회로
 - LIGHT_OUT 신호가 Sensor 신호선 포트의 CdS에 매핑
 - 빛이 들어오면 세기에 따라 저항 값이 변화



실습 2 : A/D 컨버터로 광센서 읽기

● 구동 프로그램 : 사전 지식

- A/D 컨버터 기능을 이용하여 CdS에서 출력되는 빛의 세기에 대한 아날로그 신호를 받아 디지털로 변환한 뒤, 이를 Text LCD에 표시
- Delay 함수를 이용하여 0.5초 간격으로 데이터를 읽도록 함
- A/D 컨버터의 설정 방법
 - A/D 컨버터의 입력채널 결정 : 0번 채널 사용
 - 입력 데이터의 정렬 방법 : 데이터 정렬은 디폴트인 우정렬
 - 컨버팅 모드 : 싱글 컨버팅 모드
 - ADC 클럭 : APB2 클럭의 2분주 ($84\text{MHz} / 2 = 42\text{MHz}$)
 - 컨버팅 사이클 : 3사이클(3사이클 + 12사이클(고정시간) = 15사이클)
 - 샘플링 시간 : $23.8\text{ns}(1/42\text{MHz}) * 15\text{사이클} = 0.357\mu\text{s}$
 - 인터럽트 : 사용하지 않음
- A/D 컨버팅 제어
 - 설정을 마친 후, ADC1을 Enable, 안정적인 A/D를 위해 Calibration을 수행
 - ADC_CR2의 ADON 비트를 1로 설정하여 컨버팅 시작
 - 상태레지스터에서 변환 종료 비트를 확인하여 변환 종료 비트가 설정될 때까지 대기
 - ADC_DR 레지스터에서 변환 데이터를 읽어 옴

실습 2 : A/D 컨버터로 광센서 읽기

- ADC 라이브러리
 - STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
 - ADC_InitTypeDef 구조체

USART_InitTypeDef구조체	설 명
uint32_t ADC_Resolution	ADC의 분해능을 설정한다. (ADC_CR1의 RES 비트)
FunctionalState ADC_ScanConvMode	SCAN모드를 Enable/Disable한다. (ADC_CR1의 SCAN 비트)
FunctionalState ADC_ContinuousConvMode	Discontinuous mode(regular channels)를 Enable / Disable한다. (ADC_CR1의 DISCEN 비트)
uint32_t ADC_ExternalTrigConvEdge	AD컨버팅을 시작하기 위한 신호의 종류(edge)를 선택한다. (ADC_CR2의 EXTEN 비트)
uint32_t ADC_ExternalTrigConv	AD컨버팅을 시작하기 위한 신호를 선택한다. (ADC_CR2의 EXTSEL[3:0] 비트)
uint32_t ADC_DataAlign	AD데이터 정렬 방법 설정(ADC_CR2의 ALIGN 비트)
uint8_t ADC_NbrOfChannel	컨버전 할 Regular채널의 수를 선택한다. (ACC_SQR1의 L[3:0] 비트)

실습 2 : A/D 컨버터로 광센서 읽기

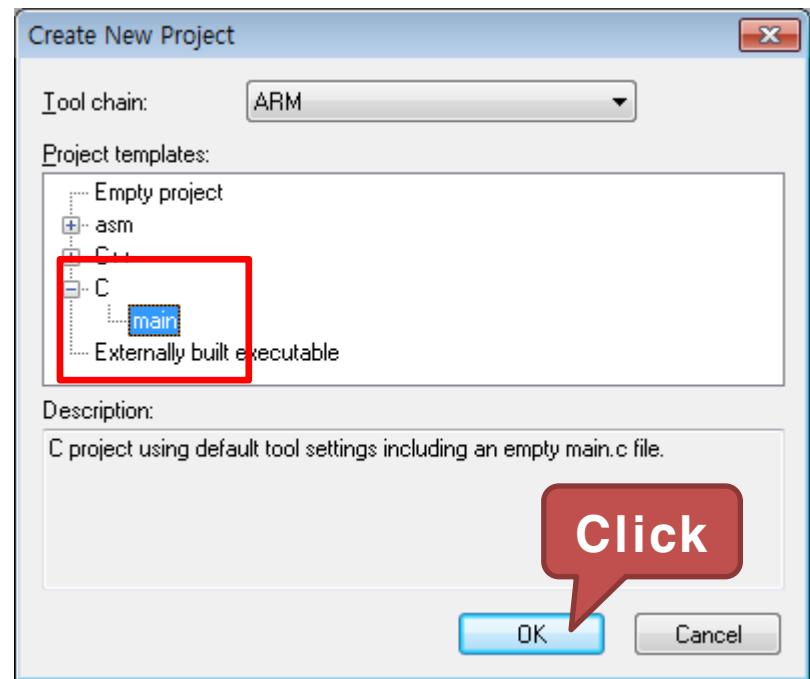
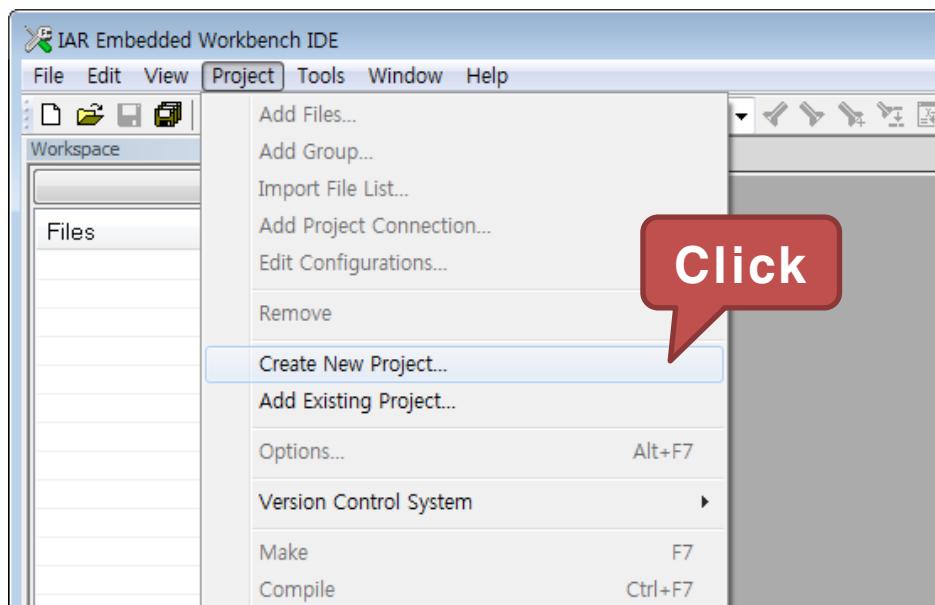
- ADC 라이브러리
 - STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
 - ADC_CommonInitTypeDef 구조체

ADC_CommonInitTypeDef구조체	설 명
uint32_t ADC_Mode	ADC의 동작 모드 설정모드를 설정한다. (ADC_CR1의 DUALMOD 비트)
uint32_t ADC_Prescaler	ADC 클럭의 분주비를 설정한다. (ADC_CCR의 ADCPRE 비트)
uint32_t ADC_DMAAccessMode	DMA 모드를 설정한다. (ADC_CCR의 DMA 비트)
uint32_t ADC_TwoSamplingDelay	ADC의 샘플링간의 대기 시간을 설정한다. (in dual or triple interleaved modes) (ACC_CCR의 DELAY 비트)

실습 2 : A/D 컨버터로 광센서 읽기

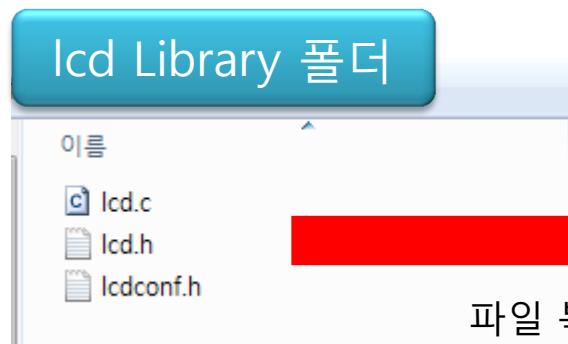
● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch9” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “adcCds”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : A/D 컨버터로 광센서 읽기

- 구동 프로그램 : EWARM에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - lcd.c, lcd.h, lcdconf.h 파일은 “Cortex Example\Projects\library\lcd” 폴더에 존재
 - 라이브러리 파일은 EWARM에서 새 프로젝트를 생성한 후 생성된 폴더에 복사



파일 복사

adcCds 프로젝트 폴더

이름	수정한 날짜	유형
lcd.c	2016-10-05 오전...	C Source File
lcd.h	2016-10-05 오전...	H 파일
lcdconf.h	2016-10-05 오전...	H 파일
stm32f4xx_conf.h	2016-09-28 오전...	H 파일
stm32f4xx_it.c	2016-09-28 오전...	C Source File
stm32f4xx_it.h	2016-09-28 오전...	H 파일
stm32f405xG.icf	2013-06-26 오후...	ICF 파일
system_stm32f4xx.c	2016-09-28 오전...	C Source File

실습 2 : A/D 컨버터로 광센서 읽기

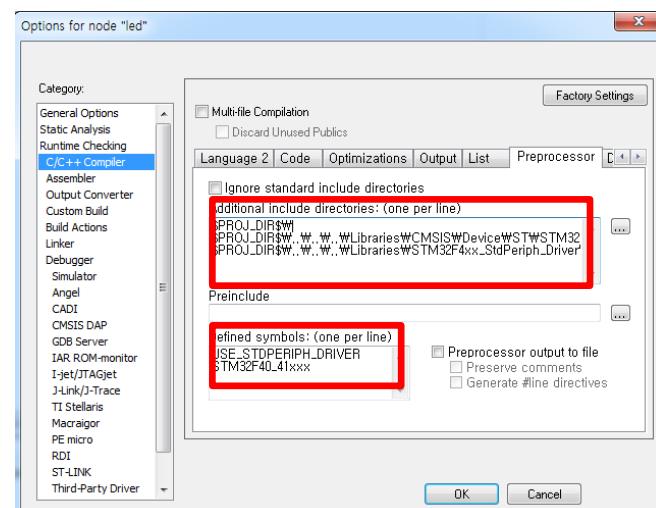
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch9\adcCds	system_stm32f4xx.c
Cortex_Example\Projects\ch9\tadcCds	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Projects\ch9\adcCds	lcd.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_adc.c

실습 2 : A/D 컨버터로 광센서 읽기

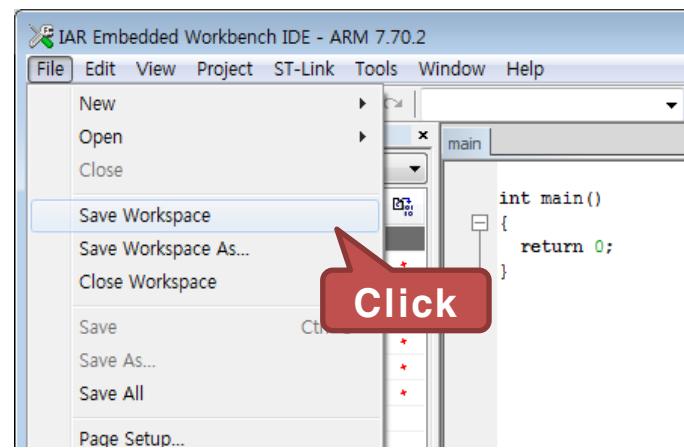
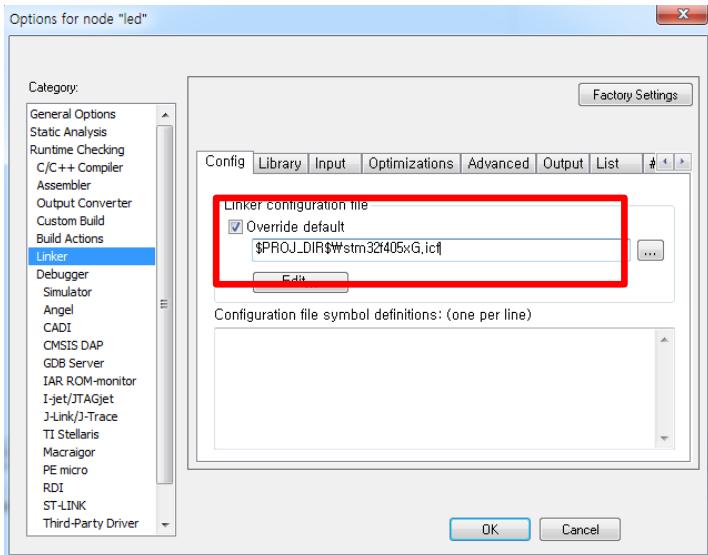
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : A/D 컨버터로 광센서 읽기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : A/D 컨버터로 광센서 읽기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"
#include "lcd.h"          // Text LCD를 사용하기 위한 헤더 파일

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;
    ADC_CommonInitTypeDef ADC_CommonInitStructure;
    uint16_t AdData;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
```

실습 2 : A/D 컨버터로 광센서 읽기

- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
  
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;  
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;  
ADC_CommonInitStructure.ADC_DMAAccessMode =  
    ADC_DMAAccessMode_Disabled;  
ADC_CommonInitStructure.ADC_TwoSamplingDelay =  
    ADC_TwoSamplingDelay_5Cycles;  
ADC_CommonInit(&ADC_CommonInitStructure);
```

실습 2 : A/D 컨버터로 광센서 읽기

- 구동 프로그램

- main.c 코드 작성

```
//...
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge =
                                ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_ExternalTrigConv =
                                ADC_ExternalTrigConv_T1_CC1;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfConversion = 1;
ADC_Init(ADC1, &ADC_InitStructure);

ADC-RegularChannelConfig(
                        ADC1, ADC_Channel_0, 1, ADC_SampleTime_3Cycles);
ADC_Cmd(ADC1, ENABLE);
```

실습 2 : A/D 컨버터로 광센서 읽기

- 구동 프로그램

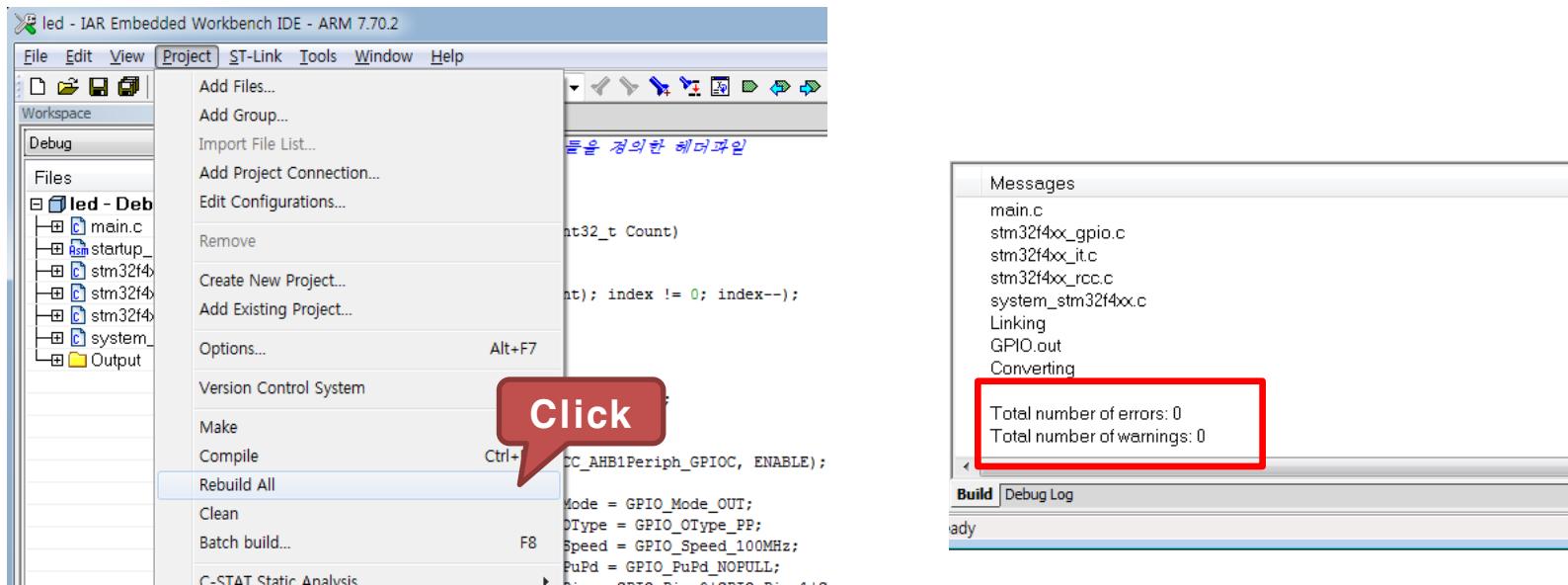
- main.c 코드 작성

```
//...
lcdInit();      // Text LCD를 초기화
while(1) {
    ADC_SoftwareStartConv(ADC1);
    while( !ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));
    ADC_ClearFlag(ADC1,ADC_FLAG_EOC);
    AdData = ADC_GetConversionValue(ADC1);
    lcdGotoXY(0,0);      // 커서위치를 첫번째 줄 첫번째 칸으로 이동
    lcdPrintData(" Cds: ",6);    // " Cds: " 출력
    lcdDataWrite((AdData/1000)%10 + '0');           //1000의 자리 출력
    lcdDataWrite((AdData/100)%10 + '0');           //100의 자리 출력
    lcdDataWrite((AdData/10)%10 + '0');           //10의 자리 출력
    lcdDataWrite((AdData)%10 + '0');           //1의 자리 출력
    Delay(500);
}
}
```

실습 2 : A/D 컨버터로 광센서 읽기

● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 adcCds 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 2 : A/D 컨버터로 광센서 읽기

- 실행 결과

- Text LCD에 광센서값을 출력
- 광센서를 손으로 가리면 센서 값이 달라지는 것을 볼 수 있음



실습 3 : A/D 컨버터로 전압값 읽기

● 실습 개요

- STM32F405 의 A/D 컨버터 기능을 이용하여 가변 저항으로부터 분배되는 전압값을 읽어내어 Text LCD에 출력
- 가변 저항 : 저항값을 변화시킬 수 있는 저항

● 실습 목표

- STM32F405 A/D 컨버터의 동작 원리 이해
- A/D 컨버터 제어 방법 습득(레지스터 설정)
- 가변 저항의 동작 원리 이해

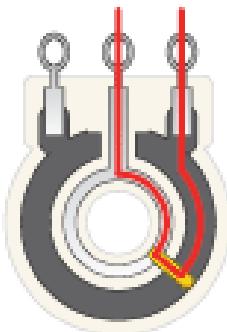
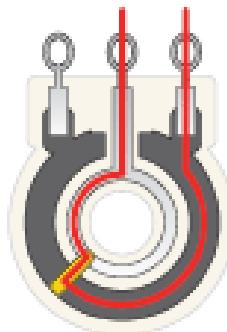
- 사용 모듈

- 가변 저항

- 가변저항은 전자회로에서 저항값을 임의로 바꿀 수 있는 저항기
가변저항을 사용하여 저항을 바꾸면 전류의 크기도 바뀜
 - 장비 운용 중 사용자에 의해 조작되는 형태와 장비 내부에서 생산 또는
교정 중 조정되는 반고정저항 등의 형태가 있음
 - 소자의 특성상 3단자인 경우가 많음. 일반적으로 회전형의 것이
많이 사용되나 오디오 믹서에서처럼 직동방식도 있음

- 가변저항 작동 원리

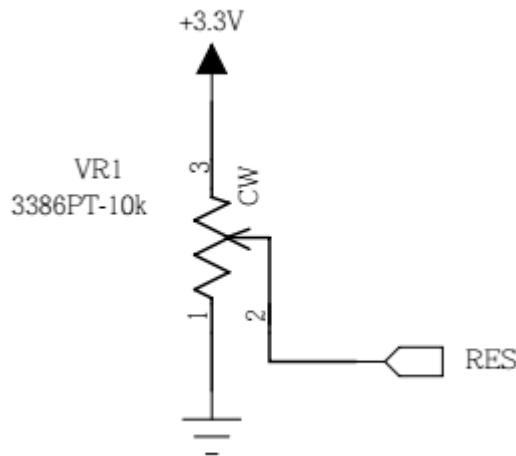
- 보통 가장자리가 카본띠로 이루어져 있어 라이얼을 돌릴 시 아래 그림처럼
+단자와 -단자간 브릿지 역할을 하는 카본의 길이가 변화하며 그에 따른
저항값도 변함



실습 3 : A/D 컨버터로 전압값 읽기

- 사용 모듈

- 센서 모듈의 가변저항 회로
 - 가변 저항의 회전에 따라 저항 값이 변화하여 VRES 핀으로 출력되는 전압이 변함



실습 3 : A/D 컨버터로 전압값 읽기

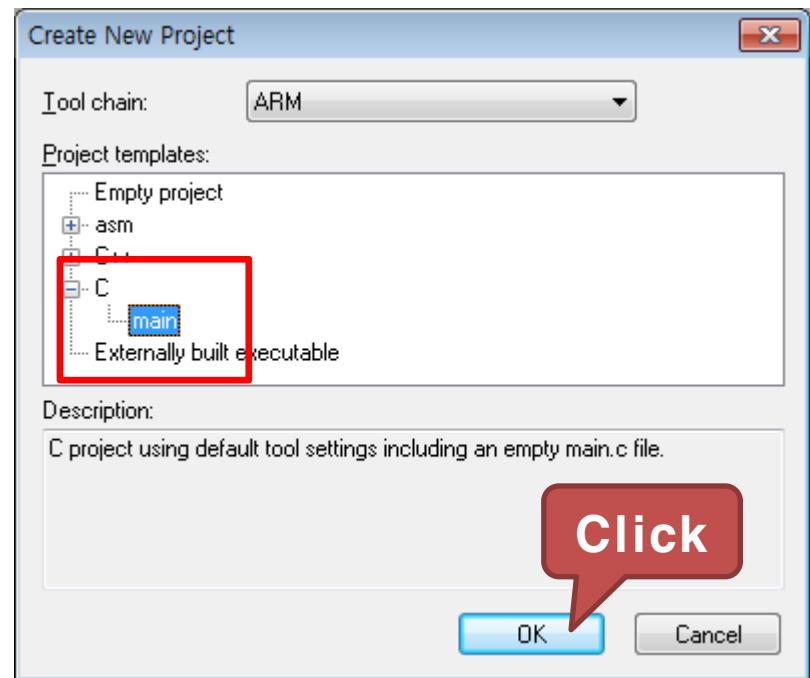
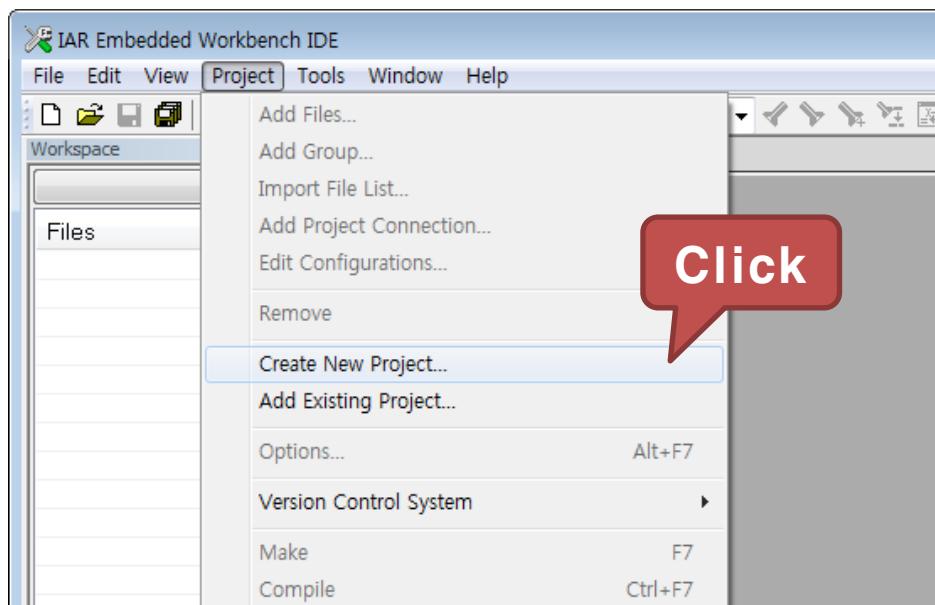
● 구동 프로그램 : 사전 지식

- A/D 컨버터 기능을 이용하여 가변 저항으로 분배되는 전압값의 아날로그 신호를 받아 디지털로 변환한 뒤, 이를 Text LCD에 표시
- Delay 함수를 이용하여 0.5초 간격으로 데이터를 읽도록 함
- A/D 컨버터의 설정 방법
 - A/D 컨버터의 입력채널 결정 : 1번 채널 사용
 - 입력 데이터의 정렬 방법 : 데이터 정렬은 디폴트인 우정렬
 - 컨버팅 모드 : 싱글 컨버팅 모드
 - ADC 클럭 : APB2 클럭의 2분주 ($84\text{MHz} / 2 = 42\text{MHz}$)
 - 컨버팅 사이클 : 3사이클(3사이클 + 12사이클(고정시간) = 15사이클)
 - 샘플링 시간 : $23.8\text{ns}(1/42\text{MHz}) * 15\text{사이클} = 0.357\mu\text{s}$
 - 인터럽트 : 사용하지 않음
- A/D 컨버팅 제어
 - 설정을 마친 후, ADC1을 Enable. 안정적인 A/D를 위해 Calibration을 수행
 - ADC_CR2의 ADON 비트를 1로 설정하여 컨버팅 시작
 - 상태레지스터에서 변환 종료 비트를 확인하여 설정될 때까지 대기
 - ADC_DR 레지스터에서 변환 데이터를 읽어 옴

실습 3 : A/D 컨버터로 전압값 읽기

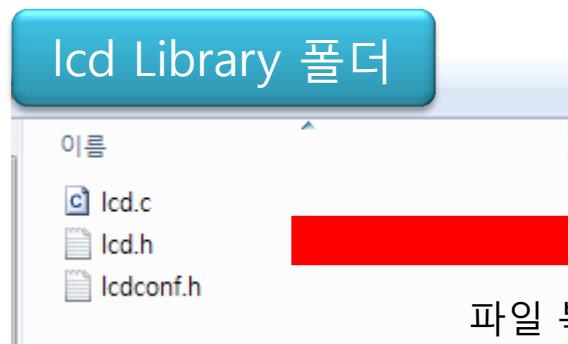
● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch9” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “adcVres”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 3 : A/D 컨버터로 전압값 읽기

- 구동 프로그램 : EWARM에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - lcd.c, lcd.h, lcdconf.h 파일은 “Cortex Example\Projects\library\lcd” 폴더에 존재
 - 라이브러리 파일은 EWARM에서 새 프로젝트를 생성한 후 생성된 폴더에 복사



파일 복사

adcVres 프로젝트 폴더

이름	수정한 날짜	유형
lcd.c	2016-10-05 오전...	C Source File
lcd.h	2016-10-05 오전...	H 파일
lcdconf.h	2016-10-05 오전...	H 파일
stm32f4xx_conf.h	2016-09-28 오전...	H 파일
stm32f4xx_it.c	2016-09-28 오전...	C Source File
stm32f4xx_it.h	2016-09-28 오전...	H 파일
stm32f405xG.icf	2013-06-26 오후...	ICF 파일
system_stm32f4xx.c	2016-09-28 오전...	C Source File

실습 3 : A/D 컨버터로 전압값 읽기

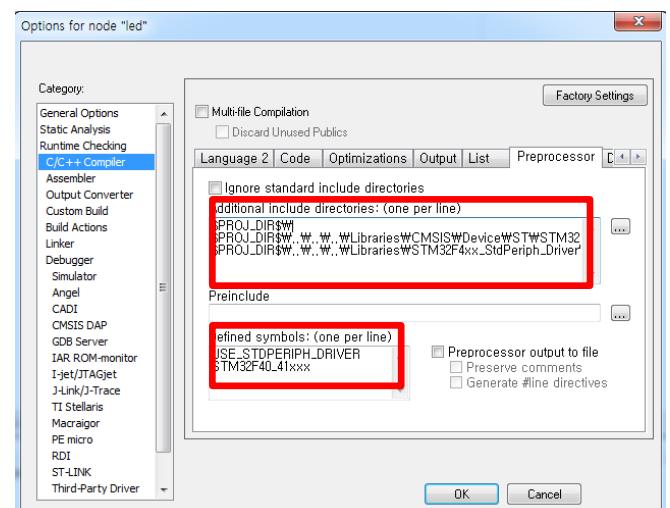
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch9\adcVres	system_stm32f4xx.c
Cortex_Example\Projects\ch9\adcVres	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Projects\ch9\adcVres	lcd.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_adc.c

실습 3 : A/D 컨버터로 전압값 읽기

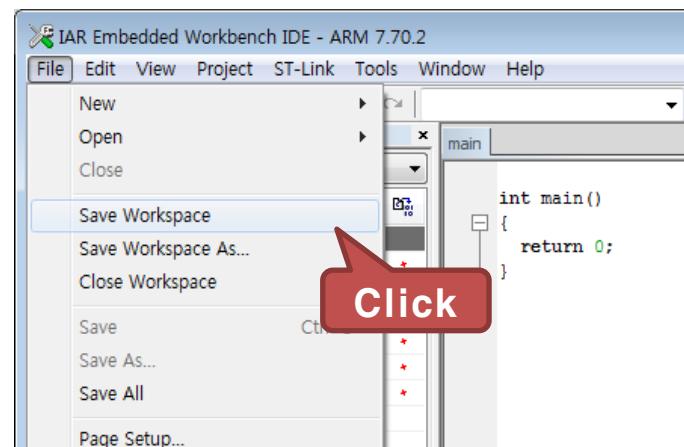
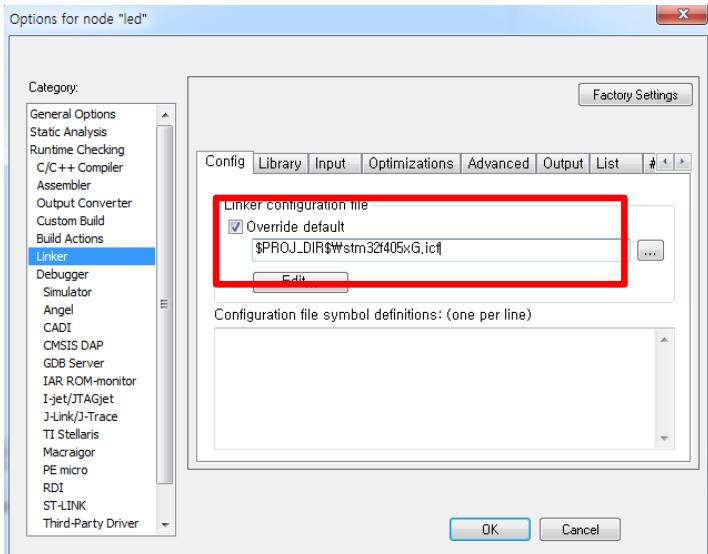
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 3 : A/D 컨버터로 전압값 읽기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 3 : A/D 컨버터로 전압값 읽기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"
#include "lcd.h"          // Text LCD를 사용하기 위한 헤더 파일

static void Delay(const uint32_t Count){ // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}
void printf_2dot1(uint16_t _temp);

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;
    ADC_CommonInitTypeDef ADC_CommonInitStructure;
    uint16_t AdData;
    float v_temp;           // 전압값을 계산할 변수
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
```

실습 3 : A/D 컨버터로 전압값 읽기

- 구동 프로그램

- main.c 코드 작성

```
//...  
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);  
  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
  
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;  
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;  
ADC_CommonInitStructure.ADC_DMAAccessMode =  
    ADC_DMAAccessMode_Disabled;  
ADC_CommonInitStructure.ADC_TwoSamplingDelay =  
    ADC_TwoSamplingDelay_5Cycles;  
ADC_CommonInit(&ADC_CommonInitStructure);
```

실습 3 : A/D 컨버터로 전압값 읽기

- 구동 프로그램

- main.c 코드 작성

```
//...
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge =
                                ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_ExternalTrigConv =
                                ADC_ExternalTrigConv_T1_CC1;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfConversion = 1;
ADC_Init(ADC1, &ADC_InitStructure);
ADC-RegularChannelConfig(
    ADC1, ADC_Channel_1, 1, ADC_SampleTime_3Cycles);
ADC_Cmd(ADC1, ENABLE);
```

실습 3 : A/D 컨버터로 전압값 읽기

- 구동 프로그램
 - main.c 코드 작성

```
//...
lcdInit(); // Text LCD를 초기화

while(1) {
    ADC_SoftwareStartConv(ADC1);
    while( !ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));
    ADC_ClearFlag(ADC1,ADC_FLAG_EOC);
    AdData = ADC_GetConversionValue(ADC1);
    v_temp = (float) AdData * 33 / 4095;
    // [(adc*3.3) /4095 ] => 실제 전압값,
    // 10을 곱해 소수점 첫째짜리까지 표현
    printf_2dot1((uint16_t)v_temp); // 전압값을 text LCD에 출력
    Delay(500);
}
}
```

실습 3 : A/D 컨버터로 전압값 읽기

- 구동 프로그램

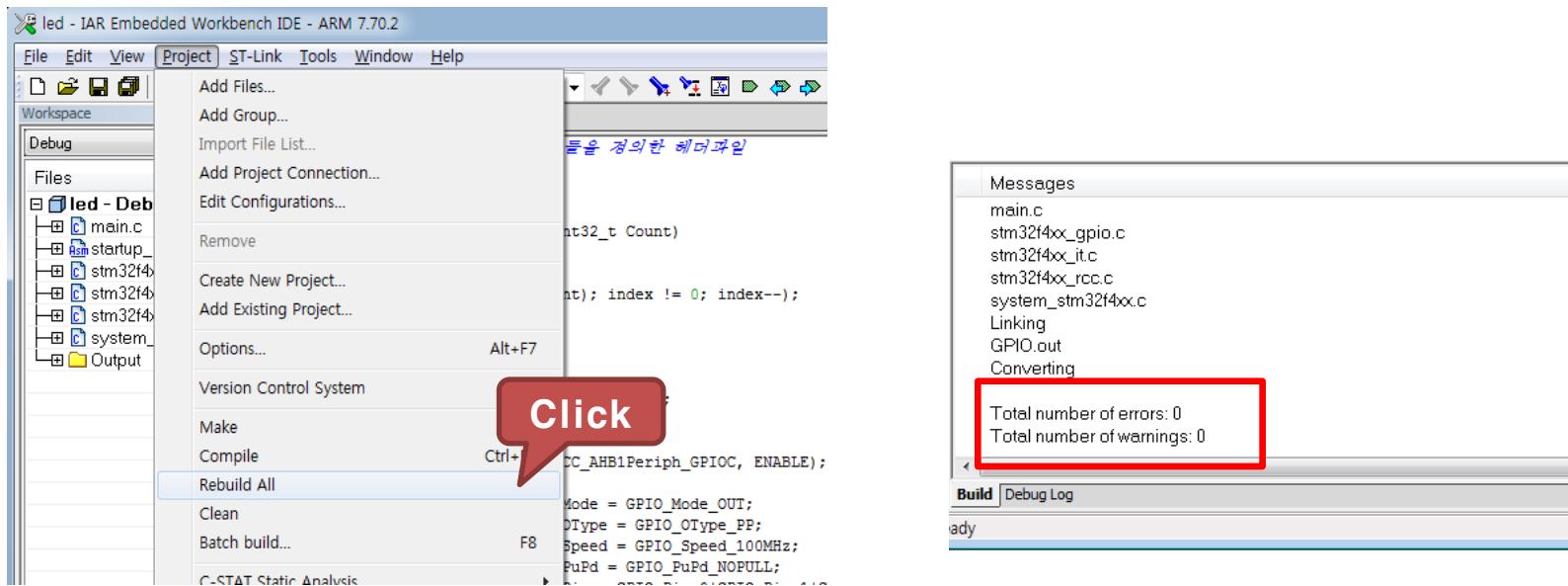
- main.c 코드 작성

```
void printf_2dot1(uint16_t _temp) {  
    uint8_t s100,s10;  
    lcdGotoXY(0,0);          // 커서위치를 첫번째 줄 첫번째 칸으로 이동  
    lcdPrintData(" Vres: ",7); // " Vres: " 출력  
  
    s100 = _temp/100;          // 100의 자리 추출  
  
    if(s100> 0) lcdDataWrite(s100+'0'); // 100의 자리 값이 있으면 출력  
    else lcdPrintData(" ",1);      // 100의 자리 값이 없으면 빈칸 출력  
  
    s10 = _temp%100;            // 100의 자리를 제외한 나머지 추출  
    lcdDataWrite((s10/10)+'0'); // 10의 자리 추출하여 출력  
    lcdPrintData(".",1);       // 소숫점 출력  
    lcdDataWrite((s10%10)+'0'); // 1의 자리 추출하여 출력  
    lcdDataWrite('V');         // 전압 단위 출력  
}
```

실습 3 : A/D 컨버터로 전압값 읽기

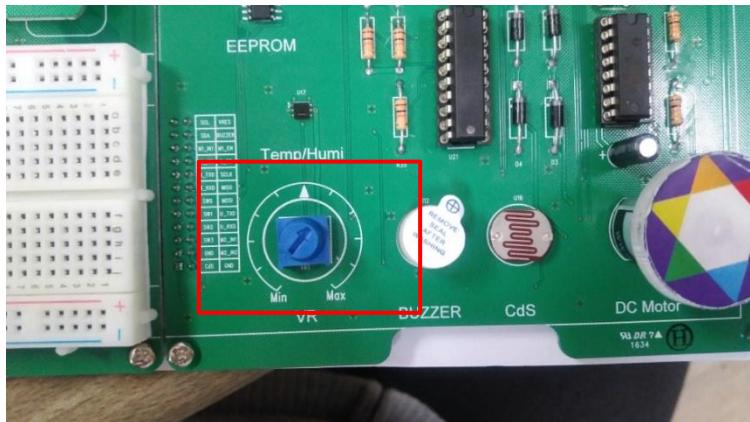
- 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 adcVres 프로젝트 빌드
- “Project” 탭에서 “Rebuild All” 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- “Project” 탭에서 “Download”의 “Download active application”을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 “Reset” 버튼 클릭



실습 3 : A/D 컨버터로 전압값 읽기

- 실행 결과
 - 가변 저항을 회전시키면 출력되는 전압값이 변함



시리얼 인터페이스

- **TWI(Two Wire Serial Interface)**
- **SPI(Serial Peripheral Interface)**
- **TWI(I²C)로 온습도 센서 제어하기**
- **SPI로 EEPROM 붙이기**



엣지아이랩

시리얼 인터페이스

- 패러렐 인터페이스와 시리얼 인터페이스
 - 패러렐 인터페이스(Parallel Interface)
 - 데이터 및 어드레스가 병렬로 동시에 처리
 - 데이터의 처리 속도가 빠르다
 - SRAM등의 외부 메모리 및 고속의 주변 장치들을 연결하는데 주로 사용
 - 다수의 어드레스 신호와 데이터 신호를 사용 칩의 소형화가 어려움
 - AVR, 8051등의 외부 메모리 인터페이스
 - 시리얼 인터페이스(Serial Interface)
 - 고속의 제어가 필요없는 장치들을 위해 소수의 신호를 사용
 - 어드레스와 데이터를 순차적으로 처리
 - 데이터의 처리속도가 비교적 느리다.
 - 필요한 핀수를 최소화하여 칩의 소형화에 유리
 - 고속제어가 필요없는 소형 칩들에 주로 사용
 - TWI(Two Wire Serial Interface), SPI(Serial Peripheral Interface)

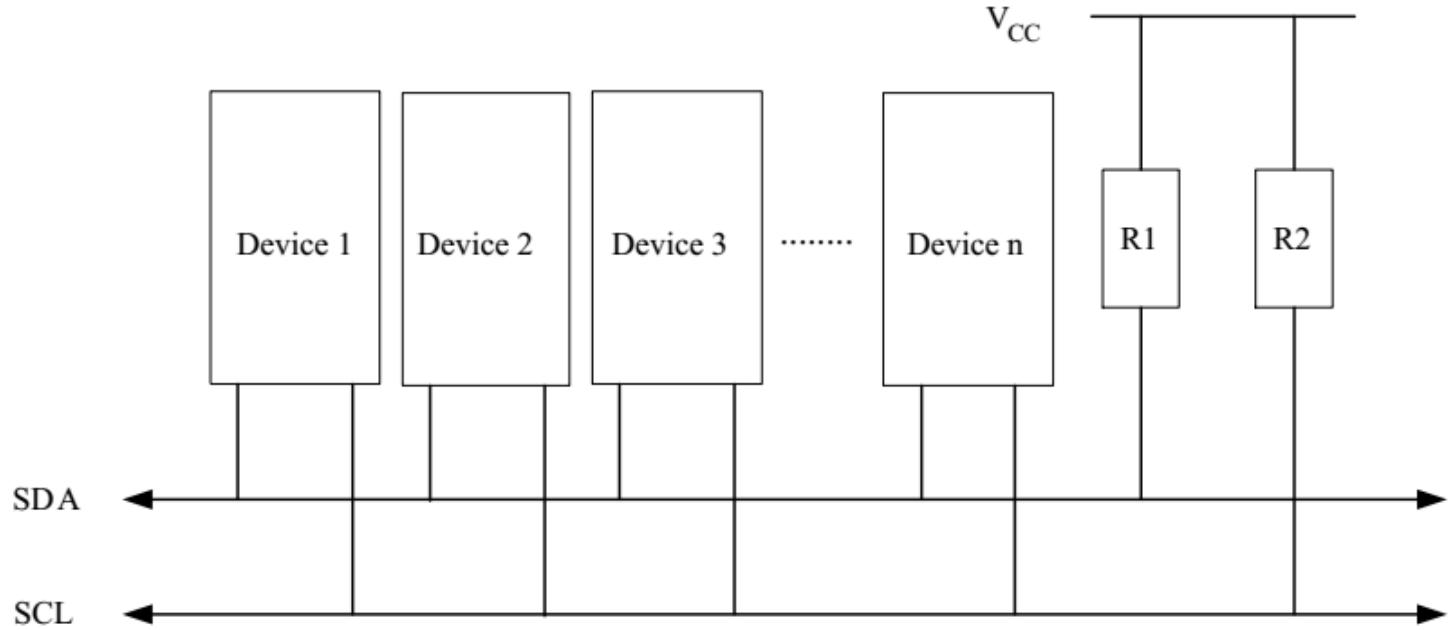
I²C(Inter-integrated circuit) Interface

- I²C(Inter-integrated circuit) Interface

- 단순하면서도 강력한 시리얼 통신 인터페이스
 - 2선을 이용해 시스템 내부에서 여러 장치들과 통신
 - I²C 프로토콜은 클록(SCL)과 데이터(SDA)만으로 양방향 버스 라인 사용
 - 마스터와 슬레이브 동작을 지원하며, 다중 마스터도 가능
 - I²C의 7비트 어드레스는 128개의 다른 슬레이브 어드레스까지 허용
 - 버스에 연결된 모든 디바이스는 독립적인 주소를 가짐
 - 디바이스 어드레스(Device Address) 혹은 디바이스 아이디(Device ID)라 하며, 칩의 구분을 위해 각 칩마다 고유의 디바이스 ID를 가짐
 - I²C는 400kHz까지의 데이터 전송 속도를 가짐

I²C(Inter-integrated circuit) Interface

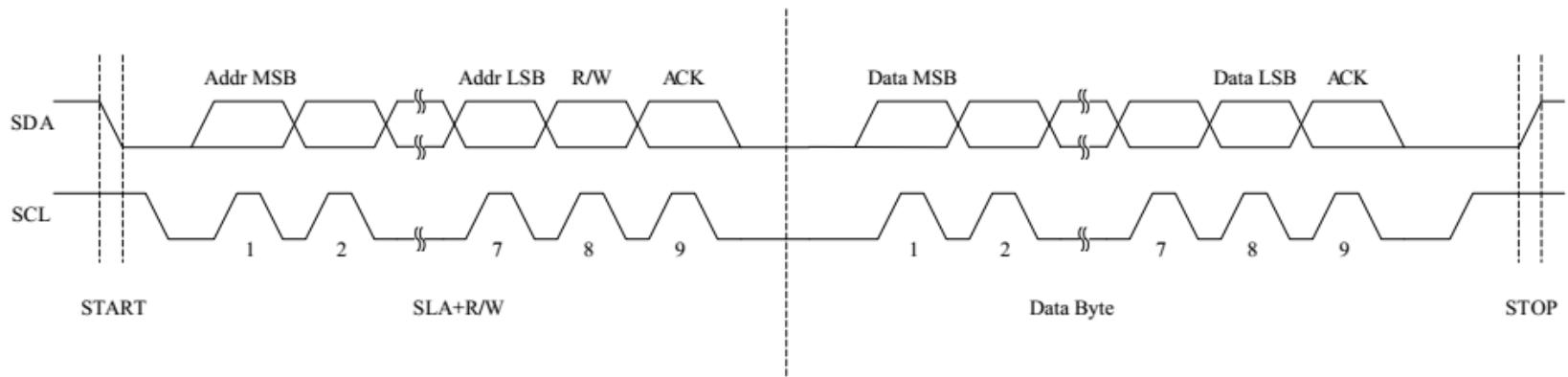
- I²C(Inter-integrated circuit) Interface



- 버스 라인(SCL, SDA)은 풀업 저항을 통하여 +Vcc 전압으로 연결
- 각 디바이스들은 평상시 Tri-State 상태를 유지
- 버스를 사용할 때에는 레벨 하이('1')는 Tri-State로, 레벨 Low('0')는 '0'로 출력

I²C(Inter-integrated circuit) Interface

- I²C 데이터 전송형식
 - 마스터가 버스에 START 조건을 출력할 때 전송은 시작되고 STOP 조건에서 완료
 - START 조건, 어드레스 패킷(SLA + R/W)와 하나 또는 많은 데이터 패킷 그리고 STOP으로 구성



I²C(Inter-integrated circuit) Interface

- STM32F405 I²C 레지스터
 - I2C_CR1(Control register 1) : 제어 레지스터 1
 - I2C_CR2(Control register 2) : 제어 레지스터 2
 - I2C_OAR1(Own address register 1) : 자기 주소 레지스터 1
 - I2C_OAR2(Own address register 2) : 자기 주소 레지스터 2
 - I2C_DR(Data register) : 데이터 레지스터
 - I2C_SR1(Status register 1) : 상태 레지스터 1
 - I2C_SR2(Status register 2) : 상태 레지스터 2
 - I2C_CCR(Clock control register) : 클록 설정 레지스터
 - I2C_TRISE(TRISE register register) : TRISE 레지스터
 - I2C_FLTR(FLTR register) : FLTR 레지스터

STM32F405 I²C 레지스터

- I2C_CR1 (Control register 1)
 - I²C 제어 레지스터 1
 - I²C 모듈의 동작을 설정하는 기능을 수행하는 레지스터
 - SWRST (Software reset)
 - 1로 설정되면, I²C는 리셋 상태로 됨. 단, 그 전에 I²C기능을 사용하지 않고 있어야 함
 - 0 : I²C Peripheral not under reset
 - 1 : I²C Peripheral under reset state
 - ALERT (SMBUS Alert)
 - PEC (Packet Error Checking)
 - 소프트웨어에 의해 Set/Clear 되고, PEC가 전송되거나, START/STOP상태 일 때, 또는 PE = 0 일 때, 이 비트는 하드웨어에 의해 클리어
 - 0 : No PEC transfer
 - 1 : PEC transfer (in Tx or Rx mode)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res	SMBUS	PE

STM32F405 I²C 레지스터

- I2C_CR1 (Control register 1)
 - POS (Acknowledge/PEC Position)
 - 소프트웨어에 의해 Set/Clear 되고, PE = 0 일 때, 이 비트는 하드웨어에 의해 클리어
 - 0 : ACK비트는 시프트레지스터의 수신된 현재 바이트의 (N)ACK을 제어. PEC비트는 시프트레지스터의 현재 바이트가 PEC라는 것을 지시
 - 1 : ACK비트는 시프트레지스터의 수신된 Next바이트(현재바이트의 다음)의 (N)ACK을 제어. PEC비트는 시프트레지스터의 Next 바이트가 PEC라는 것을 지시
 - ACK (Acknowledge Enable)
 - 소프트웨어에 의해 Set/Clear 되고, PE = 0 일 때, 이 비트는 하드웨어에 의해 클리어
 - 0 : No acknowledge returned
 - 1 : Acknowledge returned after a byte is received (matched address or data)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res	SMBUS	PE

STM32F405 I²C 레지스터

- I2C_CR1 (Control register 1)
 - STOP (Stop Generation)
 - 소프트웨어에 의해 Set/Clear 되고, Stop상태가 검출되었을 때
이 비트는 하드웨어에 의해 클리어 되며, 타임아웃 에러를 검출했을 때는
하드웨어에서 1로 설정
 - In Master Mode:
 - 0 : No Stop generation.
 - 1 : Stop generation after the current byte transfer or after the
current Start condition is sent.
 - In Slave mode:
 - 0 : No Stop generation.
 - 1 : Release the SCL and SDA lines after the current byte transfer.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res	SMBUS	PE

STM32F405 I²C 레지스터

- I2C_CR1 (Control register 1)
 - START(Start Generation)
 - 소프트웨어에 의해 Set/Clear 되고, PE = 0 일때, 이 비트는 하드웨어에 의해 클리어
 - In Master Mode:
 - 0 : No Start generation
 - 1 : Repeated start generation
 - In Slave mode:
 - 0 : No Start generation
 - 1 : Start generation when the bus is free

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res	SMBUS	PE

STM32F405 I²C 레지스터

- I2C_CR1 (Control register 1)

- NOSTRETCH (Clock Stretching Disable (slave mode))
 - 소프트웨어 Reset동안, ADDR이나 BTF 플래그가 설정되어 있을 때, 슬레이브 모드의 clock stretching을 disable 함
 - 0 : Clock stretching enabled
 - 1 : Clock stretching disabled
- ENGC (General Call Enable)
 - 0 : General call disabled. Address 00h is NACKed.
 - 1 : General call enabled. Address 00h is ACKed.
- ENPEC (PEC Enable)
 - 0 : PEC calculation disabled
 - 1 : PEC calculation enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res	SMBUS	PE

STM32F405 I²C 레지스터

- I2C_CR1 (Control register 1)

- ENARP (ARP Enable)
- SMBTYPE (SMBus type)
- SMBUS (SMBus Mode)
- PE (Peripheral Enable)

■ I2C통신 ON/OFF 비트로서, 이 비트가 통신 중에 리셋되면 통신 종료 후, IDLE상태가 됨. 그리고 PE=0이 되면 모든 비트는 통신 종료 후 리셋
마스터모드에서는 반드시 통신 종료할 때까지 이 비트를 리셋 하면 안됨

- 0 : Peripheral disable
- 1 : Peripheral enable: the corresponding I/Os are selected as alternate functions depending on SMBusbit.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res	SMBUS	PE	

STM32F405 I²C 레지스터

- I2C_CR2 (Control register 2)
 - I²C 제어 레지스터 2
 - I²C 모듈의 동작을 설정하는 기능을 수행하는 레지스터
 - LAST (DMA Last Transfer)
 - DMAEN (DMA Requests Enable)
 - ITBUFEN (Buffer Interrupt Enable)
 - 0 : TxE = 1 or RxNE = 1 does not generate any interrupt.
 - 1 : TxE = 1 or RxNE = 1 generates Event Interrupt (whatever the state of DMAEN)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved		FREQ[5:0]						

STM32F405 I²C 레지스터

- I2C_CR2 (Control register 2)
 - ITEVTEN (Event Interrupt Enable)
 - 0 : Event interrupt disabled
 - 1 : Event interrupt enabled
 - 인터럽트가 발생되기 위해서는,
 - SB = 1(Master)
 - ADDR = 1 (Master/Slave)
 - ADD1 = 1(Master)
 - STOPF = 1 (Slave)
 - BTF = 1 with no TXE or RxEN event
 - TxE event to 1 if ITBUFEN = 1
 - RxEN event to 1 if ITBUFEN = 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved		FREQ[5:0]						

STM32F405 I²C 레지스터

- I2C_CR2 (Control register 2)
 - ITERREN (Error Interrupt Enable)
 - 0 : Event interrupt disabled
 - 1 : Event interrupt enabled
 - 인터럽트가 발생되기 위해서는,
 - BERR = 1
 - ARLO = 1
 - AF = 1
 - OVR = 1
 - PECERR = 1
 - TIMEOUT = 1
 - SMBAlert = 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved	FREQ[5:0]							

STM32F405 I²C 레지스터

- I2C_CR2 (Control register 2)

- FREQ[5:0] (Peripheral Clock Frequency)

- 입력 클럭 주파수는 반드시 정확하게 프로그래밍 되어야 함

클럭의 범위는 2MHz~ 36MHz

- 000000 : Not allowed
- 000001 : Not allowed
- 000010 : 2MHz
-
- 100100 : 36MHz
- Others : Not allowed

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		LAST	DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved		FREQ[5:0]						

STM32F405 I²C 레지스터

- I2C_OAR1 (Own address register 1)
 - 자기 주소 레지스터 1
 - 자신의 디바이스 어드레스를 설정하는 레지스터
 - ADDMODE (Addressing Mode (Slave Mode))
 - 0 : 7비트 슬레이브 어드레스 (10비트 어드레스는 인정하지 않음)
 - 1 : 10비트 슬레이브 어드레스 (7비트 어드레스는 인정하지 않음)
 - 비트 14 : 항상 소프트웨어에 의해 1로 유지되어야 함
 - ADD[9:8] (Interface Address)
 - 10비트 어드레스 모드인 경우만 사용
 - ADD[7:1] (Interface Address)
 - 어드레스 7:1 영역
 - ADD0 (Interface Address)
 - 10비트 어드레스 모드인 경우, 0으로 되어야 함

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD MODE	Reserved					ADD[9:8]		ADD[7:1]						ADD0	

STM32F405 I²C 레지스터

- I2C_OAR2 (Own address register 2)

- 자기 주소 레지스터 2
- 자신의 디바이스 어드레스를 설정하는 레지스터
- ADD2[7:1] (Interface Address)
- 듀얼 어드레스 모드의 어드레스 주소 7:1 영역
- ENDUAL (Dual addressing mode enable)
 - 0 : 7비트 어드레스 모드에서 OAR1 레지스터만 인정한다.
 - 1 : 7비트 어드레스 모드에서 OAR1, OAR2 둘다 인정한다.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						ADD2[7:1]						ENDUAL			

STM32F405 I²C 레지스터

- I2C_DR (Data register)
 - 데이터 레지스터
 - 송수신 데이터를 저장하는 레지스터
 - DR[7:0] (8비트 데이터 레지스터)
 - 전송 모드
 - DR레지스터에 1바이트를 쓰게 되면, 데이터 전송이 자동으로 시작
전송이 시작 되었을 때(TxE = 1), DR레지스터에 이어지는 데이터를
입력하게 되면, 연속 데이터 전송이 가능
 - 수신 모드
 - RxNE = 1일 때, 수신 받은 데이터가 DR레지스터로 복사
수신된 데이터는 다음 수신될 데이터가 현재 데이터의 위치에
저장되기 전에 (overrun) 읽어져야 함

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DR[7:0]							

STM32F405 I²C 레지스터

- I2C_SR1 (Status register 1)
 - I²C 상태 레지스터 1
 - 통신 상태를 나타내는 레지스터 1
 - SMBALERT (SMBus Alert)
 - TIMEOUT (Timeout or Tlow Error)
 - 0 : No timeout error
 - 1 : SCL이 25ms동안 LOW를 유지하고 있거나(타임아웃), 마스터 누적 클럭이 10ms보다 낮아질 때(Tlow: mext), 슬레이브 누적 클럭이 25ms보다 낮아질 때(Tlow = sext) 설정
 - 슬레이브 모드로 설정되면서 슬레이브는 통신을 리셋하고, 라인들은 하드웨어에 의해 해제될 때
 - 마스터 모드로 설정되면서 하드웨어에 의해 Stop 상태가 전송될 때, 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res	STOPF	ADD10	BTF	ADDR	SB

STM32F405 I²C 레지스터

- I2C_SR1 (Status register 1)
 - PECERR (PEC Error in reception)
 - 0 : no PEC error. Receiver는 PEC 수신후에(ACK = 1), ACK를 돌려보냄
 - 1 : PEC error. Receiver는 PEC 수신후에(ACK = 1), NACK를 돌려보냄
 - 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어
 - OVR (Overrun / Underrun)
 - 0 : No overrun / underrun
 - 1 : Overrun or underrun
 - 슬레이브 모드에서 하드웨어에 의해 설정되는 경우는 아래와 같음 (NOSTRETCH=1)
 - 수신모드에서 새로운 데이터를 수신받고 (ACK 펄스와 같이), DR레지스터를 아직 읽지 않았다면, 새로운 데이터는 손실
 - 전송모드에서 새로운 데이터가 전송될 예정이고, DR레지스터에 아직 쓰기를 않았다면, 같은 데이터는 2번 전송
 - 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res	STOPF	ADD10	BTF	ADDR	SB

STM32F405 I²C 레지스터

- I2C_SR1 (Status register 1)
 - AF (Acknowledge Failure)
 - 0 : No acknowledge failure.
 - 1 : Acknowledge failure
 - ACK 신호가 오지 않으면 하드웨어에 의해 설정
 - 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어
 - ARLO (Arbitration Lost (master mode))
 - BERR (Bus Error)
 - 0 : Start 또는 Stop 상태를 놓치지 않음.
 - 1 : Start 또는 Stop 상태를 놓침.
 - Start 또는 Stop 상태를 놓치게 되면, 하드웨어에 의해 설정
 - 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res	STOPF	ADD10	BTF	ADDR	SB

STM32F405 I²C 레지스터

- I2C_SR1 (Status register 1)
 - TxE (Data Register Empty)
 - 0 : 데이터 레지스터가 비어있지 않음
 - 1 : 데이터 레지스터가 비어있음(데이터 전송가능)
 - 전송모드에서 DR 레지스터가 비워져 있다면, 1로 설정
어드레스 비교 중 일 때는, 설정되지 않음
 - 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어
 - RxNE (Data Register not Empty)
 - 0 : 데이터 레지스터가 비어있음
 - 1 : 데이터 레지스터가 비어있지 않음 (데이터가 수신되어 있음)
 - 수신모드에서 DRO이 비워져 있지 않다면 1로 설정, RxEN비트는 어드레스 비교중 일 때는 설정되지 않음
 - 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res	STOPF	ADD10	BTF	ADDR	SB

STM32F405 I²C 레지스터

- I2C_SR1 (Status register 1)
 - STOPF (Stop detection)
 - 0 : 스톱 상태가 감지되지 않음
 - 1 : 스톱 상태 감지
 - 슬레이브로부터 ACK 신호 수신후에, 스톱 상태(Stop Condition)이 검출되면 하드웨어에 의해 설정
 - 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어
 - ADD10 (10-bit header sent)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res	STOPF	ADD10	BTF	ADDR	SB

STM32F405 I²C 레지스터

- I2C_SR1 (Status register 1)
 - BTF (Byte Transfer Finished)
 - 0 : 데이터 전송이 완료되지 않음
 - 1 : 데이터 전송 완료
 - 하드웨어에 의해 설정되는 경우는 아래와 같음 (NOSTRETCH = 1)
 - 수신 중에, 새로운 데이터를 수신 받고(ACK풀스 포함), DR레지스터의 값을 아직 읽지 않았을 때(RxNE = 1)
 - 전송 중에, 새로운 데이터가 전송될 예정이고, DR레지스터에 아직 쓰지 않았을 때(Write)
 - DR레지스터를 Read/Write하거나, PE=0이 되거나, 전송 중에서 Start 또는 Stop 상태 후에 하드웨어에 의해 SR1 레지스터를 읽어 들이면, 소프트웨어에 의해 0으로 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res	STOPF	ADD10	BTF	ADDR	SB

STM32F405 I²C 레지스터

- I2C_SR1 (Status register 1)

- ADDR (Address sent (master mode) / matched (slave))
 - SR1, SR2 레지스터를 읽으면 소프트웨어에 의해, PE=1로 설정하면 하드웨어에 의해 클리어
 - Address Matched (Slave)
 - 0 : 어드레스가 맞지 않거나, 수신되지 않았음
 - 1 : 수신된 어드레스가 일치함(OAR 레지스터의 주소와 비교해서)
 - Address Sent (Master)
 - 0 : 어드레스 전송이 끝나지 않음
 - 1 : 어드레스 전송 완료
 - For 10-bit addressing, the bit is set after the ACK of the 2nd byte.
 - For 7-bit addressing, the bit is set after the ACK of the byte.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res	STOPF	ADD10	BTF	ADDR	SB

STM32F405 I²C 레지스터

- I2C_SR1 (Status register 1)
 - SB (Start Bit (Master mode))
 - 0 : 스타트 상태가 아님
 - 1 : 스타트 상태가 발생함
 - 소프트웨어에 의해 0으로 클리어 되거나, PE=0일 때 하드웨어에 의해 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res	STOPF	ADD10	BTF	ADDR	SB

STM32F405 I²C 레지스터

- I2C_SR2 (Status register 2)
 - I²C 상태 레지스터 2
 - 통신 상태를 나타내는 레지스터
 - PEC[7:0] (Packet Error Checking Register)
 - 이 비트들은 ENPEC = 1일 때, 내부 PEC를 포함
 - DUALF (Dual Flag(Slave Mode))
 - 0 : 수신된 어드레스를 OAR1과 비교
 - 1 : 수신된 어드레스를 OAR2과 비교
 - Stop 상태후 또는, Start condition이 재전송되거나, PE=0 일 때, 하드웨어에 의해 클리어
 - SMBHOST (SMBus Host Header)
 - SMBDEFAULT (SMBus Device Default Address)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEC[7:0]								DUALF	SMB HOST	SMB DEFAULT	GEN CALL	Res	TRA	BUSY	MSL

STM32F405 I²C 레지스터

- I2C_SR2 (Status register 2)
 - GENCALL (General Call Address)
 - 0 : No General Call
 - 1 : General Call Address received when ENGC=1
 - Stop 상태 후 또는, Start condition이 재전송되거나, PE=0 일때, 하드웨어에 의해 클리어
 - TRA (Transmitter / Reciever)
 - 0 : 데이터를 수신
 - 1 : 데이터를 전송

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEC[7:0]								DUALF	SMB HOST	SMB DEFAULT	GEN CALL	Res	TRA	BUSY	MSL

STM32F405 I²C 레지스터

- I2C_SR2 (Status register 2)
 - BUSY (Bus Busy)
 - 통신 상태를 나타내며, PE = 0이 될 때까지 상태정보를 업데이트
 - 0 : No communication on the bus
 - 1 : Communication ongoing on the bus
 - SDA나 SCL의 low를 검출하면, 하드웨어가 이 비트를 설정
 - Stop condition을 검출하면, 하드웨어에 의해 클리어
 - MSL (Master / Slave)
 - 0 : Slave Mode
 - 1 : Master Mode
 - SB = 1이 되어 마스터 모드가 되면, 하드웨어에 의해 설정
 - Stop condition 다음이나, loss of arbitration(ARLO = 1) 또는, PE = 0일 때, 하드웨어에 의해 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEC[7:0]								DUALF	SMB HOST	SMB DEFAULT	GEN CALL	Res	TRA	BUSY	MSL

STM32F405 I²C 레지스터

- I2C_CCR (Clock control register)
 - I²C 클럭 제어 레지스터
 - I²C의 클럭을 설정하는 레지스터
 - F/S (I²C Master Mode Selection)
 - 0 : Standard Mode I²C
 - 1 : Fast Mode I²C
 - DUTY (Fast Mode Duty Cycle)
 - 0 : Fast Mode t_{low} / t_{high} = 2
 - 1 : Fast Mode t_{low} / t_{high} = 16/9

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved	CCR[11:0]												

STM32F405 I²C 레지스터

- I2C_TRISE (TRISE register)
 - I²C TRISE 레지스터
 - TRISE[5:0] (Maximum Rise Time in Fast/Standard mode (Master mode))
 - 최대 SCL클럭의 펄스유지기간을 결정(PE=0일때만 설정가능)
예를 들어, Standard모드에서는 최대 SCL rise time은 1000ns(1MHz)인데,
만약, I2C_CR2레지스터의 FREQ[5:0]=8 (8MHz)이고, Tck = 125ns 으로
설정되어 있다면, TRISE[5:0]은 9(8+1)로 설정(1000ns / 125ns = 8+1).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TRISE[5:0]							

STM32F405 I²C 레지스터

- I2C_CCR (Clock control register)
 - Clock Control Register in Fast/Standard mode (Master mode)
 - 마스터모드의 SCL 클럭을 제어
 - Standard Mode or SMBus
 - $T_{high} = CCR * TPCLK1$
 - $T_{low} = CCR * TPCLK1$
 - Fast Mode
 - DUTY = 0이면 $\rightarrow T_{high} = CCR * T_{ck}$, $T_{low} = 2 * CCR * TPCLK1$
 - DUTY = 1이면(to reach 400KHz) $\rightarrow T_{high} = 9 * CCR * T_{ck}$,
 - $T_{low} = 16 * CCR * T_{ck}$
 - TPCLK1 는 FREQR비트로 선택된 Peripheral 클럭주파수의 주기
 - 예) FREQR = 8이면, $T_{ck} = 125\text{ns}$. Standard모드에서 100KHz를 만들려면, CCR은 40 (0x28)이 되야 함.
 $40 \times 125\text{ns} = 5000 \text{ ns} \Rightarrow$ 주파수는 200KHz $T_{high} + T_{low} = 10\text{ms} \Rightarrow$ 100KHz가 계산

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved													CCR[11:0]

I²C(Inter-integrated circuit) Interface

- I²C 동작
 - I²C는 바이트 단위로 동작이 이루어지고, 인터럽트를 기본으로 함
 - 인터럽트는 바이트의 수신이나 START 조건의 전송처럼 모든 버스의 이벤트 뒤에 발생
 - I2C_CR2의 ITEVFEN (Event Interrupt Enable) 비트와 ITBUFEN(Buffer Interrupt Enable)비트가 1로 세트 되면 인터럽트가 발생
 - 만약 인터럽트를 사용하지 않으면, 응용 소프트웨어에서 I2C 버스에서의 동작상태를 알기 위해 I2C SR1,2 레지스터를 정기적으로 조사(poll)해야 함

I2C(Inter-integrated circuit) Interface

- TWI를 동작시키는 방법
 1. I2C_CR1의 START 비트를 세팅하여 START Condition을 내보냄
 2. START Condition이 전송되면, I2C_SR1의 SB비트가 세팅 되고, ITEVFEN비트가 세팅 되어 있다면 인터럽트가 발생
 3. 전송모드일때, 7비트 어드레스와 SLA+W 를 I2C_DR 레지스터에 넣음
 4. 주소와 SLA+W가 전송되고 ACK비트가 정상적으로 도착하면, I2C_SR의 ADDR 비트가 세팅 되며, ITEVFEN비트가 세팅 되어 있다면 인터럽트가 발생
 5. I2C_SR1레지스터의 TxE비트가 하드웨어에 의해 클리어 되고, ITEVFEN비트가 세팅되어 있다면 인터럽트가 발생된다. 이때, I2C_DR레지스터에 데이터를 넣으면, 데이터가 전송
 6. I2C_CR1레지스터의 STOP비트를 세팅하면, Stop condition이 전송되고, 자동적으로 I2C_SR2의 MSL비트가 클리어 되면서, 슬레이브 모드가 됨
 7. 수신 모드일 경우, 1,2번 과정을 진행
 8. 어드레스 전송 시, SLA+R을 I2C_DR레지스터에 넣음
 9. 주소와 SLA+R가 전송되고 ACK비트가 정상적으로 도착하면, I2C_SR의 ADDR 비트가 세팅 되며, ITEVFEN비트가 세팅 되어 있다면 인터럽트가 발생
 10. I2C_SR1레지스터의 RxNE비트가 하드웨어에 의해 클리어 되고, ITEVFEN비트가 세팅 되어 있다면 인터럽트가 발생된다. 이때, 전송된 데이터가 저장되어 있는 I2C_DR레지스터에 읽어 들이면, RxNE비트는 클리어 됨
 11. 통신을 종료하려면 6번 과정을 진행

SPI(Serial Peripheral Interface)

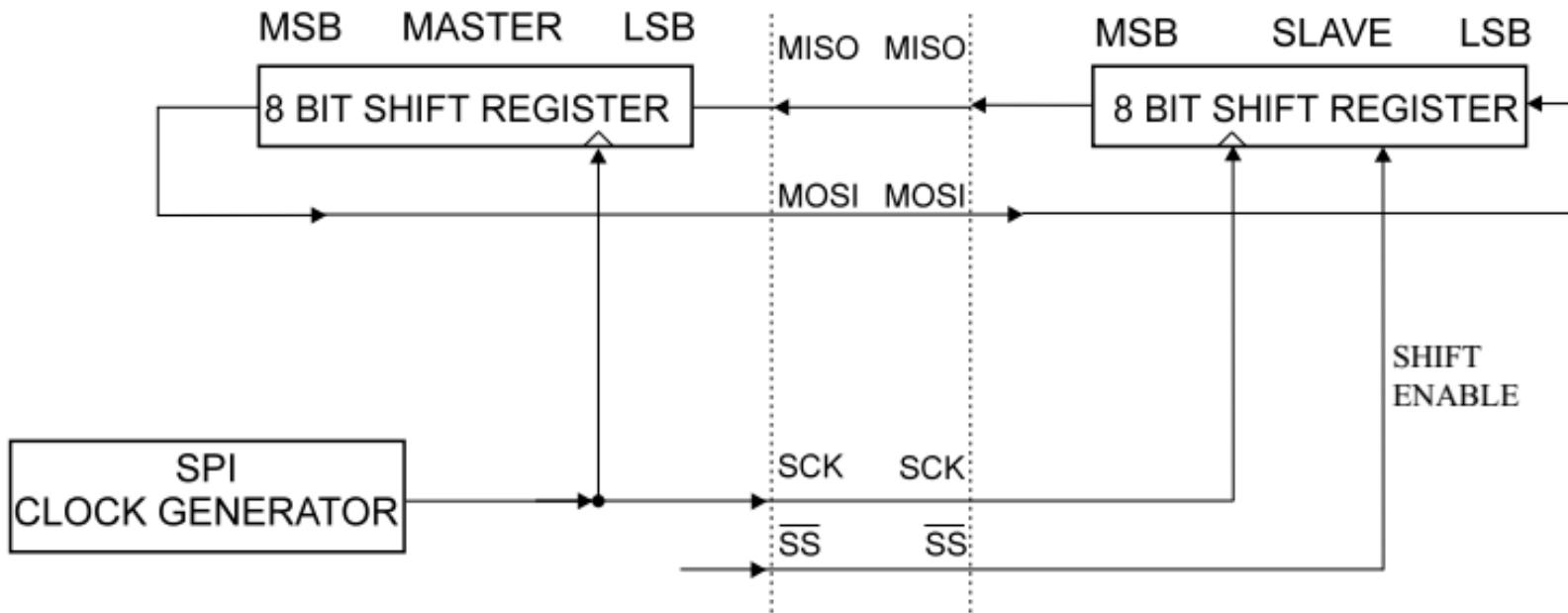
- SPI(Serial Peripheral Interface)
 - 직렬 주변 장치 인터페이스(모토로라사에서 제창한 방식)
 - STM32F405와 주변 장치 디바이스간에, 또는 여러 디바이스들간에 고속 동기 데이터를 전송하기 위한 인터페이스
 - STM32F405는 3개의 SPI를 지원
 - 전이중(Full-Duplex) 통신 방식으로, 3개의 신호선을 이용하는 동기 데이터 전송 방식을 따름
 - 마스터와 슬레이브 방식으로 동작
 - 8가지의 보우레이트 세팅
 - 하드웨어에 의한 CRC 기능을 지원

SPI(Serial Peripheral Interface)

- SPI(Serial Peripheral Interface) 동작
 - SPI는 반드시 1개의 마스터와 1개의 슬레이브 사이에서만 동작
 - 마스터가 슬레이브에게 데이터를 보낼 때
 - 여러 슬레이브에서 원하는 슬레이브에게 SS(Slave Select) 신호를 0 레벨로 출력하여 선택
 - 클록 신호를 발생하여 SCK(Serial Clock)을 통해 출력
 - 송신할 데이터를 시프트 레지스터에 데이터를 준비하여 MOSI(Master Output Slave Input) 단자로 출력
 - 동시에 MISO(Master Input Slave Output) 단자를 통해서는 더미 데이터가 입력

SPI(Serial Peripheral Interface)

- SPI(Serial Peripheral Interface) 동작
 - SPI Master와 Slave간 상호 연결



SPI(Serial Peripheral Interface)

- SPI(Serial Peripheral Interface) 레지스터
 - SPI_CR1(Control register 1)
 - 제어 레지스터 1
 - SPI_CR2(Control register 2)
 - 제어 레지스터 2
 - I2C_SR(Status register)
 - 상태 레지스터
 - I2C_DR(Data register)
 - 데이터 레지스터

STM32F405 SPI 레지스터

- SPI_CR1 (Control register 1)
 - SPI 제어 레지스터 1
 - SPI 모듈의 동작을 설정하는 기능을 수행하는 레지스터
 - BIDMODE (Bidirectional data mode enable)
 - 0 : 2-line unidirectional data mode selected
 - 1 : 1-line bidirectional data mode selected
 - BIDIOE (Output enable in bidirectional mode)
 - 0 : Output disabled (receive-only mode)
 - 1 : Output enabled (transmit-only mode)
 - CRCEN (Hardware CRC calculation enable)
 - 0 : CRC calculation disabled
 - 1 : CRC calculation enabled
 - 이 비트는 SPI가 disabled 되어 있을 때만, 설정 가능

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDIOE	CRCEN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]	MSTR	CPOL	CPHA		

STM32F405 SPI 레지스터

- SPI_CR1 (Control register 1) : SPI 제어 레지스터 1
 - CRCNETX (Transmit CRC next)
 - 0 : Next transmit value is from Tx buffer
 - 1 : Next transmit value is from Tx CRC register
 - DFF (Data Frame Format)
 - 0 : 송/수신을 위한 8비트 프레임포맷을 선택
 - 1 : 송/수신을 위한 16비트 프레임포맷을 선택
 - 이 비트는 SPI가 disabled 되어 있을 때만, 설정 가능

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
BIDI MODE	BIDIOE	CRCEN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]			MSTR	CPOL	CPHA	

STM32F405 SPI 레지스터

- SPI_CR1 (Control register 1)

- RXONLY

- BIDIMODE와 같이 2 line unidirectional mode의 전송 방향을 선택한다.
 - 0 : Full duplex (Tx and Rx)
 - 1 : Output disabled (Receive only mode)

입력			모드
BIDIMODE	BIDIOE	RXONLY	
0	0	0	2 Line FullDuplex Mode
0	0	1	2 Line Rxonly Mode
1	0	0	1 Line Rx Mode
1	1	0	1 Line Tx Mode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDIOE	CRCEN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]			MSTR	CPOL	CPHA

STM32F405 SPI 레지스터

- SPI_CR1 (Control register 1)
 - SSM (Software slave management)
 - SSM비트를 설정하면, NSS핀 입력은 SSI비트의 값에 의해 결정
 - 0 : Software slave management disabled
 - 1 : Software slave management enabled
 - SSI(Internal slave select)
 - 이 SSM비트가 설정되었을 때 동작, 이 비트의 값은 NSS핀에 영향을 주고, NSS핀의 I/O값은 무시
 - LSBFIRST (Frame Format)
 - 0 : MSB부터 전송
 - 1 : LSB부터 전송
 - SPE (SPI Enable)
 - 0 : Peripheral disabled
 - 1 : Peripheral enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDIOE	CRCEN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]			MSTR	CPOL	CPHA

STM32F405 SPI 레지스터

- SPI_CR1 (Control register 1)

- BR[2:0] (Baud Rate Control)

- SPI의 동작에 사용되는 f_{pclk} 는 내부 PLL클럭 168MHz를 사용했을 때의 84MHz를 사용하게 됨. 이 비트들의 값으로 클럭의 분주비를 결정

- 000 : $f_{\text{pclk}} / 2$
 - 001 : $f_{\text{pclk}} / 4$
 - 010 : $f_{\text{pclk}} / 8$
 - 011 : $f_{\text{pclk}} / 16$
 - 100 : $f_{\text{pclk}} / 32$
 - 101 : $f_{\text{pclk}} / 64$
 - 110 : $f_{\text{pclk}} / 128$
 - 111 : $f_{\text{pclk}} / 256$

- 이 비트들은 통신중에 변경하면 안됨

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDIOE	CRCEN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]	MSTR	CPOL	CPHA		

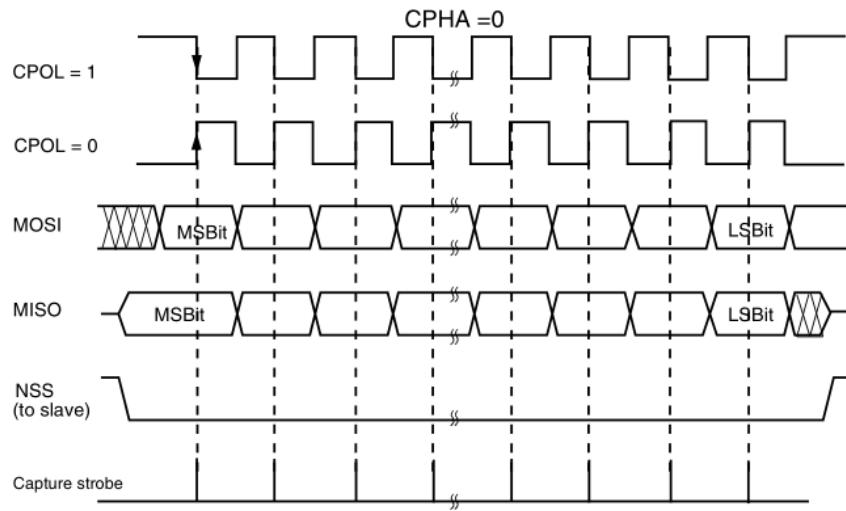
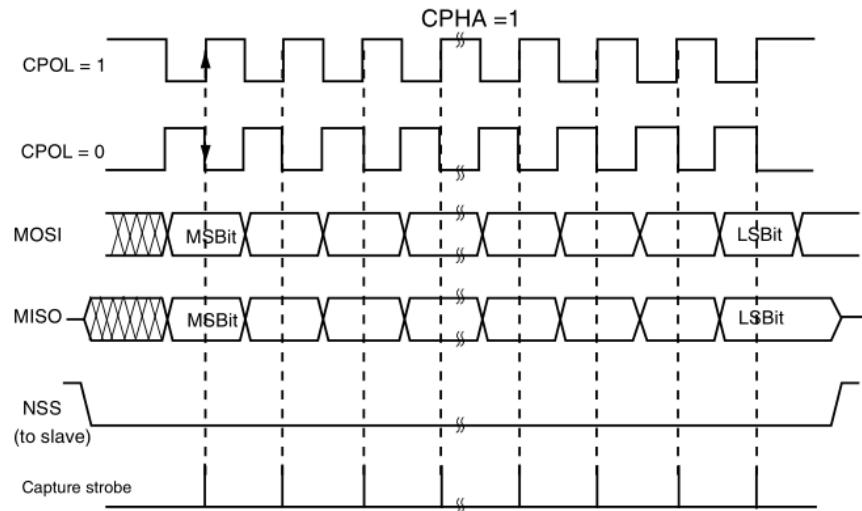
STM32F405 SPI 레지스터

- SPI_CR1 (Control register 1)
 - MSTR (Master Selection)
 - 0 : Slave configuration
 - 1 : Master configuration
 - 이 비트는 통신중에 변경하면 안됨
 - CPOL (Clock Ploarity)
 - 0 : SCK to 0 when idle
 - 1 : SCK to 1 when idle
 - 이 비트는 통신중에 변경하면 안됨
 - CPHA (Clock Phase)
 - 0 : 첫번째 클럭전송은 첫번째 데이터의 capture edge가 됨
 - 1 : 두번째 클럭전송은 첫번째 데이터의 capture edge가 됨
 - 이 비트는 통신중에 변경하면 안됨

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDIOE	CRCEN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]	MSTR	CPOL	CPHA		

STM32F405 SPI 레지스터

- SPI Data clock timing diagram



STM32F405 SPI 레지스터

- SPI_CR2 (Control register 2)
 - SPI 제어 레지스터 2
 - SPI 모듈의 동작을 설정하는 기능을 수행하는 레지스터
 - TXEIE (Tx buffer empty interrupt enable)
 - 0 : TXE 인터럽트를 허용하지 않음
 - 1 : TXE 인터럽트를 허용.
TXE 플래그가 세팅되어 있다면, 인터럽트를 발생
 - RXNEIE (Rx buffer not empty interrupt enable)
 - 0 : RXEN 인터럽트를 허용하지 않음
 - 1 : RXEN 인터럽트를 허용.
RXEN 플래그가 설정되어 있다면, 인터럽트를 발생

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	FRF	Res	SSOE	TXDMAEN	RXDMAEN

STM32F405 SPI 레지스터

- SPI_CR2 (Control register 2)
 - ERRIE (Error interrupt enable)
 - 이 비트는 error condition 이 발생되었을 때(CRCERR,OVRMODF), 인터럽트 발생을 제어
 - 0 : Error 인터럽트를 허용하지 않음
 - 1 : Error 인터럽트를 허용
 - SSOE (SS Output Enable)
 - 0 : SS 출력이 Disable된다(Master Mode)
 - 1 : SS 출력이 Enable 된다(Master Mode)
 - TXDMAEN (Tx buffer DMA enable)
 - RXDMAEN (Rx buffer DMA enable)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	FRF	Res	SSOE	TXDMAEN	RXDMAEN

STM32F405 SPI 레지스터

- SPI_SR (Status register)
 - SPI 상태 레지스터
 - SPI 모듈의 통신 상태를 알려주는 레지스터
 - BSY (Busy flag)
 - 이 비트는 하드웨어에 의해 설정
 - 0 : SPI 는 busy 상태가 아님
 - 1 : SPI 는 busy 상태이거나 Tx buffer가 비어 있지 않은 상태
 - OVR (Overrun flag)
 - 0 : Overrun 발생되지 않음
 - 1 : Overrun 발생
 - 이 비트는 하드웨어에 의해 세팅 되며, SPI_DR레지스터를 읽어 들임으로써 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	FRF	Res	SSOE	TXDMAEN	RXDMAEN

STM32F405 SPI 레지스터

- SPI_SR (Status register)
 - MODF (Mode fault)
 - Mode fault는 하드웨어 / 소프트웨어에 의해 NSS 핀이 pulled low가 되거나, SSI비트가 0이 되면 발생되며, 자동적으로 MODF비트가 설정
 - 0 : Mode fault 발생되지 않음
 - 1 : Mode fault 발생
 - 이 비트는 하드웨어에 의해 세팅 되며, SPI_CR1레지스터를 재설정함으로써 클리어
 - CRCERR (CRC error flag)
 - 0 : 수신된 CRC값이 SPI_RXCRCR값과 일치
 - 1 : 수신된 CRC값이 SPI_RXCRCR값과 일치하지 않음
 - 이 비트는 하드웨어에 의해 세팅 되며, 소프트웨어에 의해 클리어

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	FRF	Res	SSOE	TXDMAEN	RXDMAEN

STM32F405 SPI 레지스터

- SPI_SR (Status register) : SPI 상태 레지스터
 - TXE (Transmit buffer empty)
 - 0 : Tx buffer는 비어있지 않음
 - 1 : Tx buffer가 비어 있음
 - RXNE (Receive buffer not empty)
 - 0 : Rx buffer가 비어 있음
 - 1 : Rx buffer는 비어있지 않음

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								TXEIE	RXNEIE	ERRIE	FRF	Res	SSOE	TXDMAEN	RXDMAEN

STM32F405 SPI 레지스터

- SPI_DR (Data register)
 - SPI 데이터 레지스터
 - SPI 모듈의 통신 데이터를 저장하는 레지스터
 - DR[15:0] (Data register)
 - 이 레지스터는 2중 버퍼구조로 되어 있으며, 데이터를 보낼 때는 Transmit Buffer로, 데이터를 읽어들일 때는 Receive buffer로 동작
 - 그 외에도 CRC polynomial 레지스터인 SPI_CRCPR 레지스터와, CRC값을 저장하고 있는 SPI_TXCRCR, SPI_RXCRCR 레지스터가 있음

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DR[15:0]															

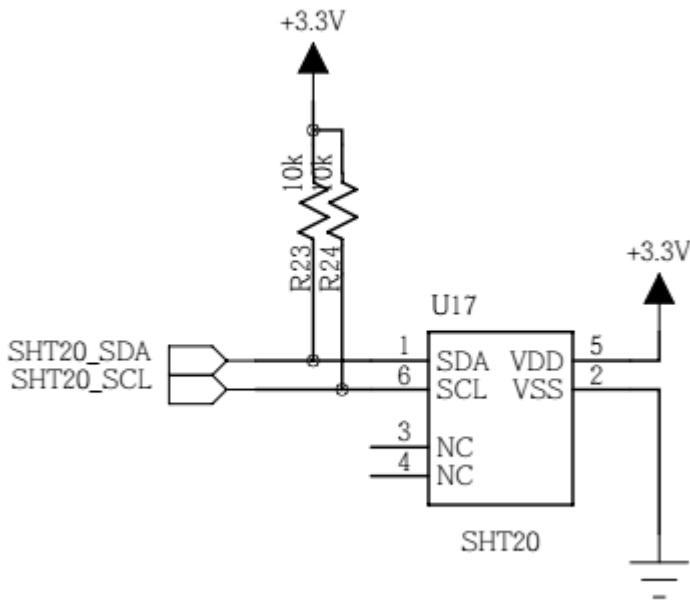
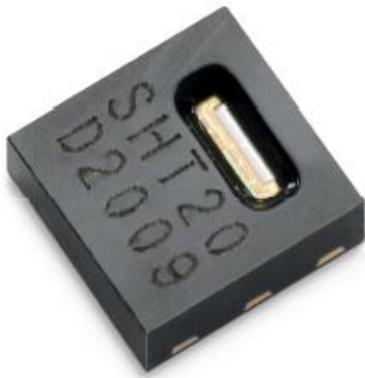
실습 1 : I²C로 온습도 센서 제어하기

- 실습 개요
 - I²C로 제어되는 온습도 센서인 SHT11을 STM32F405의 TWI 포트에 연결하고, 온도와 습도 정보를 읽어 들여 TEXT LCD에 표시
 - 제조사에서 제공하는 SHT20 칩의 TWI 인터페이스 관련 라이브러리 함수를 사용
- 실습 목표
 - TWI(I²C) 의 동작 원리 이해 (레지스터 설정)
 - SHT20 온도센서 제어 방법 습득

실습 1 : I²C로 온습도 센서 제어하기

● 사용 모듈

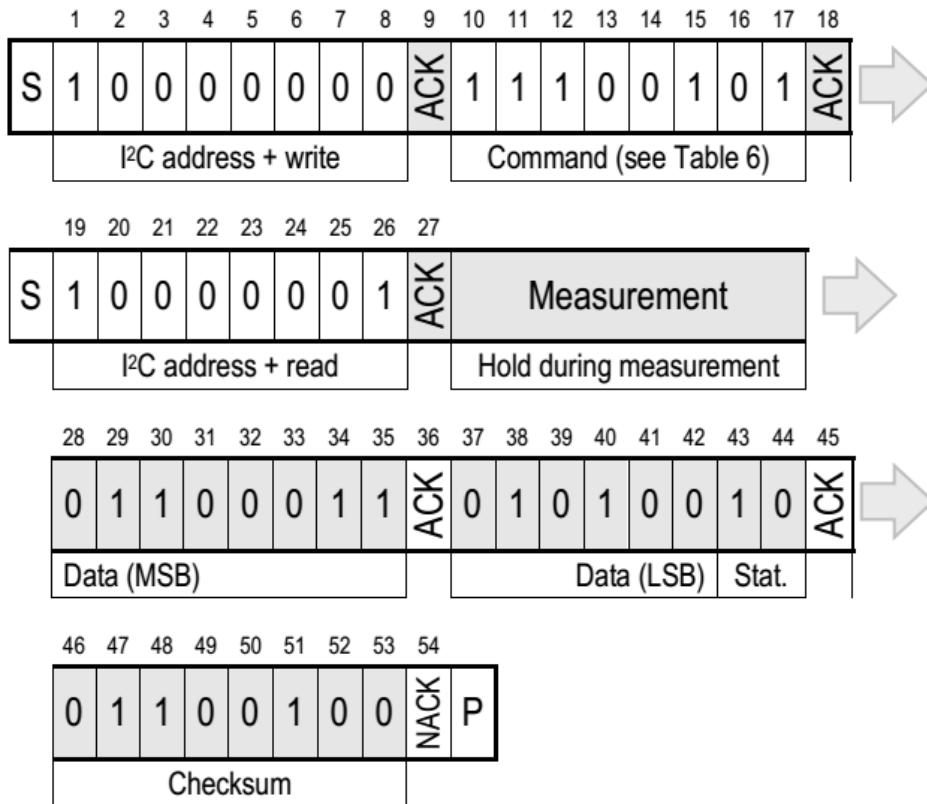
- 센서 모듈 중 온습도 센서 부분의 회로
 - SHT20
 - 상대습도와 상대 온도를 측정하는 칩이며, 이슬점(Dew Point) 측정이 가능함
 - 이 칩은 TWI(I2C) 인터페이스를 사용



실습 1 : I²C로 온습도 센서 제어하기

- 사용 모듈
 - SHT20 온습도 센서

SHT20의 (I²C)데이터 형식



실습 1 : I²C로 온습도 센서 제어하기

- I²C 라이브러리

- STMicroelectronics사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
- I2C_InitTypeDef 구조체

I2C_InitTypeDef구조체	설 명
uint32_t I2C_ClockSpeed	I ² C의 클럭 속도를 설정한다. 400KHz 이하로 설정해야 한다.
uint16_t I2C_Mode	I ² C의 모드를 설정한다. (I2C_CR1의 SMBTYPE, SMBUS, PE 비트)
uint16_t I2C_DutyCycle	I ² C의 Fast 모드의 클럭을 설정한다. (I2C_CCR 레지스터)
uint16_t I2C_OwnAddress1	자신의 어드레스(7bit 또는 10 bit)를 설정한다. (I2C_OAR1 레지스터)
uint16_t I2C_Ack	I ² C의 Acknowledge를 설정한다. (I2C_CR1의 ACK 비트)
uint16_t I2C_AcknowledgedAddress	자신의 어드레스의 비트수(7bit 또는 10 bit)를 설정한다. (I2C_OAR1의 ADDMODE 비트)

실습 1 : I²C로 온습도 센서 제어하기

- 구동 프로그램 : 사전 지식
 - SHT2x.c 에서 제공하는 라이브러리 함수
 - int8_t SHT2x_CheckCrc(uint8_t *data, uint8_t nbrOfBytes, uint8_t checksum)
 - SHT20으로부터 수신된 데이터의 체크섬을 계산
 - int8_t SHT2x_ReadUserRegister(uint8_t *pRegisterValue)
 - SHT20의 유저 레지스터를 읽어옴
 - void SHT2x_WriteUserRegister(uint8_t *pRegisterValue)
 - SHT20의 유저 레지스터에 데이터를 씁
 - int8_t SHT2x_MeasureHM(etSHT2xMeasureType eSHT2xMeasureType, nt16 *pMeasurand)
 - SHT20의 측정 데이터를 Hold Master 방식으로 읽어옴
(한번의 읽기시도)
 - int8_t SHT2x_MeasurePoll(etSHT2xMeasureType eSHT2xMeasureType, nt16 *pMeasurand)
 - SHT20의 측정 데이터를 Hold Master 방식으로 읽어옴
(10ms 마다 20번의 읽기시도)

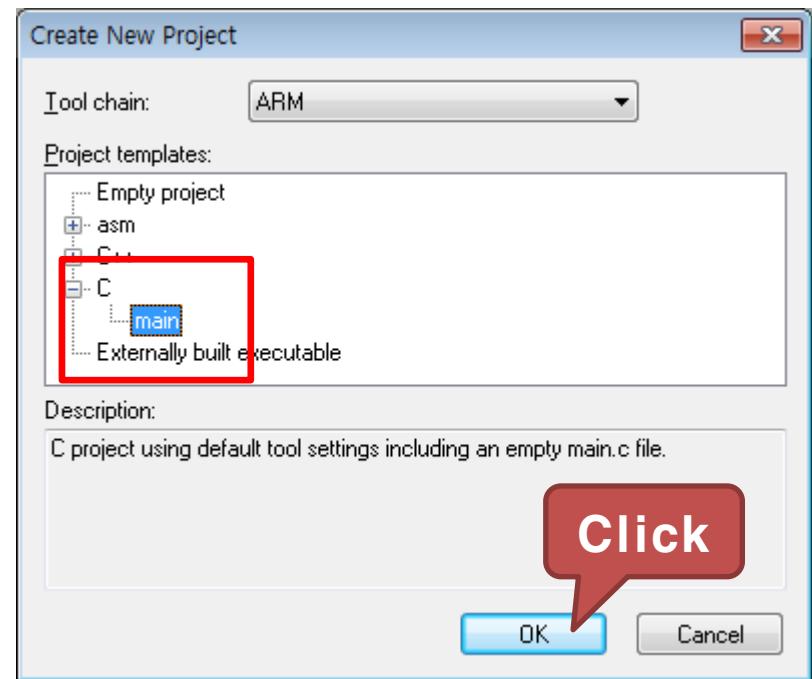
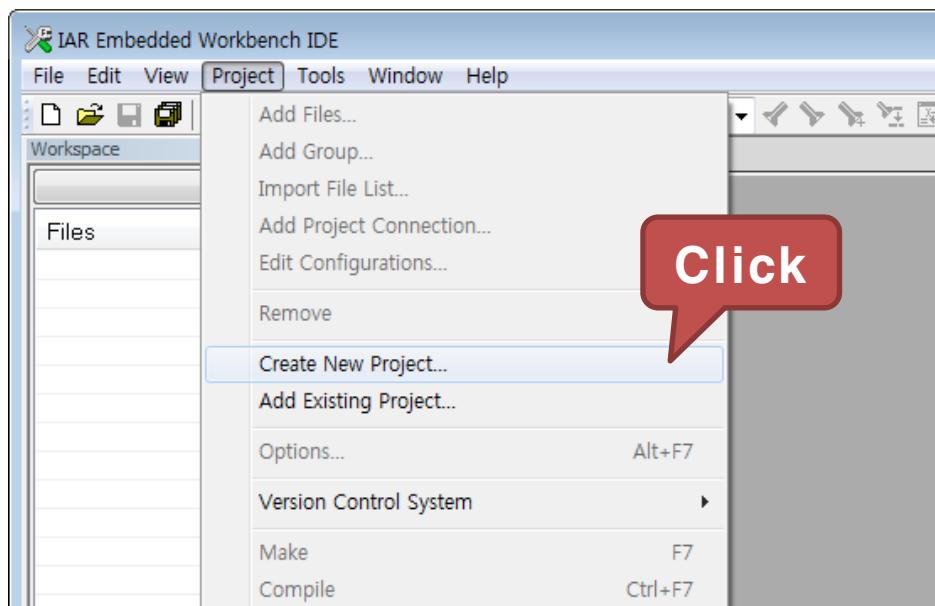
실습 1 : I²C로 온습도 센서 제어하기

- 구동 프로그램 : 사전 지식
 - SHT2x.c 에서 제공하는 라이브러리 함수
 - void SHT2x_SoftReset()
 - SHT20을 Software 리셋
 - float SHT2x_CalcRH(uint16_t u16sRH)
 - SHT20으로부터 읽어온 측정데이터를 실제 습도값으로 변환
 - float SHT2x_CalcTemperatureC(uint16_t u16sT)
 - SHT20으로부터 읽어온 측정데이터를 실제 온도값으로 변환
 - void SHT2x_GetSerialNumber(uint8_t *u8SerialNumber)
 - SHT20의 시리얼 넘버를 읽어옴
 - int8_t SHT2x_Init(void)
 - SHT20을 초기화

실습 1 : I²C로 온습도 센서 제어하기

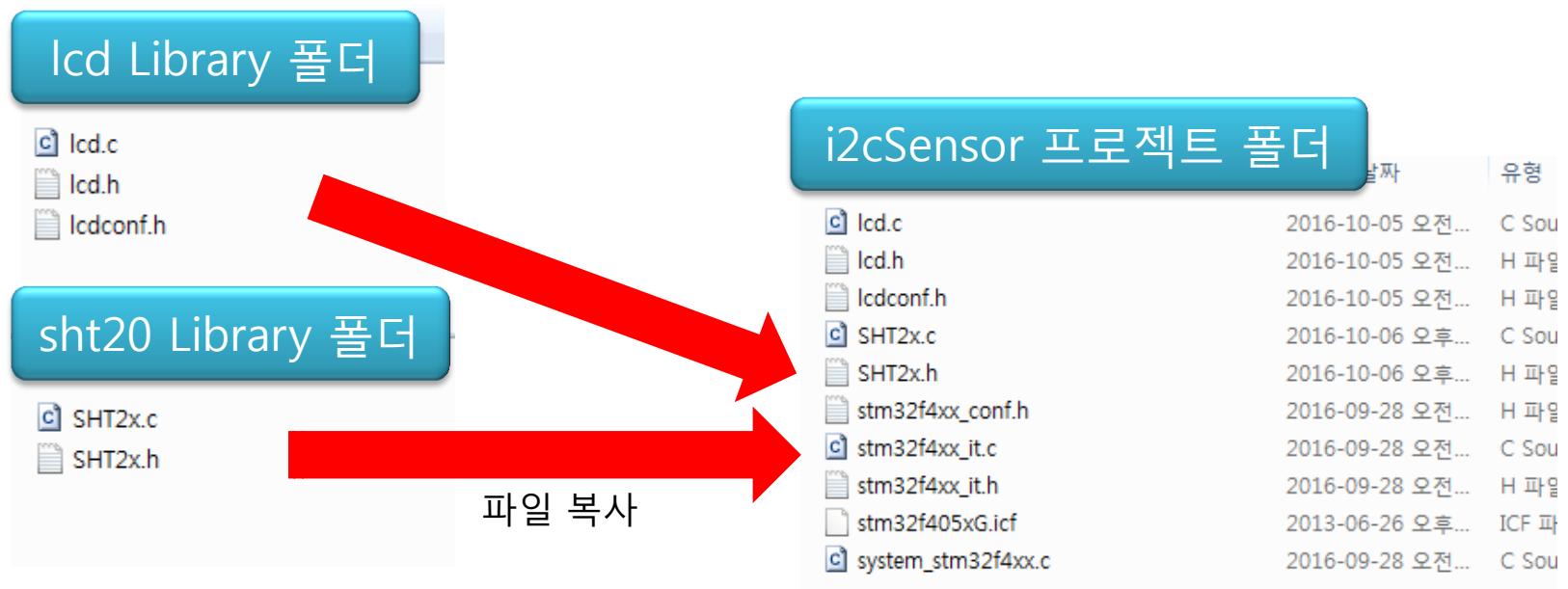
● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch10” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “i2cSensor”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1 : I²C로 온습도 센서 제어하기

- 구동 프로그램 : EWARM에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - SHT2x.c, SHT2x.h파일은 “Cortex_Example\Projects\library\sht20” 폴더에 존재
 - Lcd.c, Lcd.h, Lcdconf.h 파일은 “Cortex_Example\Projects\library\lcd” 폴더에 존재
 - 라이브러리 파일은 EWARM에서 새 프로젝트를 생성한 후 생성된 폴더에 복사



실습 1 : I²C로 온습도 센서 제어하기

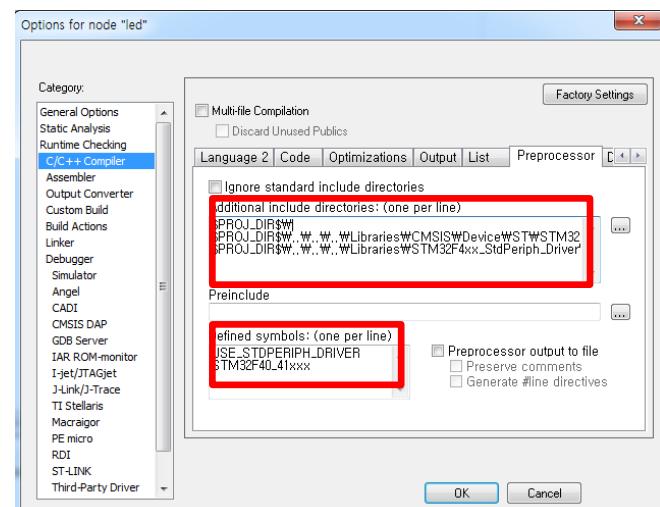
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch10\i2cSensor	system_stm32f4xx.c
Cortex_Example\Projects\ch10\i2cSensor	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Projects\ch10\i2cSensor	lcd.c
Cortex_Example\Projects\ch10\i2cSensor	SHT2x.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_i2c.c

실습 1 : I²C로 온습도 센서 제어하기

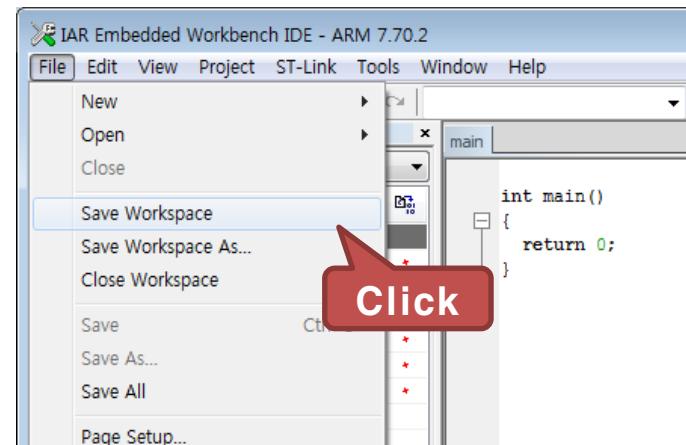
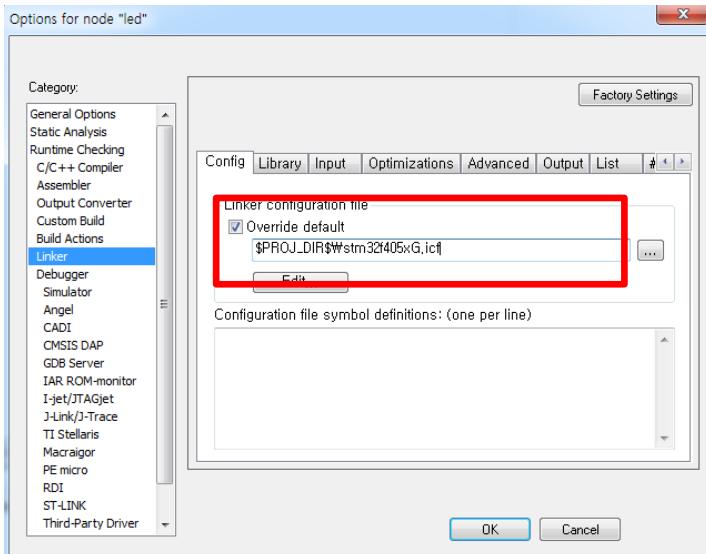
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1 : I²C로 온습도 센서 제어하기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1 : I²C로 온습도 센서 제어하기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"
#include "SHT2x.h"          // SHT2x 라이브러리 사용을 위한 헤더 파일
#include "lcd.h"            // Text LCD를 사용하기 위한 헤더 파일

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

// 온도 및 습도를 LCD에 출력하는 함수
void printf_2dot1(uint8_t sense,uint16_t sense_temp);
// 온도, 습도 값 측정에 사용되는 변수
unsigned int temperatureC,humidityRH;
```

실습 1 : I²C로 온습도 센서 제어하기

- 구동 프로그램
 - main.c 코드 작성

```
int main(void)
{
    uint8_t error = 0;          // 에러를 저장하는 변수
    nt16 sRH;                 // 습도의 raw 데이터를 저장하는 변수
    nt16 sT;                  // 온도의 raw 데이터를 저장하는 변수
    lcdInit();                // Text LCD를 초기화
    SHT2x_Init();              // SHT 센서를 초기화

    while(1) {
        error |= SHT2x_MeasureHM(HUMIDITY, &sRH);      // 습도를 측정
        error |= SHT2x_MeasureHM(TEMP, &sT);             // 온도를 측정
        // 온도 습도를 계산, 소숫점 첫째자리까지 출력하기 위해 10을 곱함
        temperatureC = SHT2x_CalcTemperatureC(sT.u16)*10; //온도를 계산
        humidityRH   = SHT2x_CalcRH(sRH.u16)*10; // 습도를 계산
    }
}
```

실습 1 : I²C로 온습도 센서 제어하기

- 구동 프로그램

- main.c 코드 작성

```
if(error == SUCCESS) {          // 에러없이 정상 측정될 경우
    lcdGotoXY(0,0);    // 커서위치를 첫번째 줄 첫번째 칸으로 이동
    printf_2dot1(TEMP,temperatureC); // 온도를 출력
    lcdGotoXY(0,1);    // 커서위치를 두번째 줄 첫번째 칸으로 이동
    printf_2dot1(HUMIDITY,humidityRH); // 습도를 출력
}
else {                         // 에러가 있을 경우
    lcdGotoXY(0,0);    // 커서위치를 첫번째 줄 첫번째 칸으로 이동
    lcdPrintData(" Temp: ---.-C",12); // 온도를 ---.-로 출력
    lcdGotoXY(0,1);    // 커서위치를 두번째 줄 첫번째 칸으로 이동
    lcdPrintData(" Humi: ---.-%",12); // 습도를 ---.-로 출력
}
error = 0;
Delay(300);                  // 다음 측정을 위한 시간 지연(300ms)
}
```

실습 1 : I²C로 온습도 센서 제어하기

- 구동 프로그램

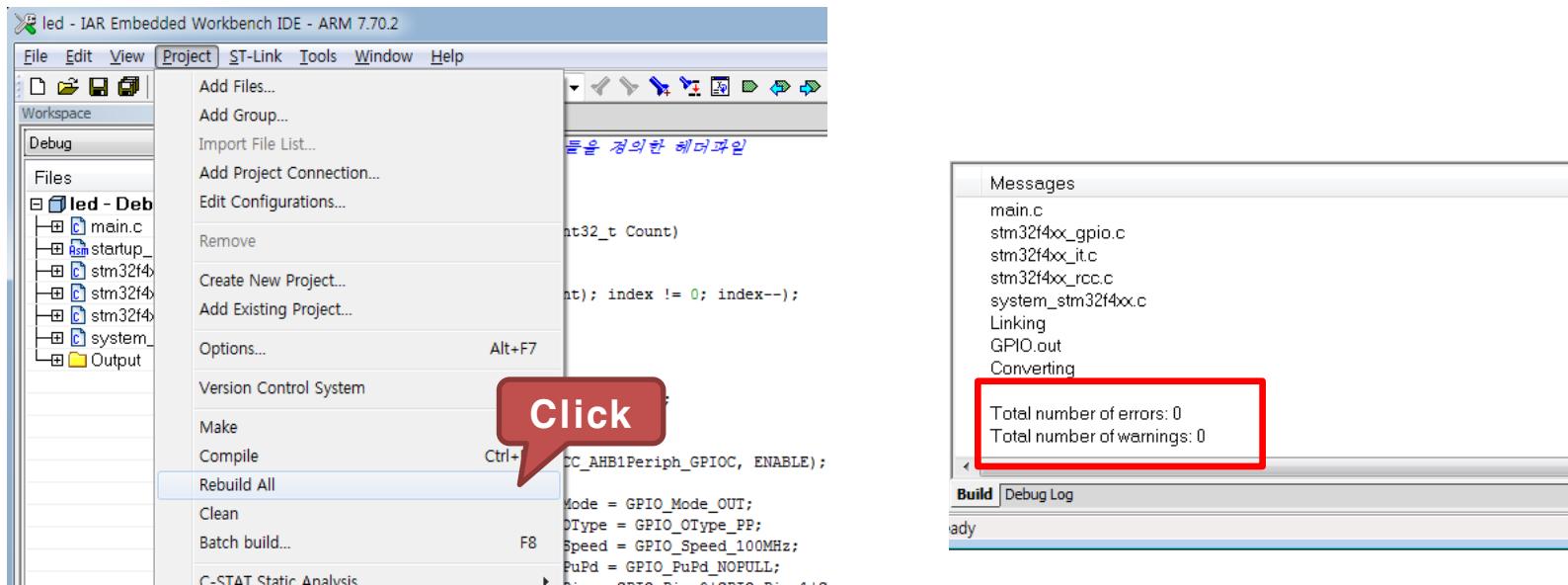
- main.c 코드 작성

```
void printf_2dot1(uint8_t sense,uint16_t sense_temp) {  
    uint8_t s100,s10;  
    if(sense == TEMP) lcdPrintData(" Temp: ",7);  
    // 온도 출력시 " Temp: " 출력  
    else if(sense == HUMIDITY) lcdPrintData(" Humi: ",7);  
    // 습도 출력시 " Humi: " 출력  
    s100 = sense_temp/100; // 100의 자리 추출  
    if(s100>0) lcdDataWrite(s100+'0'); // 100의 자리 값이 있으면 출력  
    else lcdPrintData(" ",1); // 100의 자리 값이 없으면 빈칸 출력  
    s10 = sense_temp%100; // 100의 자리를 제외한 나머지 추출  
    lcdDataWrite((s10/10)+'0'); // 10의 자리 추출하여 출력  
    lcdPrintData(".",1); // 소수점 출력  
    lcdDataWrite((s10%10)+'0'); // 1의 자리 추출하여 출력  
    if(sense == TEMP) lcdDataWrite('C'); // 온도 단위 출력  
    else if(sense == HUMIDITY) lcdDataWrite('%'); // 습도 단위 출력  
}
```

실습 1 : I²C로 온습도 센서 제어하기

● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 i2cSensor 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭

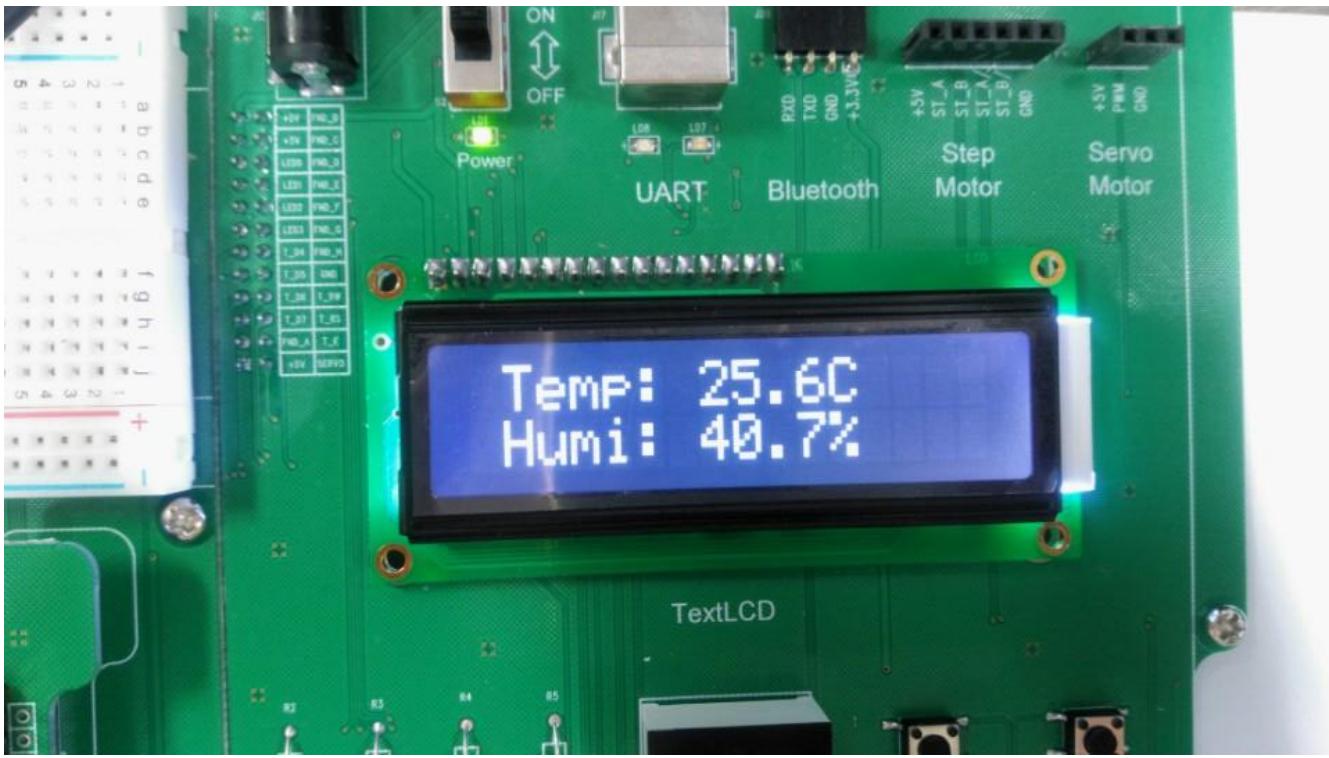


실습 1 : I²C로 온습도 센서 제어하기



실행 결과

- 온습도 센서로부터 온도 값 센서 값을 TextLCD에 출력
- 온습도 센서에 입김을 불거나, 손을 갖다대가면서 온도와 습도값이 변화하는지 확인



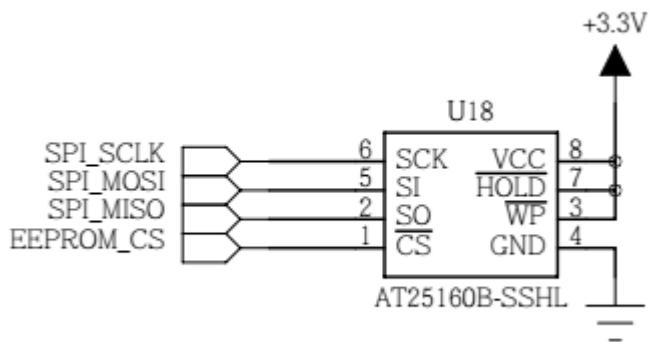
실습 2 : SPI로 EEPROM 불이기

- 실습 개요
 - 정해진 문자열을 SPI 인터페이스를 이용하여 Serial EEPROM Memory에 저장했다가, 이를 다시 꺼내어 TEXT LCD에 표시
 - SPI로 제어되는 EEPROM Memory인 AT25160B칩을 메모리로 사용
 - AT25160B 칩의 SPI 인터페이스 관련 라이브러리 함수는 at25160.c에 포함
- 실습 목표
 - SPI 인터페이스 동작 원리 이해 (레지스터 설정)
 - STM32F405의 SPI 포트에 대한 프로그램 방법 습득
 - Serial EEPROM Memory(AT25160B) 동작 원리 이해

실습 2 : SPI로 EEPROM 불이기

● 사용 모듈

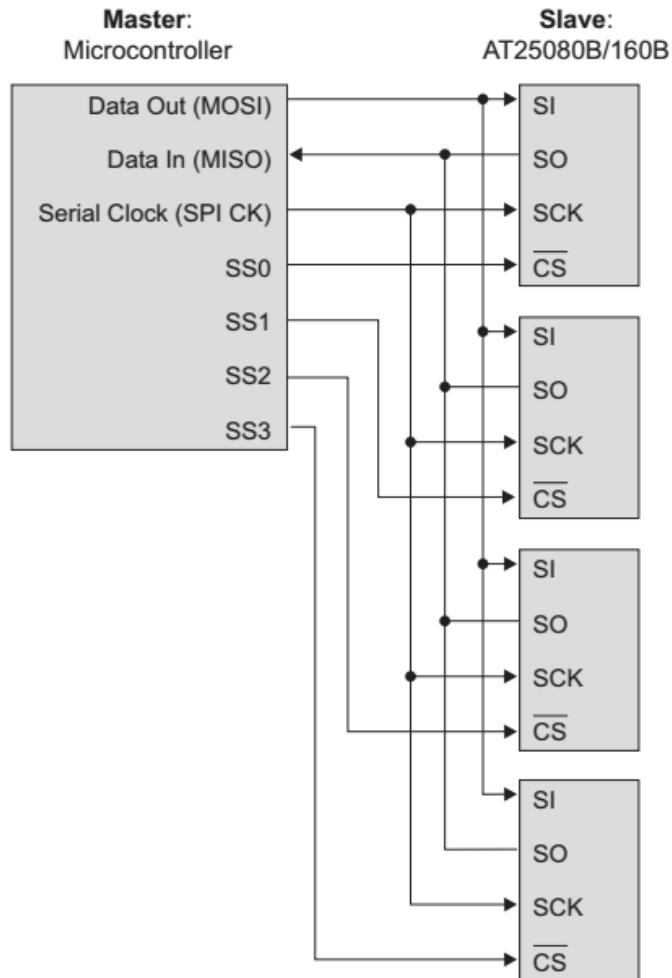
- 메모리 모듈의 SPI EEPROM 메모리 부분 회로
 - AT25160B
 - SPI를 지원하는 16K(2,048 x 8) 비트 용량의 Serial EEPROM memory
 - 32-Byte 크기의 페이지를 가지고 있으며, Write-Protect 기능을 가지고 있음



실습 2 : SPI로 EEPROM 불이기

● 사용 모듈

- AT25160B를 STM32F405에 연결하는 방법



실습 2 : SPI로 EEPROM 불이기

- 사용 모듈

- AT25160B 핀 설명

Pin Name	Function
\overline{CS}	Chip Select
GND	Ground
\overline{HOLD}	Suspends Serial Input
SCK	Serial Data Clock
SO	Serial Data Output
SI	Serial Data Input
\overline{WP}	Write Protect
Vcc	Power Supply

실습 2 : SPI로 EEPROM 불이기

- 사용 모듈
 - AT25160B 8비트 명령어 세트

Instruction Name	Instruction Format	Operation
WREN	0000 X 110	Ser Write Enable Latch
WRDI	0000 X 100	Reset Write Enable Latch
RDSR	0000 X 101	Read Status Register
WRSR	0000 X 001	Write Status Register
Read	0000 X 011	Read Data from Memory Array
Write	0000 X 010	Write Data to Memoy Array

실습 2 : SPI로 EEPROM 불이기

- AT25160B Serial EEPROM Memory 제어
 - AT25160B 칩의 임의의 메모리 주소로부터 데이터 읽어 오기
1. NSS핀을 low로 만듦(Chip-Select active)
 2. SPI_DR에 READ(0x03)명령을 넣어 AT25F512로 전송, 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 3. SPI_DR에 0x00을 넣음(최상위 어드레스, AT25F512A에서는 2바이트주소만 사용하기 때문), 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 4. SPI_DR에 상위 8비트 어드레스를 넣음, 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 5. SPI_DR에 하위 8비트 어드레스를 넣음, 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 6. SPI_DR에 0을 넣음(데이터 수신 대기), 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 7. SPI_SR레지스터의 RXNE(Receive buffer not empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기, SPI_DR에서 데이터를 읽어옴
 8. NSS를 High로 만듦(Chip-Select inactive)

실습 2 : SPI로 EEPROM 불이기

- AT25160B Serial EEPROM Memory 제어
 - AT25160B 칩의 임의의 메모리 주소에 데이터를 써 넣기(Program)
 1. NSS를 low로 만듦(Chip-Select active)
 2. SPI_DR에 WREN(0x06) 명령을 넣음으로써, AT25F512로 전송, 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 3. SPI_DR에 PROGRAM(0x00)을 넣음(쓰기선언), 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 4. SPI_DR에 0x00을 넣음(최상위 어드레스), 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 5. SPI_DR에 상위 8비트 어드레스를 넣음, 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 6. SPDR에 하위 8비트 어드레스를 넣음, 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 7. SPI_DR에 데이터를 넣음(데이터 송신), 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기

실습 2 : SPI로 EEPROM 불이기

- AT25160B Serial EEPROM Memory 제어
 - AT25160B 칩의 임의의 메모리 주소에 데이터를 써 넣기(Program)
8. SPI_DR에 RDSR(0x05)을 넣음(Status Register 읽기 선언), SPSR의 SPIF가 '1'로 세트될 때까지 대기
 9. SPI_DR에 0을 넣음(레지스터 내용 수신대기), 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 10. SPI_SR레지스터의 RXNE(Receive buffer not empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기, SPI_DR에서 데이터를 읽어옴
 11. SPI_DR에 RDSR(0x05)을 넣음(Status Register 읽기 선언), 전송하기 전에 SPI_SR 레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 12. SPI_DR에 0을 넣음(레지스터 내용 수신대기), 전송하기 전에 SPI_SR레지스터의 TXE(Transmit buffer empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기
 13. SPI_SR레지스터의 RXNE(Receive buffer not empty) 비트를 관찰하면서 '1'로 세트될 때까지 대기, SPI_DR에서 데이터를 읽어옴. 이 데이터는 칩상태 레지스터 값인데, RDY비트가 0으로 클리어 되어야 앞에서 한 프로그램 동작이 완료된 것, 만약 RDY비트가 1이라면 11~13의 과정을 반복
 14. NSS를 High로 만듦(Chip-Select inactive)

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : 사전 지식(SPI 인터페이스 제어)
 - SPI 초기화
 - SPI 신호로 사용할 핀들을 설정하고, 입출력을 결정
 - STM32F405가 마스터로 사용될 경우, MISO는 입력으로, 나머지 MOSI, SCK, /SS는 출력으로 설정
 - 동작 모드와 클럭 모드를 결정
 - MIDMODE, BIDIOE, RxONLY 비트를 세팅하여, SPI 사용핀과 입출력을 설정
 - DFF비트와 LSBFIRST비트를 설정해서 데이터 전송 순서를 결정 : 여기서는 AT25160B에 적용하기 위해, 8비트 단위 전송, MSB 먼저 보내도록 설정
 - CPOL과 CPHA 비트를 세팅하여, 클럭의 극성(Polarity)와 위상(Phase)을 결정 : 여기서는 그냥 디폴트('0')로 설정
 - 클럭의 주파수를 결정
 - SPI_CR1의 BR[2:0]를 설정 : SPI1은 APB2버스클럭(84MHz)을 사용, 32분주로 설정하여 2.625MHz로 설정
 - SSM, SSI, MSTB 비트를 설정, NSS핀 제어를 소프트웨어에서 하도록 함
 - CRC관련 설정은 디폴트를 사용
 - SPI_CR1의 SPE 비트를 '1'로 세팅하여 SPI를 Enable

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : 사전 지식
 - AT25160B칩을 위한 라이브러리 함수
 - SPI_Init()
 - SPI를 초기화
 - at25160_Write_Byte(unsigned int addr, unsigned char data)
 - 임의의 주소에 1바이트 데이터를 써넣음. 인수로 주소와 써넣을 1바이트 데이터를 받음
 - unsigned char at25160_Read_Byte(unsigned int addr)
 - 인수로 주어진 주소의 1 바이트 데이터를 읽어냄
 - void at25160_Write_Arry(unsigned int addr, unsigned char* BPdata, unsigned char size)
 - 인수로 주어진 주소로부터 size만큼 BPData에 존재하는 데이터를 써넣음
 - void at25160_Read_Arry(unsigned int addr, unsigned char* BPbuf, unsigned char size)
 - 인수로 주어진 주소로부터 size만큼의 데이터를 읽어 BPbuff에 넘겨줌

실습 2 : SPI로 EEPROM 불이기

- SPI 라이브러리
 - STMicroelectronics 사에서는 STM32F4xx 시리즈에 사용할 수 있는 라이브러리를 제공
 - SPI_InitTypeDef 구조체

SPI_InitTypeDef구조체	설 명
uint16_t SPI_Direction	통신 라인 설정 및 Tx,Rx설정 (SPI_CR1의 BIDIMODE,BIDIOE,RXONLY비트)
uint16_t SPI_Mode	마스터,슬레이브 선택(SPI_CR1의 SSI,MSTR비트)
uint16_t SPI_DataSize	통신 데이터 사이즈 설정,8,16비트 (SPI_CR1의 DFF비트)
uint16_t SPI_CPOL uint16_t SPI_CPHA	통신 클럭의 Polarity, Phase 를 설정한다. (SPI_CR1의 CPOL,CPHA)
uint16_t SPI_NSS	NSS핀의 제어 방법(S/W,H/W)을 선택한다. (SPI_CR1의 SSM비트)
uint16_t SPI_BaudRatePrescaler	SPI통신의 클럭분주비를 결정한다. (SPI_CR1의 BR비트 (2,4,816,32,64,128,256))
uint16_t SPI_FirstBit	프레임 전송 방법을 선택한다. (MSB,LSB) (SPI_CR1의 LSBFIRST비트)
uint16_t SPI_CRCPolynomial	CRC 계산에 대한 다항식값을 포함하는 SPI_CRCPR 레지스터값을 설정. (여기서는 초기값 그대로 설정 : 7)

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : AT25160B칩을 위한 라이브러리 함수
 - SPI 초기화

```
void SPI_Init() {  
    GPIO_InitTypeDef GPIO_InitStructure;  
    SPI_InitTypeDef SPI_InitStructure;  
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);  
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);  
  
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;  
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;  
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;  
    // PA5(SPI1_SCK), PA6(SPI1_MISO), PA7(SPI1_MOSI)  
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7;  
    GPIO_Init(GPIOA, &GPIO_InitStructure);
```

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : AT25160B칩을 위한 라이브러리 함수
 - SPI 초기화

```
//...  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;           // PA4(SPI1_NSS)  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
  
// SYNC핀을 HIGH로 설정, 칩선택 해제  
GPIO_SetBits(GPIOA, GPIO_Pin_4);  
  
// SPI1 SCK  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);  
// SPI1 MISO  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);  
// SPI1 MOSI  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);
```

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : AT25160B칩을 위한 라이브러리 함수
 - SPI 초기화

```
//...
SPI_InitStructure.SPI_Direction =
    SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler =
    SPI_BaudRatePrescaler_32;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_Init(SPI1, &SPI_InitStructure);
SPI_SSOoutputCmd(SPI1, ENABLE);           // NSS(SYNC) 핀을 사용
SPI_Cmd(SPI1, ENABLE);
}
```

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : AT25160B칩을 위한 라이브러리 함수
 - SPI EEPROM 칩 쓰기 활성화

```
#define AT25160_CS_LOW  GPIO_ResetBits(GPIOA, GPIO_Pin_4)
#define AT25160_CS_HIGH GPIO_SetBits(GPIOA, GPIO_Pin_4)

void at25160_WREN ()
{
    AT25160_CS_LOW;           // 칩 셀렉트
    SPI_EEPROM_SendByte(WREN); // wren명령
    AT25160_CS_HIGH;         // cs high 가 되어야 다음부터 쓰기 실행
}
```

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : AT25160B칩을 위한 라이브러리 함수
 - SPI EEPROM 칩 명령 대기

```
void at25160_Ready()
{
    unsigned char data;
    do{
        AT25160_CS_LOW;
        SPI_EEPROM_SendByte(RDSR);
        data = SPI_EEPROM_SendByte(0x00);
        SPI_EEPROM_SendByte(0x00);
        AT25160_CS_HIGH;
    }while((data & 1<<(RDY)));
    // 레지스터를 읽어와 준비상태가 되어 있을 때 까지 루프
}
```

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : AT25160B칩을 위한 라이브러리 함수
 - SPI EEPROM 칩 1바이트 읽기

```
unsigned char at25160_Read_Byte(unsigned int addr)
{
    unsigned char data=0;
    AT25160_CS_LOW;

    SPI_EEPROM_SendByte(READ);           // 읽기선언
    SPI_EEPROM_SendByte(0x00);
    SPI_EEPROM_SendByte((addr>>8) & 0xff); // 주소
    SPI_EEPROM_SendByte((addr) & 0xff);   // 주소
    data = SPI_EEPROM_SendByte(0x00);     // 데이터

    AT25160_CS_HIGH;
    return data;
}
```

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : AT25160B칩을 위한 라이브러리 함수
 - SPI EEPROM 칩 1바이트 쓰기

```
void at25160_Write_Byte(unsigned int addr, unsigned char data)
{
    at25160_WREN ();
    AT25160_CS_LOW;

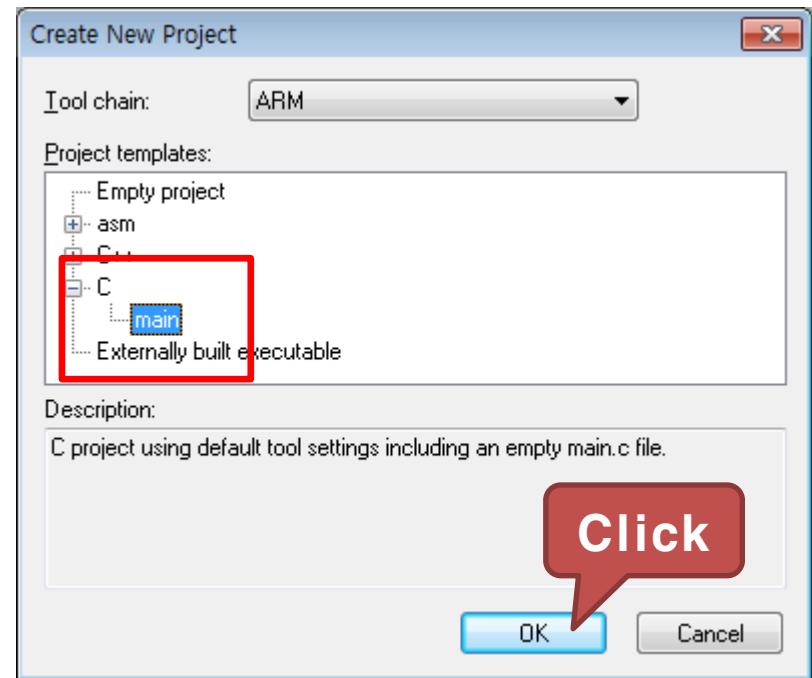
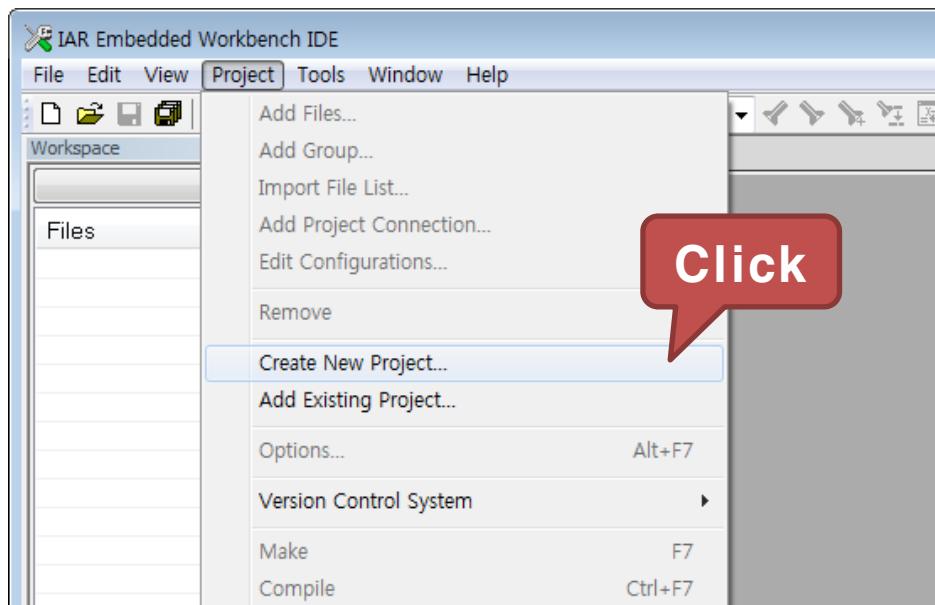
    SPI_EEPROM_SendByte(PROGRAM);           // 쓰기 선언
    SPI_EEPROM_SendByte(0x00);
    // at25160B는 여기 주소를 사용되지 않음 (16비트주소만 사용)
    SPI_EEPROM_SendByte((addr>>8) & 0xff); // 주소
    SPI_EEPROM_SendByte((addr) & 0xff);
    SPI_EEPROM_SendByte(data);              // 데이터

    AT25160_CS_HIGH;                      // cs high 가 되어야 입력한 내용이 적용
    at25160_Ready();
}
```

실습 2 : SPI로 EEPROM 불이기

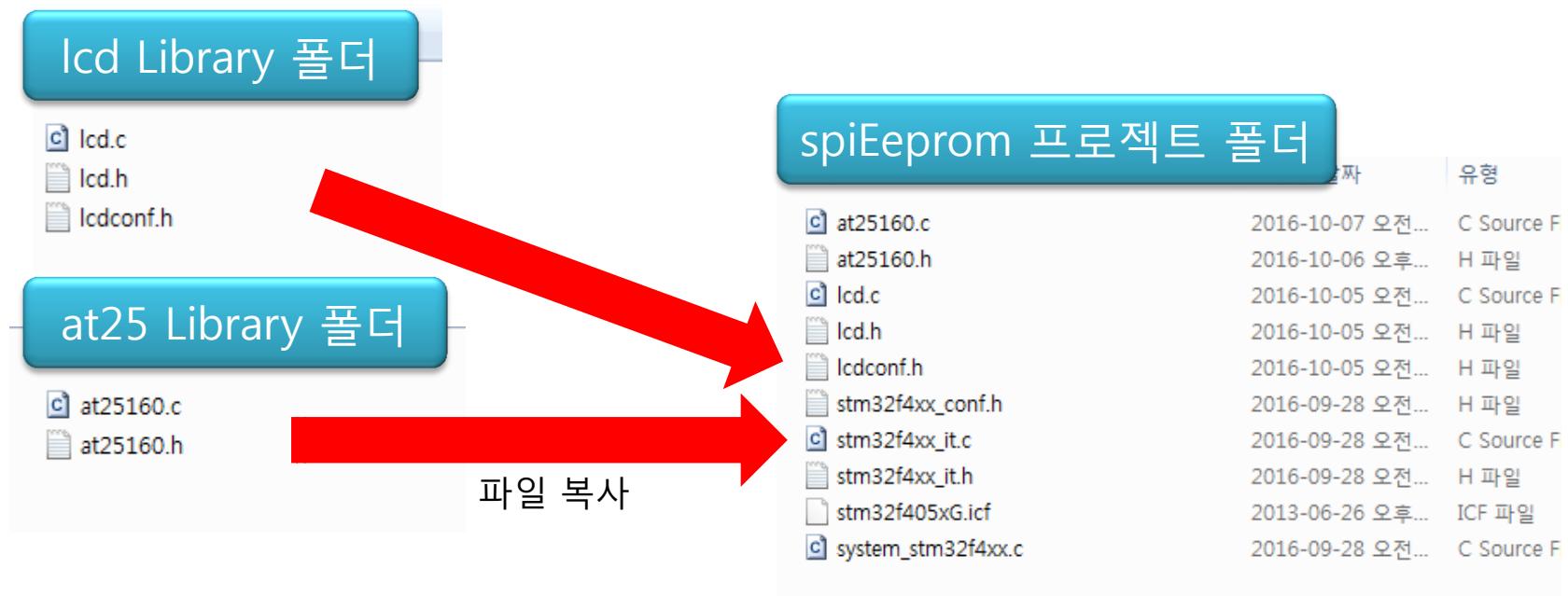
● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch10” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “i2cSensor”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램 : EWARM에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - at25160.c, at25160.h 파일은 "Cortex_Example\Projects\library\at25" 폴더에 존재
 - lcd.c, lcd.h, lcdconf.h 파일은 "Cortex_Example\Projects\library\lcd" 폴더에 존재
 - 라이브러리 파일은 EWARM에서 새 프로젝트를 생성한 후 생성된 폴더에 복사



실습 2 : SPI로 EEPROM 불이기

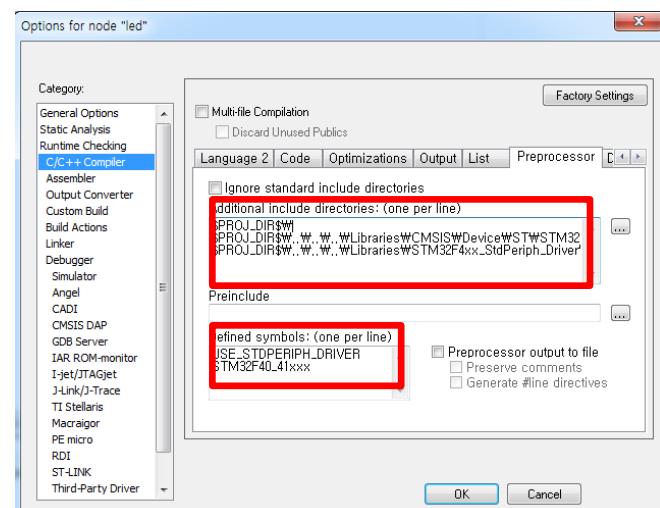
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch10\spiEeprom	system_stm32f4xx.c
Cortex_Example\Projects\ch10\spiEeprom	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Projects\ch10\spiEeprom	lcd.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_spi.c

실습 2 : SPI로 EEPROM 불이기

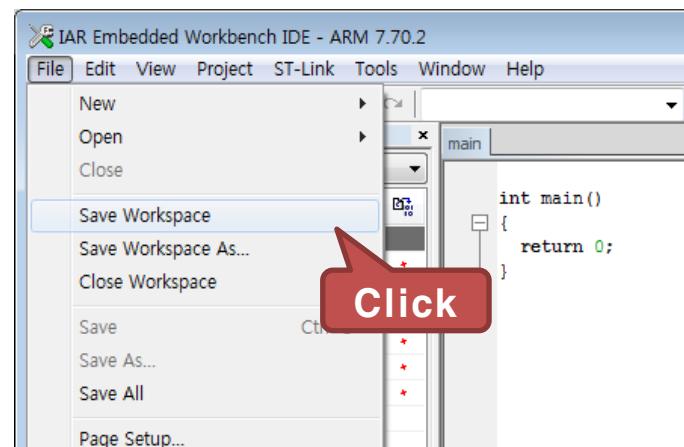
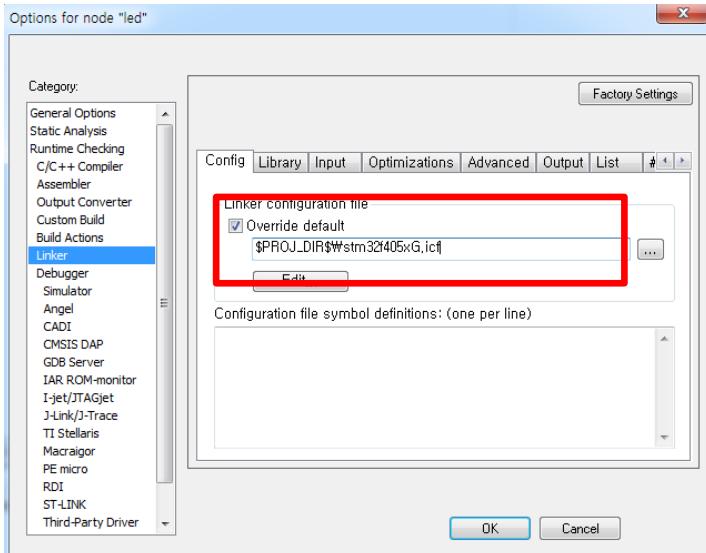
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : SPI로 EEPROM 불이기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"
#include "at25160.h"      // AT25160B 라이브러리 사용을 위한 헤더 파일
#include "lcd.h"          // Text LCD를 사용하기 위한 헤더 파일

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

#define ARRAY_SIZE(array) (sizeof(array)/sizeof(array[0]))

unsigned char msg1[] = "Welcome!!";
unsigned char msg2[] = "edgeiLAB-World!!";
unsigned char msg3[] = "SPI-Flash Exam";
```

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램
 - main.c 코드 작성

```
int main(void) {
    unsigned char i=0;
    unsigned char buf1[20]={0};
    unsigned char buf2[20]={0};
    unsigned char buf3[20]={0};

    SPI1_Init();
    lcdInit();

    // "Welcome!!" 저장
    at25160_Write_Arry(0x0100, msg1,ARRAY_SIZE(msg1));
    // "edgeiLAB-World!!" 저장
    at25160_Write_Arry(0x0200, msg2,ARRAY_SIZE(msg2));
    // "SPI-Flash Exam" 저장
    at25160_Write_Arry(0x0300, msg3,ARRAY_SIZE(msg3));
```

실습 2 : SPI로 EEPROM 불이기

- 구동 프로그램

- main.c 코드 작성

```
// 플래쉬에 저장한 문자열을 읽어와서 각 배열에 저장
at25160_Read_Arry(0x0100, buf1,ARRAY_SIZE(msg1));
at25160_Read_Arry(0x0200, buf2,ARRAY_SIZE(msg2));
at25160_Read_Arry(0x0300, buf3,ARRAY_SIZE(msg3));

// 읽어들인 문자열 출력
for(lcdGotoXY(0, 0);i<ARRAY_SIZE(msg1)-1;i++) {
    lcdDataWrite(buf1[i]);
    Delay(100);
}
for(i=0,lcdGotoXY(0, 1);i<ARRAY_SIZE(msg2)-1;i++){
    lcdDataWrite(buf2[i]);
    Delay(100);
}
```

실습 2 : SPI로 EEPROM 불이기

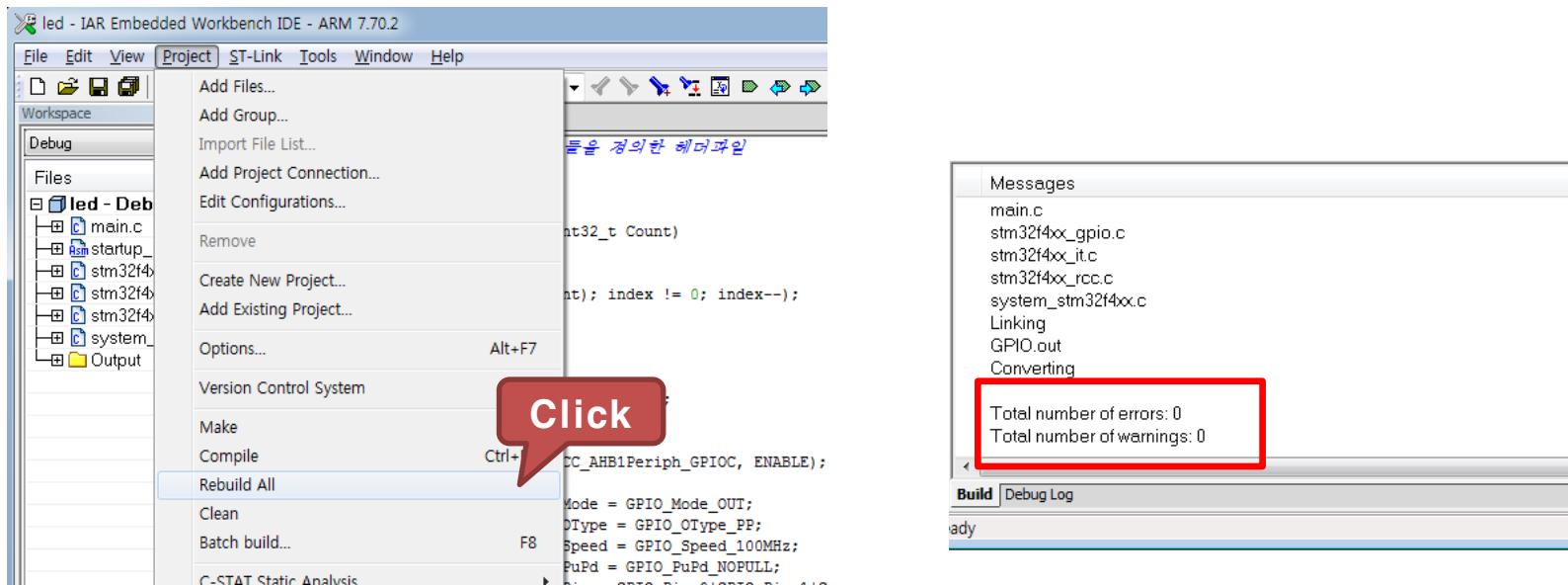
- 구동 프로그램
 - main.c 코드 작성

```
//...
for(i=0;i<ARRAY_SIZE(msg3)-1;i++){
    lcdDataWrite(buf3[i]);
    Delay(100);
}
for(i=0;i<16;i++) { // LCD 화면을 왼쪽으로 시프트
    lcdControlWrite(1<<LCD_MOVE | 1<<LCD_MOVE_DISP );
    Delay(100);
}
while(1)
{
}
}
```

실습 2 : SPI로 EEPROM 불이기

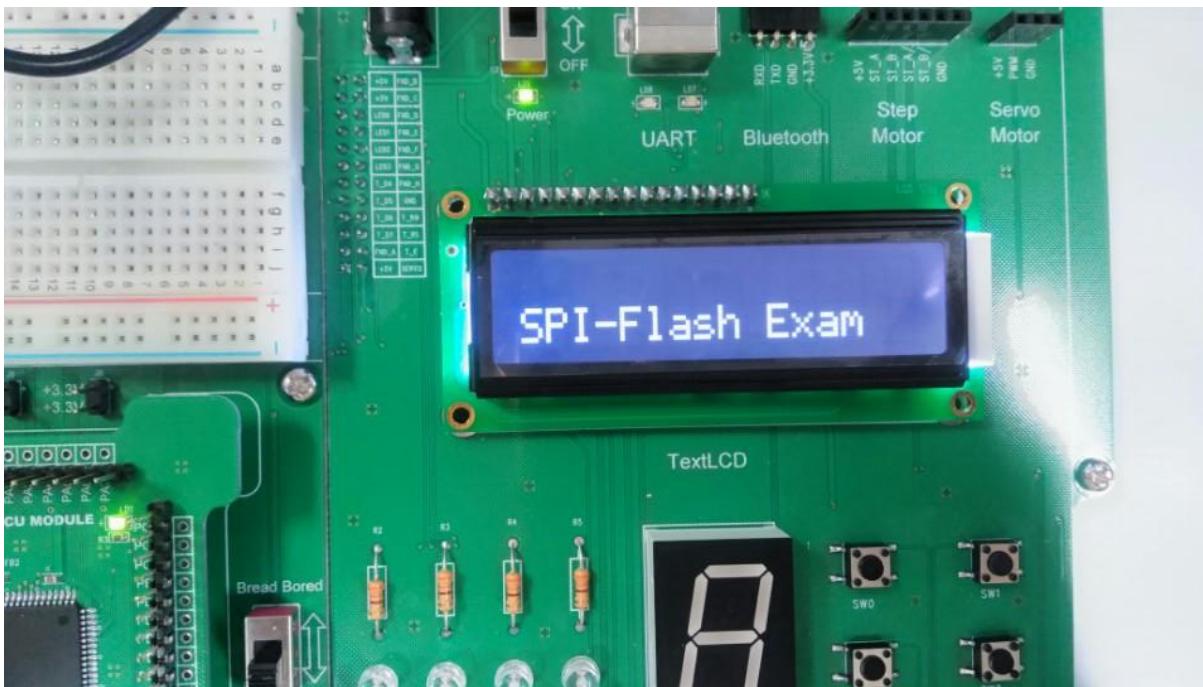
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 spiEeprom 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 2 : SPI로 EEPROM 불이기

- 실행 결과
 - "Welcome! edgeiLAB-World!! SPI-FLASH Exam" 문자열을 LCD에 출력



DC MOTOR

- DC Motor의 개요 및 원리
- DC Motor의 속도 제어 방법
- DC Motor 정회전, 역회전 시키기
- DC Motor 정지 시키기
- PWM을 이용하여 DC 모터 속도 제어하기 1
- PWM을 이용하여 DC 모터 속도 제어하기 2



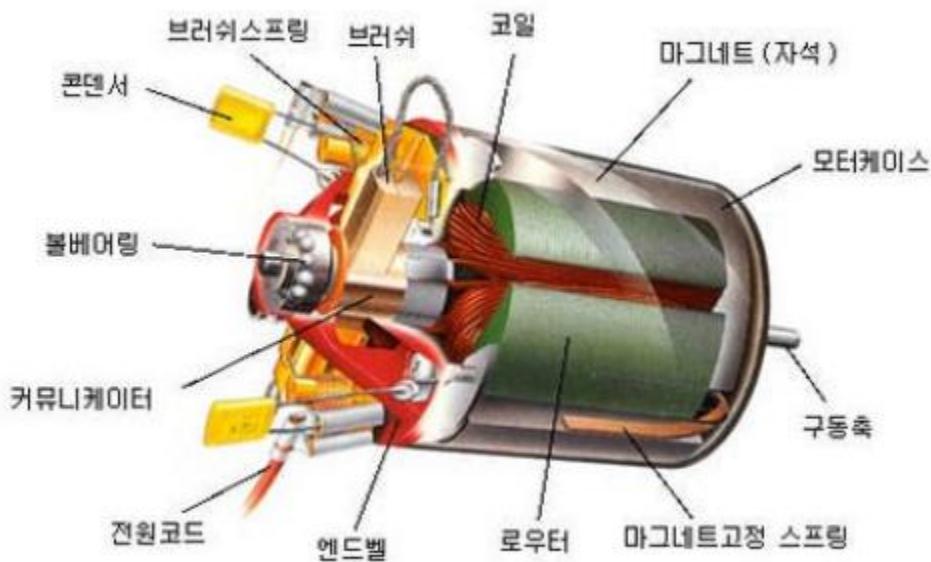
엣지아이랩

- DC 모터 개요
 - DC 모터란?
 - 고정자로 영구자석을 사용하고, 회전자(전기자)로 코일을 사용하여 구성한 것으로, 전기자에 흐르는 전류의 방향을 전환함으로써 자력의 반발, 흡인력으로 회전력을 생성시키는 모터
 - 특징
 - 기동 토크가 큼
 - 인가 전압에 대하여 회전 특성이 직선적으로 비례
 - 입력 전류에 대하여 출력 토크가 직선적으로 비례하며, 출력 효율이 양호
 - 저렴한 가격
 - 제어회로가 단순하고 제어하기 쉬움
 - 구조상 브러시(brush)와 정류자(commutator)에 의한 기계식 접점을 가지고 있어, 전기 불꽃(spark), 회전 소음, 수명 단축 등의 단점

DC 모터 개요 및 원리

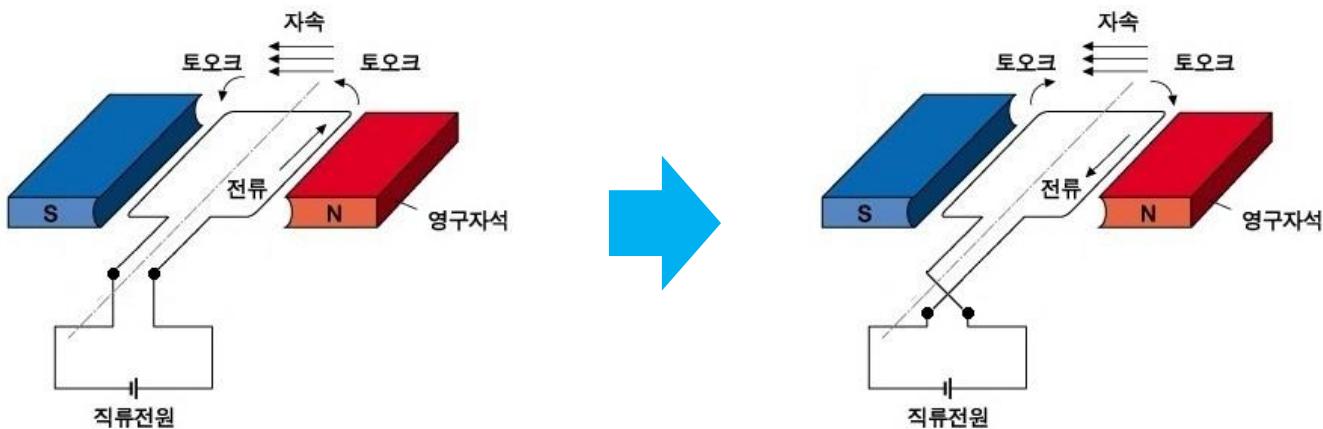
● DC 모터의 구조

- DC모터의 구조는 크게 전기자(armature, 로터), 영구자석(permanent magnet, 계자용 마그넷), 브러시(brush), 베어링 모터 케이스 등으로 이루어져 있음



DC 모터 개요 및 원리

- DC 모터 구동 원리
 - 플래밍의 왼손법칙에 의해 전류가 흐르는 도선은 시계 반대 방향으로 회전

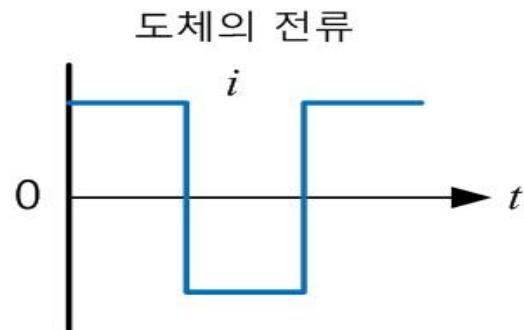
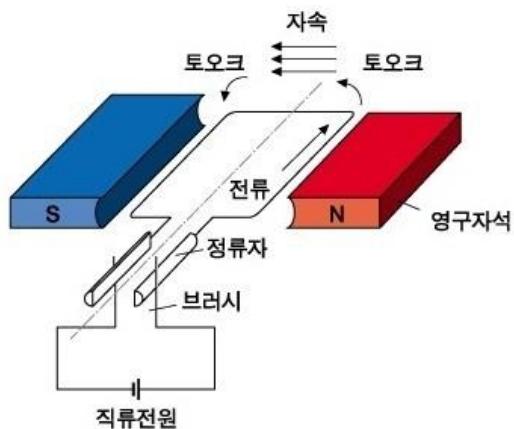


- 그러나 도선이 회전하여 왼쪽 그림과 같이 위치하게 되면 시계 방향으로 회전하려는 힘 때문에 도선은 초기 위치로 돌아가게 됨

DC 모터 개요 및 원리

● DC 모터 구동 원리

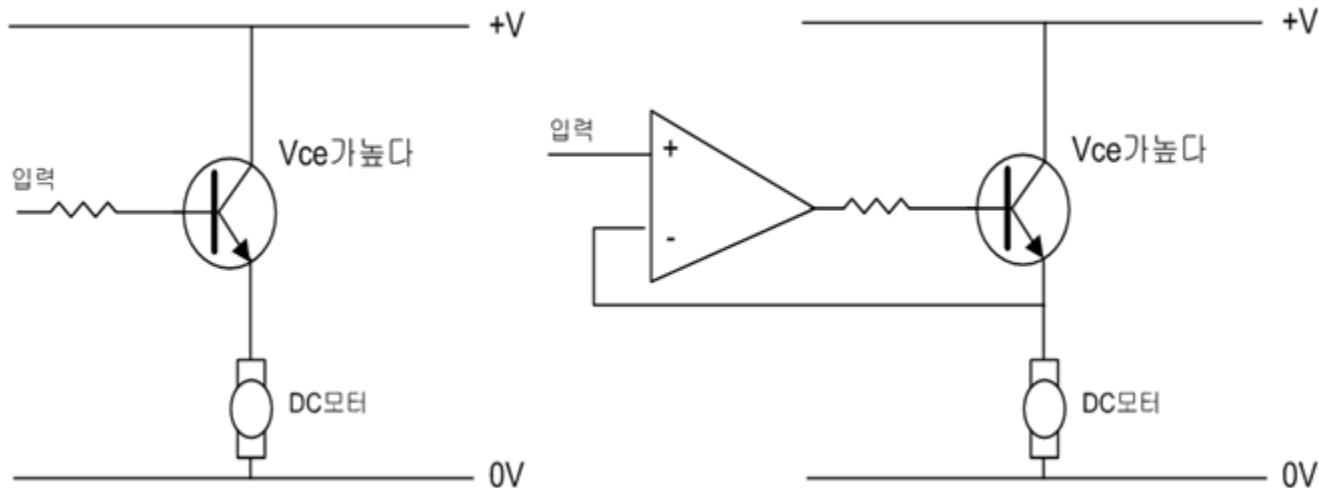
- 도선에 전류를 흘려 주기 위한 브러시(Brush)와 정류자(Commutator)가 추가된 그림을 보자. 플래밍의 왼손 법칙에 의해 도선은 시계 반대 방향으로 회전하게 되지만, 도선이 회전하더라도 정류자와 브러시에 의해 도선에 흐르는 전류는 뒤바뀌지 않게 되어 한쪽 방향으로 연속해서 회전할 수 있게 됨



DC 모터 속도 제어 방법

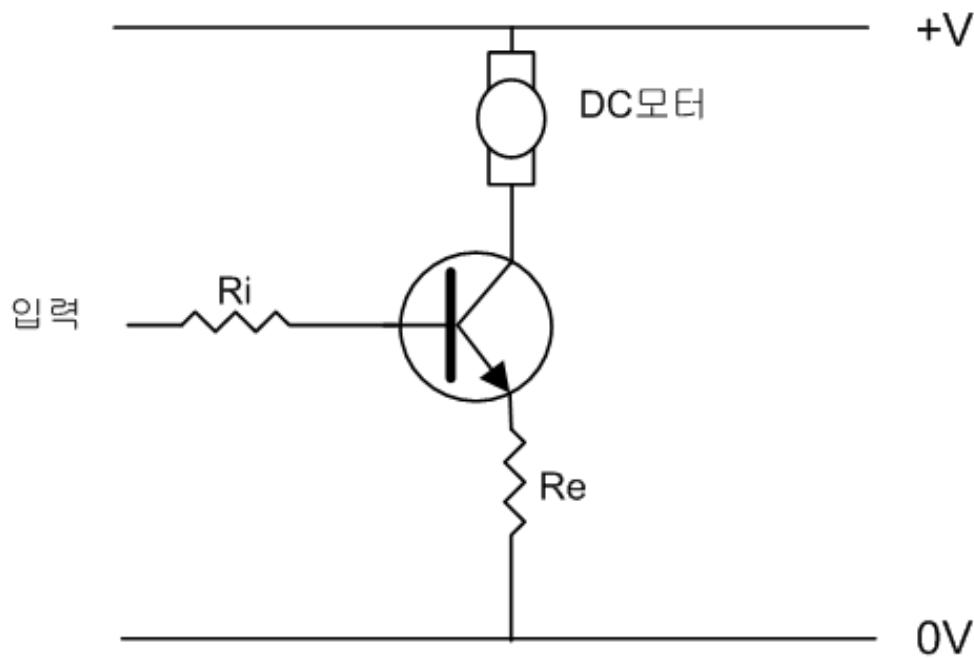
- 트랜지스터 구동(이미터 부하)

- 아래 [그림]의 회로에 의해 트랜지스터를 On/Off함으로써 모터를 On/Off
- 이 회로는 트랜지스터가 완전히 포화 되는 On 상태로는 할 수 없고, V_{ce} 가 크기 때문에 전압 손실이 커지게 됨. 동작으로서는 자동적으로 부귀환이 동작하기 때문에 동작은 안정적
- 이 때문에 간단한 속도 제어를 하기 위해 OP 앰프를 추가한 회로가 사용
 - 이 경우, 트랜지스터에서의 전력손실이 그대로 열로 되기 때문에 트랜지스터의 열 대책 고려



DC 모터 속도 제어 방법

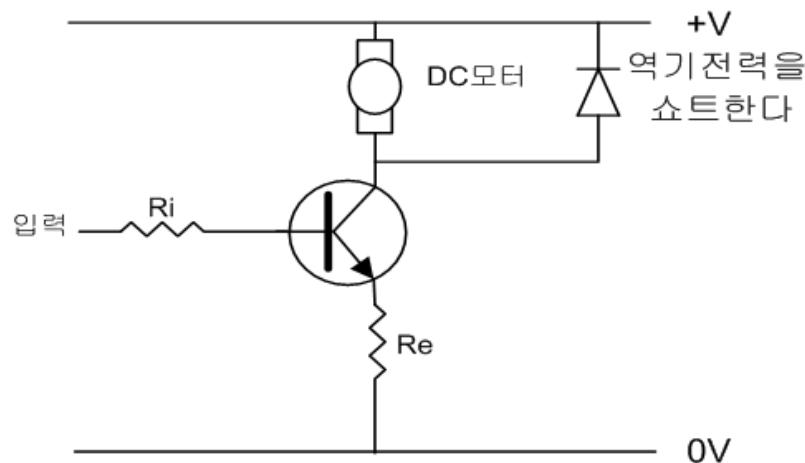
- 트랜지스터 구동(컬렉터 부하)
 - 모터를 트랜지스터 컬렉터의 부하로 이용한 것으로, 트랜지스터가 완전히 포화된 On 상태로 구동할 수 있기 때문에 드라이브 능력이 크고 전압 손실도 적게 할 수 있음
 - 일반적으로는 이 회로가 많이 사용



DC 모터 속도 제어 방법

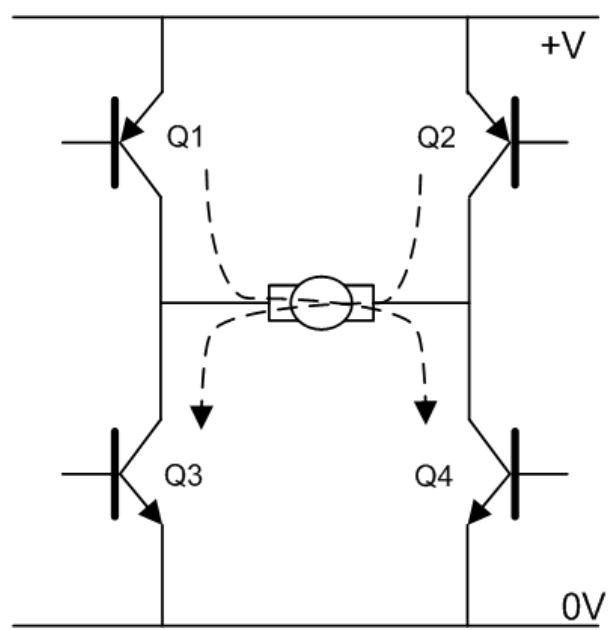
● 역기전력의 처리

- 모터가 회전하고 있는 동안에는 모터의 코일에 에너지가 축적되고, 트랜지스터가 Off로 되면 그 에너지를 방출하려고 하기 때문에, 모터 코일의 양단에는 플러스, 마이너스가 역방향의 기전력이 발생
 - 이 전압은 매우 크기 때문에 그대로는 트랜지스터가 파괴되어 버리는 경우도 있음
- 모터 양단에 다이오드를 달아서 코일을 쇼트시켜 남아 있는 에너지를 순간적으로 전류로서 흘려 버리는 식으로 해서, 역기전력을 억제하도록 함



DC 모터 속도 제어 방법

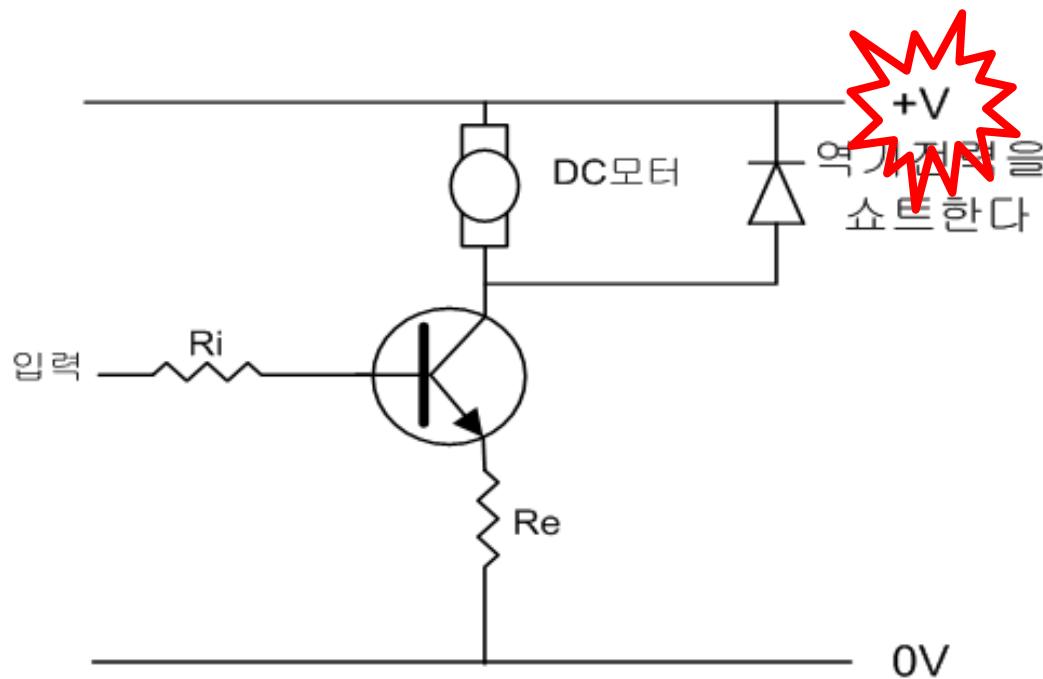
- 회전 방향을 바꾸는 H 브리지 제어회로
 - 단일전원으로 모터에 가하는 전압의 방향을 바꿀 수 있는 회로로 고안된 것이 "H 브리지 회로"
 - 모터는 정회전
 - Q1과 Q4의 트랜ジ스터만 동시에 On
 - 모터는 역회전
 - Q2와 Q3만 트랜ジ스터만 동시에 On



DC 모터 속도 제어 방법

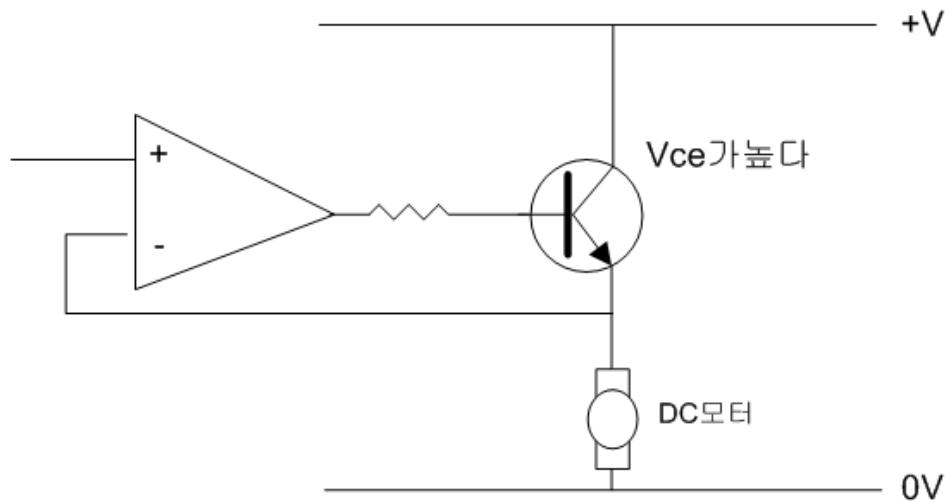
- DC 모터의 가변속 제어법

- DC 모터의 속도를 연속적으로 바꾸려는 경우에는 기본적으로는 DC 모터에 가하는 전압을 바꾸면 속도는 변함
- 단순히 모터의 코일에 흐르는 전류와 속도가 정비례하기 때문에, 모터의 구동 전압 +V를 변화시키면 속도 변함



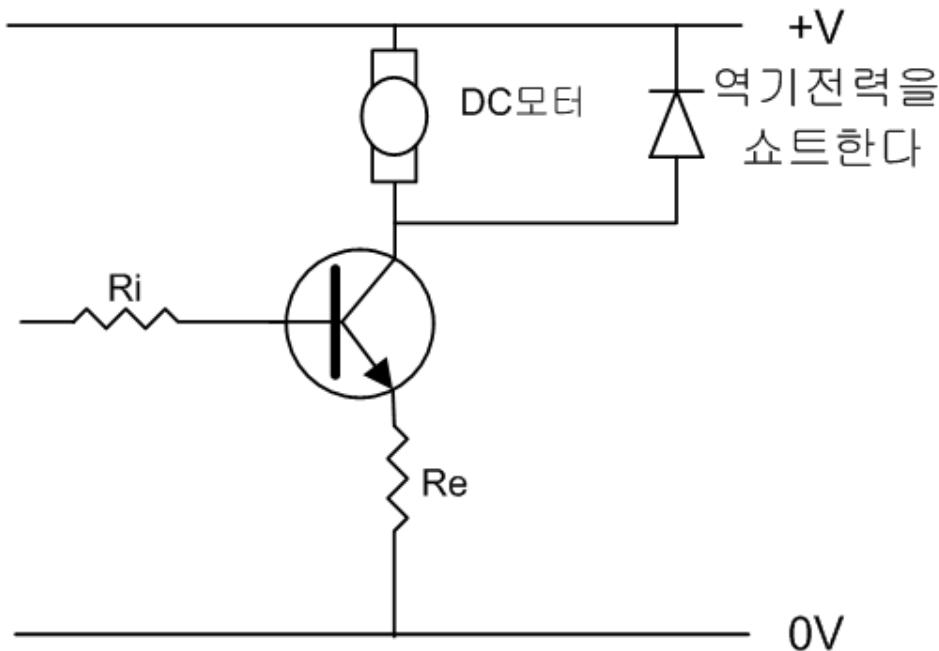
DC 모터 속도 제어 방법

- 구동 전압을 변화시키는 방법으로 아날로그 방식과 펄스폭 변조방식이 있음
 - 아날로그 방식의 가변속 제어
 - 직접 구동 전압을 변화시키는 것으로, 기본 회로는 그림과 같음
 - 트랜지스터로 전압 dropper를 구성하고, 컬렉터 이미터간의 드롭 전압을 바꿈으로써 모터에 가해지는 구동 전압을 가변



DC 모터 속도 제어 방법

- 펄스폭 변조(PWM : Pulse Width Modulation)
 - PWM 방식은 구동전압을 바꾸고 있는 것과 같은 효과를 내고 있지만, 그 방법이 펄스폭에 따르고 있음
 - 펄스의 duty비(On 시간과 Off 시간의 비)를 바꿈으로써 실현



DC 모터 사양과 제어 회로

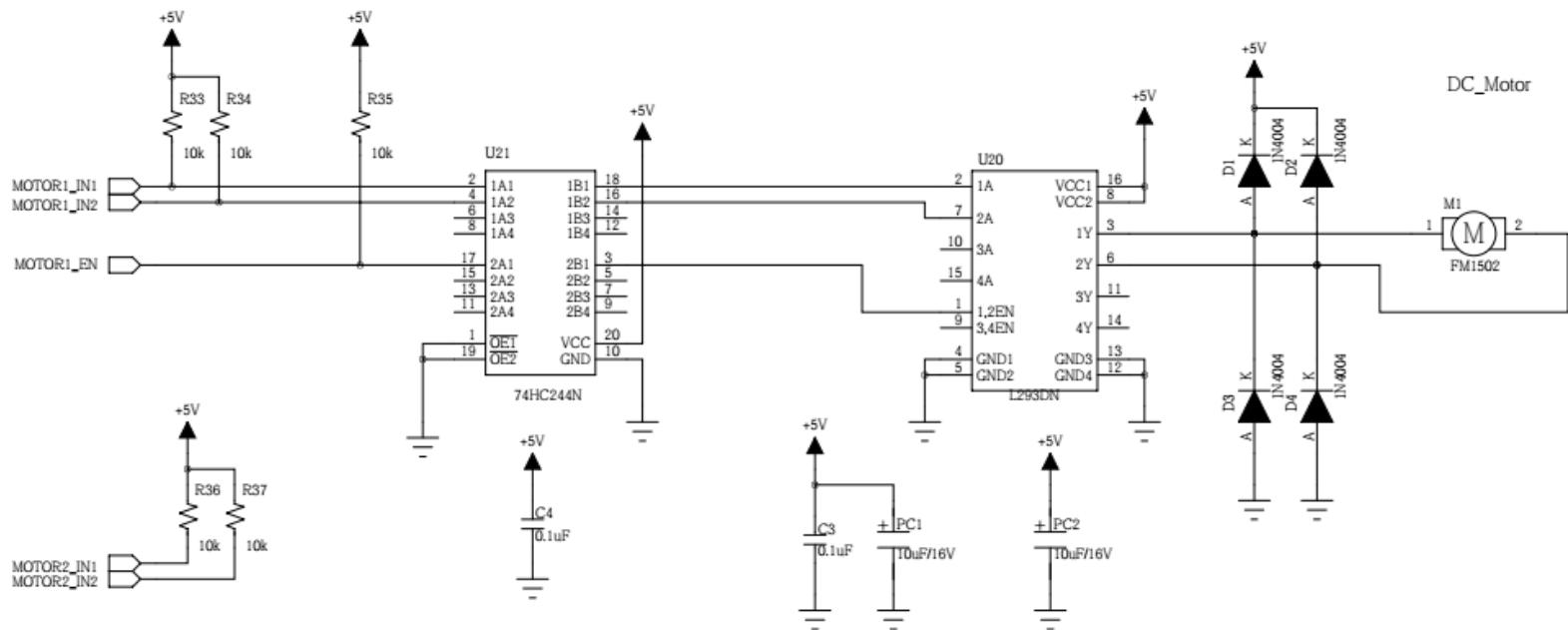
- 실습에 사용하는 모델은 FM1502
 - 정격 전압 : DC 5V
 - 정격 토크 : 3.54g·cm
 - 정격 회전수 : 8,670rpm
 - 정격 전류 : 90mA 이하
 - 무부하 회전수 : 10,860rpm
 - 무부하 전류 : 20mA 이하



DC 모터 사양과 제어 회로

- DC 모터 회로도

- L293DN – DC 모터 드라이버 IC, 2 채널
- 방향제어 - MOTOR1_IN1, MOTOR1_IN2,
MOTOR2_IN1(사용안함), MOTOR2_IN2(사용안함)
- 속도제어 - MOTOR1_EN, MOTOR2_EN(사용안함)



실습 1 : DC 모터 정회전, 역회전 시키기

- 실습 개요
 - DC Motor를 정회전 시켰다가 다시 역회전 시켜기
 - H-Bridge 회로가 내장되어 있는 L293DN 칩을 사용
- 실습 목표
 - STM32F405의 GPIO 포트에 대한 프로그램 방법 습득
 - L293DN을 사용한 DC Motor 제어의 원리 이해

실습 1 : DC 모터 정회전, 역회전 시키기

● DC 모터 제어 방법

- Forward(정회전)는 시계방향 회전이며 Reverse(역회전)은 반시계방향 회전
- Fast Motor Stop은 모터 양단에 같은 전압을 입력하여 급정지 시키는 것이며, Free Running Motor Stop은 모터 출력 enable 신호를 'disable' 시켜서 모터의 회전에 토크를 가하지 않아 서서히 정지하게되는 것

입력		기능
MOTOR1_EN = H	MOTOR1_IN1 = H MOTOR1_IN2 = L	Forward
	MOTOR1_IN1 = L MOTOR1_IN2 = H	Reverse
	MOTOR1_IN1 = MOTOR1_IN2	Fast Motor Stop
MOTOR1_EN = L	MOTOR1_IN1 = X MOTOR1_IN2 = X	Free Running Motor Stop

실습 1 : DC 모터 정회전, 역회전 시키기

- DC 모터 방향

- 정회전

- MOTOR1_IN1(PB8) : '1'(High)
MOTOR1_IN2(PB9) : '0'(Low)
MOTOR1_EN(PA3) : '1'(High)

```
GPIO_ResetBits(GPIOB, GPIO_Pin_8);
GPIO_SetBits(GPIOB, GPIO_Pin_9);
GPIO_SetBits(GPIOA, GPIO_Pin_3);
```

- 역회전

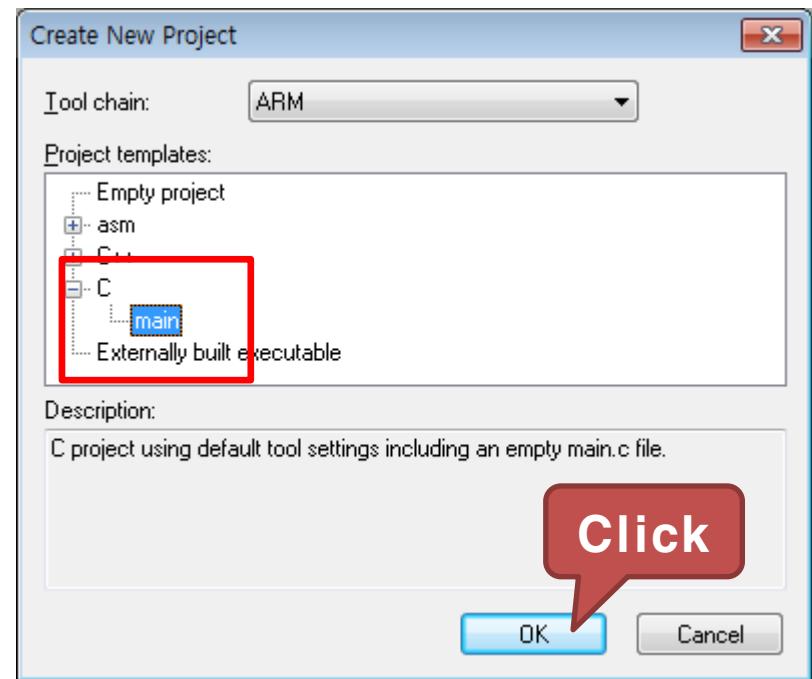
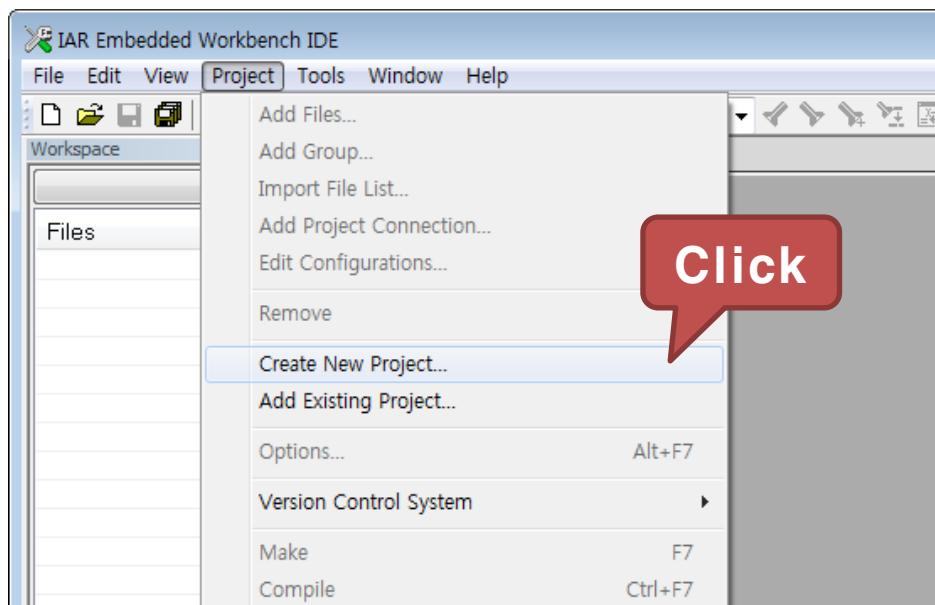
- MOTOR1_IN1(PB8) : '0'(Low)
MOTOR1_IN2(PB9) : '1'(High)
MOTOR1_EN(PA3) : '1'(High)

```
GPIO_SetBits(GPIOB, GPIO_Pin_8);
GPIO_ResetBits(GPIOB, GPIO_Pin_9);
GPIO_SetBits(GPIOA, GPIO_Pin_3);
```

실습 1 : DC 모터 정회전, 역회전 시키기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch11” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “dcMotorRotation”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1 : DC 모터 정회전, 역회전 시키기

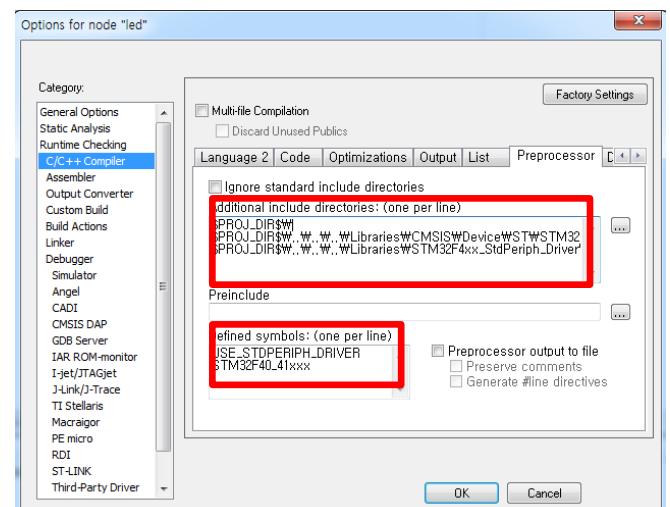
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch11\dcMotorRotation	system_stm32f4xx.c
Cortex_Example\Projects\ch11\dcMotorRotation	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 1 : DC 모터 정회전, 역회전 시키기

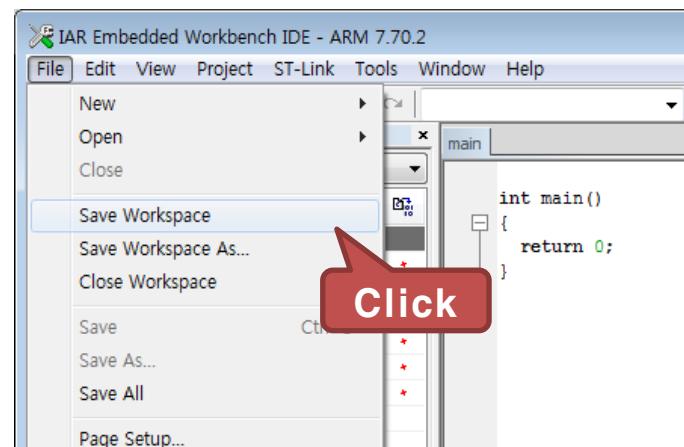
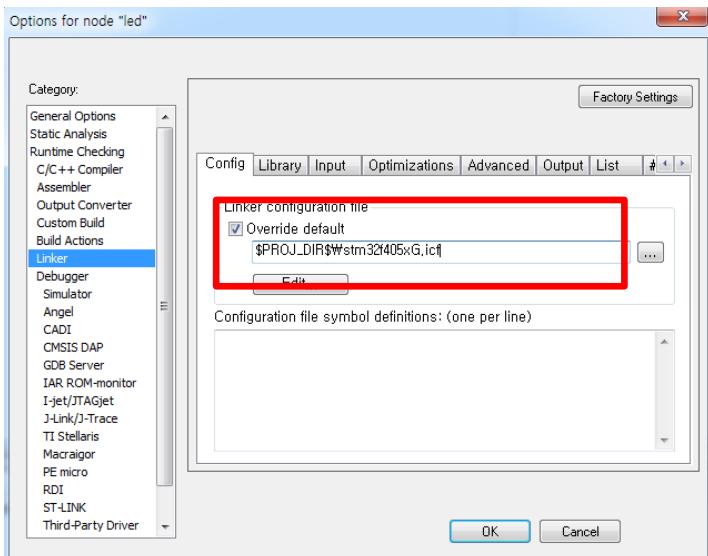
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1 : DC 모터 정회전, 역회전 시키기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1 : DC 모터 정회전, 역회전 시키기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE);
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;           // MOTOR1_EN
    GPIO_Init(GPIOA, &GPIO_InitStructure);
```

실습 1 : DC 모터 정회전, 역회전 시키기

- 구동 프로그램

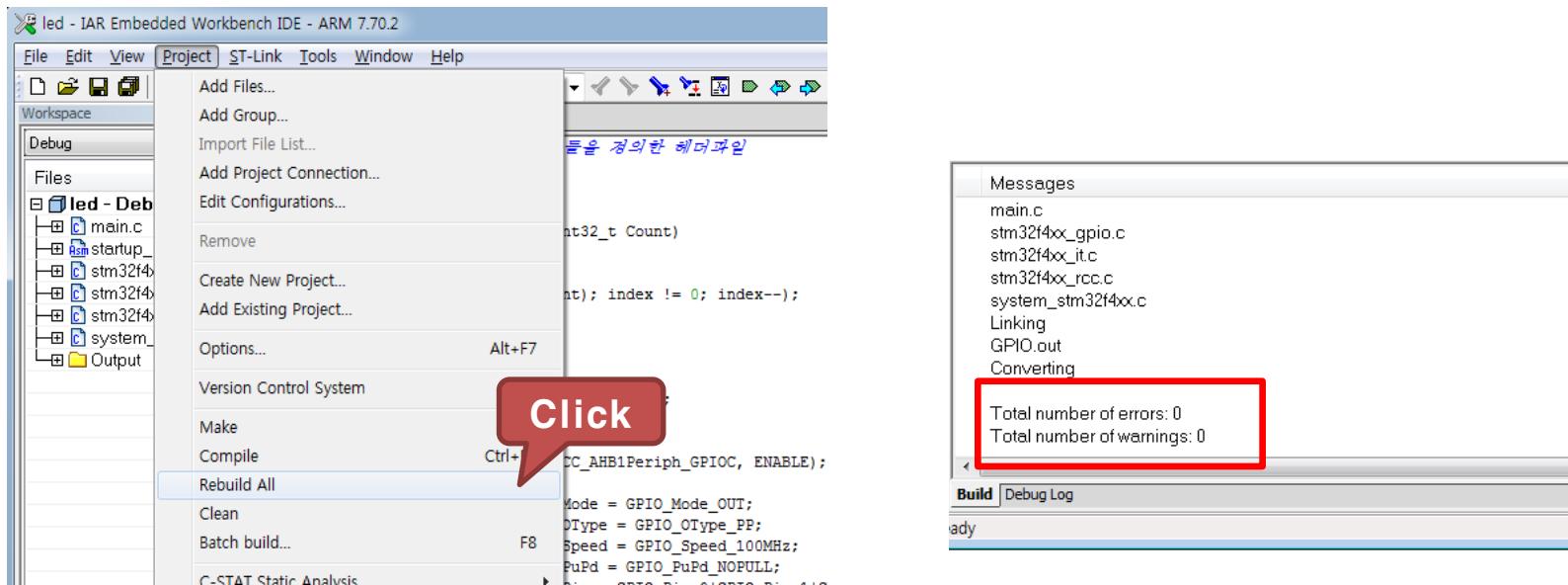
- main.c 코드 작성

```
// MOTOR1_IN1, MOTOR1_IN2
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;
GPIO_Init(GPIOB, &GPIO_InitStructure);
while(1) {
    // DC Motor 정회전
    GPIO_SetBits(GPIOB, GPIO_Pin_8);      // Motor 1
    GPIO_ResetBits(GPIOB, GPIO_Pin_9);     // Motor 2
    GPIO_SetBits(GPIOA, GPIO_Pin_3);       // Motor Enable
    Delay(1000);                         // 1초간 시간지연
    // DC Motor 역회전
    GPIO_ResetBits(GPIOB, GPIO_Pin_8);     // Motor 1
    GPIO_SetBits(GPIOB, GPIO_Pin_9);       // Motor 2
    GPIO_SetBits(GPIOA, GPIO_Pin_3);       // Motor Enable
    Delay(1000);                         // 1초간 시간지연
}
}
```

실습 1 : DC 모터 정회전, 역회전 시키기

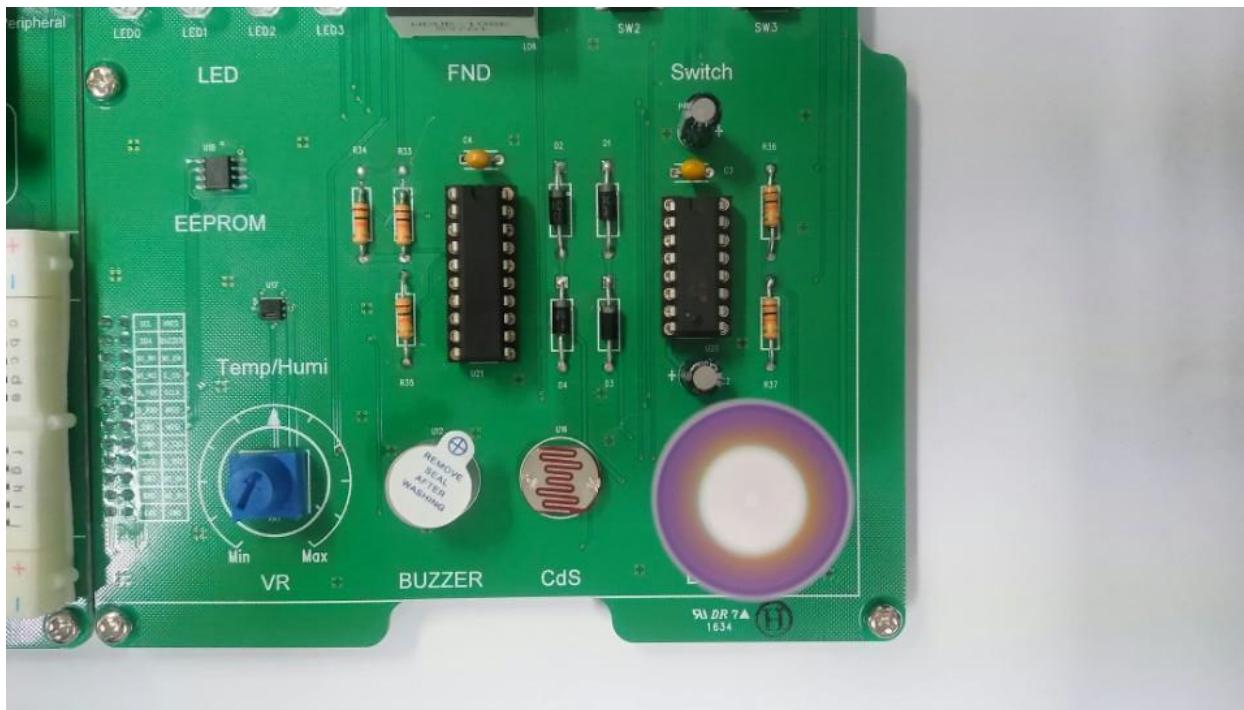
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 dcMotorRotation 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 1 : DC 모터 정회전, 역회전 시키기

- 실행 결과
 - 모터가 반복적으로 정회전, 역회전함



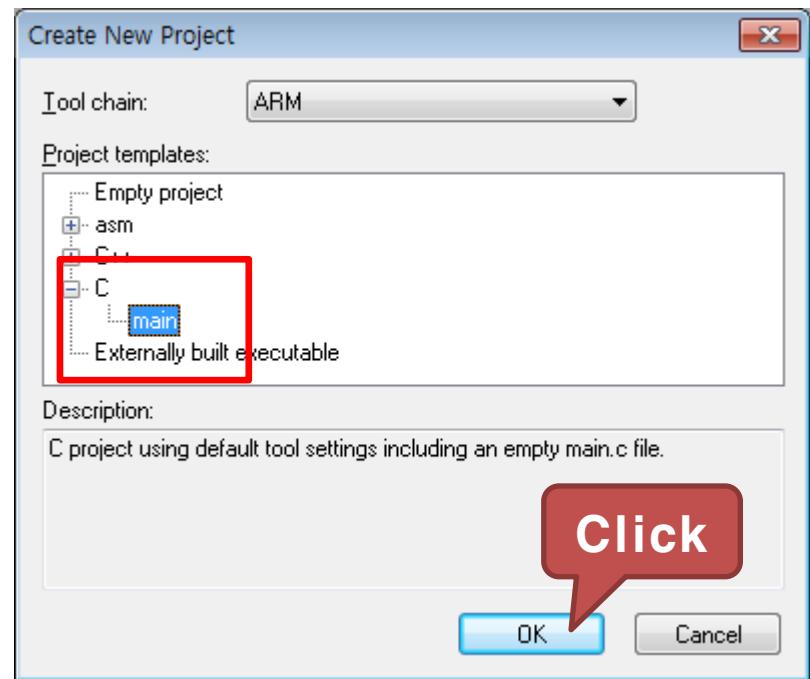
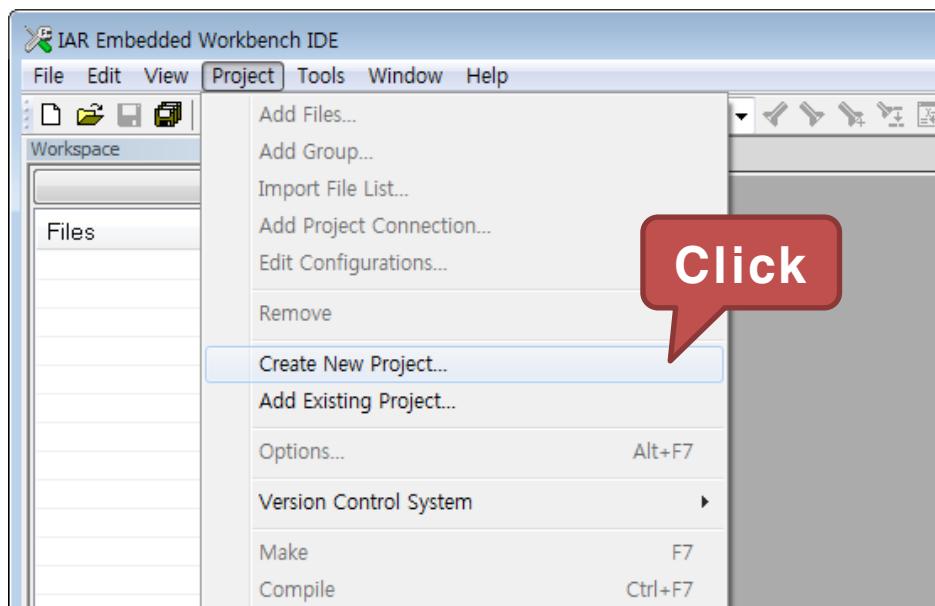
실습 2 : DC 모터 정지 시키기

- 실습 개요
 - DC 모터의 정지 방법인 Fast Motor Stop과 Free Running Motor Stop의 차이점을 알아보기
 - 스위치 입력을 통해 'SW0' 키가 눌리면 Fast Motor Stop을 'SW1' 키가 눌리면 Free Running Motor Stop을 'SW2' 키가 눌리면 다시 모터가 정회전시키기
- 실습 목표
 - STM32F405의 GPIO 포트에 대한 프로그램 방법 습득
 - L293DN을 사용한 DC Motor 제어의 원리 이해
 - Fast Motor Stop과 Free Running Motor Stop의 차이점 이해

실습 2 : DC 모터 정지 시키기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch11” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “dcMotorStop”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : DC 모터 정지 시키기

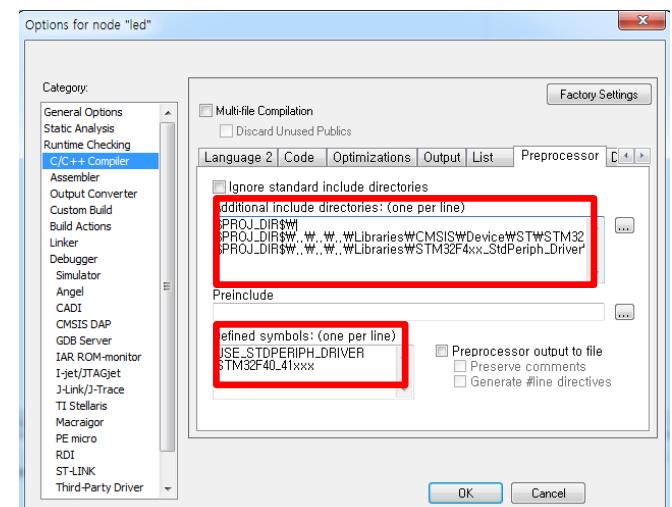
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch11\dcMotorStop	system_stm32f4xx.c
Cortex_Example\Projects\ch11\dcMotorStop	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 2 : DC 모터 정지 시키기

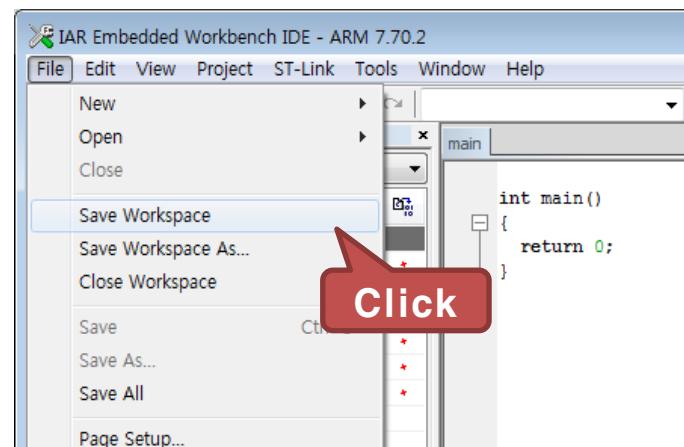
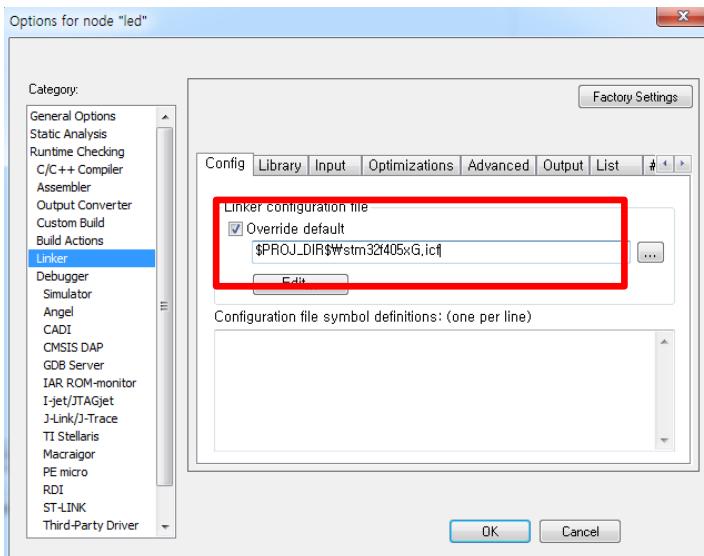
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : DC 모터 정지 시키기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : DC 모터 정지 시키기

- 구동 프로그램
 - main.c 코드 작성

```
#include "stm32f4xx.h"

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    unsigned int key = 0;
    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;      // MOTOR1_EN
    GPIO_Init(GPIOA, &GPIO_InitStructure);
```

실습 2 : DC 모터 정지 시키기

- 구동 프로그램
 - main.c 코드 작성

```
// MOTOR1_IN1, MOTOR1_IN2
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;
GPIO_Init(GPIOB, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
// B 포트 상위 4비트를 입력으로 선언
GPIO_InitStructure.GPIO_Pin =
    GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
GPIO_Init(GPIOB, &GPIO_InitStructure);

// DC Motor 정회전
GPIO_SetBits(GPIOB, GPIO_Pin_8);          // Motor 1
GPIO_ResetBits(GPIOB, GPIO_Pin_9);         // Motor 2
GPIO_SetBits(GPIOA, GPIO_Pin_3);           // Motor Enable
```

실습 2 : DC 모터 정지 시키기

- 구동 프로그램
 - main.c 코드 작성

```
while(1) {  
    key = GPIO_ReadInputData(GPIOB)&0xF000;  
    if(key == 0x1000) {          // '1' 키가 눌렸을때  
        // Motor1 정지, Fast Motor Stop  
        GPIO_SetBits(GPIOB, GPIO_Pin_8); // Motor 1  
        GPIO_SetBits(GPIOB, GPIO_Pin_9); // Motor 2  
    }  
    else if(key == 0x2000) {      // '2' 키가 눌렸을때  
        // Motor1 정지, Free Running Motor Stop  
        GPIO_ResetBits(GPIOA, GPIO_Pin_3); // Motor Enable  
    }  
}
```

실습 2 : DC 모터 정지 시키기

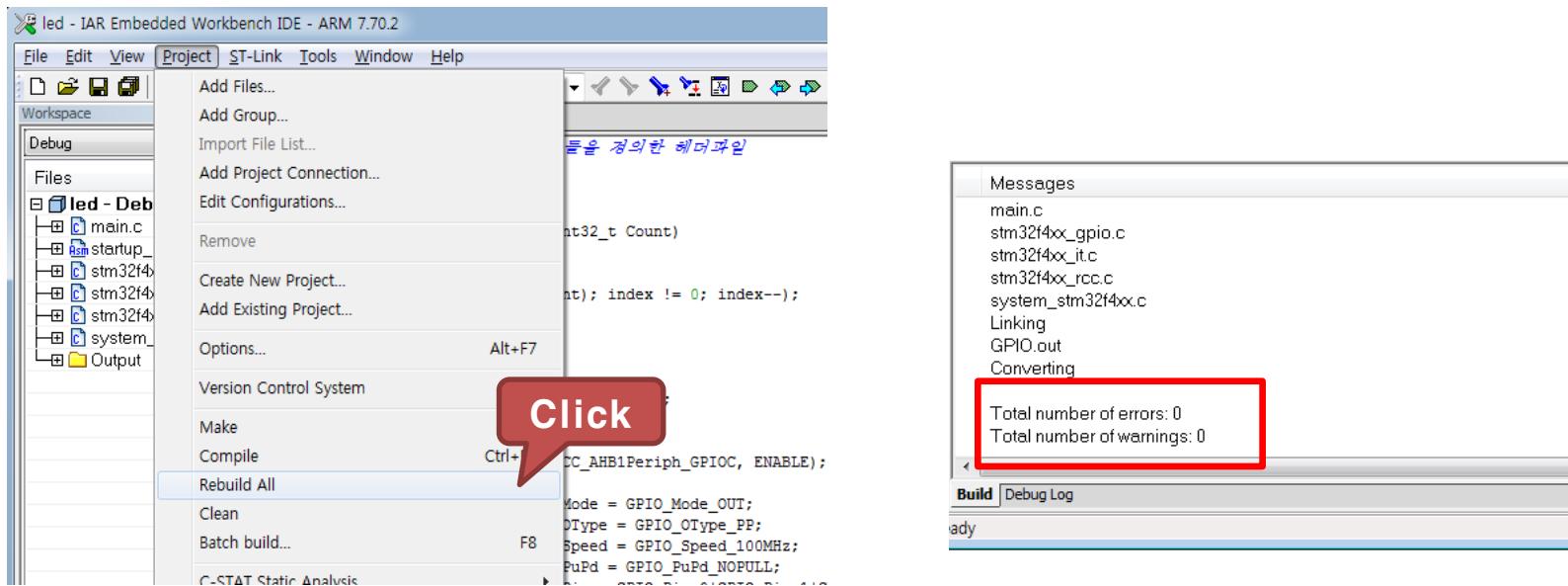
- 구동 프로그램
 - main.c 코드 작성

```
while(1) {  
    //...  
    else if(key == 0x4000) {      // '3' 키가 눌렸을때  
        // DC Motor 정회전  
        GPIO_SetBits(GPIOB, GPIO_Pin_8);    // Motor 1  
        GPIO_ResetBits(GPIOB, GPIO_Pin_9); // Motor2  
        GPIO_SetBits(GPIOA, GPIO_Pin_3);    // Motor Enable  
    }  
}
```

실습 2 : DC 모터 정지 시키기

● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 dcMotorStop 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 2 : DC 모터 정지 시키기

● 실행 결과

- 동작 버튼인 'SW2'를 눌러 모터를 회전 확인
- 정지 버튼인 'SW0'과 'SW1'을 각각 눌러 정지 확인
- 모터가 빠르게 정지할 때와 천천히 정지하는 것을 확인
- 모터를 바로 정지하는 Fast Motor stop은 강제로 정지시키는 것이므로 순간 전류가 많이 필요하므로 유의해서 사용



실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

- 실습 개요
 - 타이머2의 PWM 기능을 이용하여 DC 모터를 제어
 - PWM 모드를 이용하여 20KHz(70% 듀티비)의 PWM신호를 발생시켜 DC 모터를 회전시키기
- 실습 목표
 - STM32F405의 PWM 핀에 대한 프로그램 방법 습득
 - L293DN을 사용한 DC Motor 제어의 원리 이해
 - 타이머2의 PWM 기능 동작원리 이해

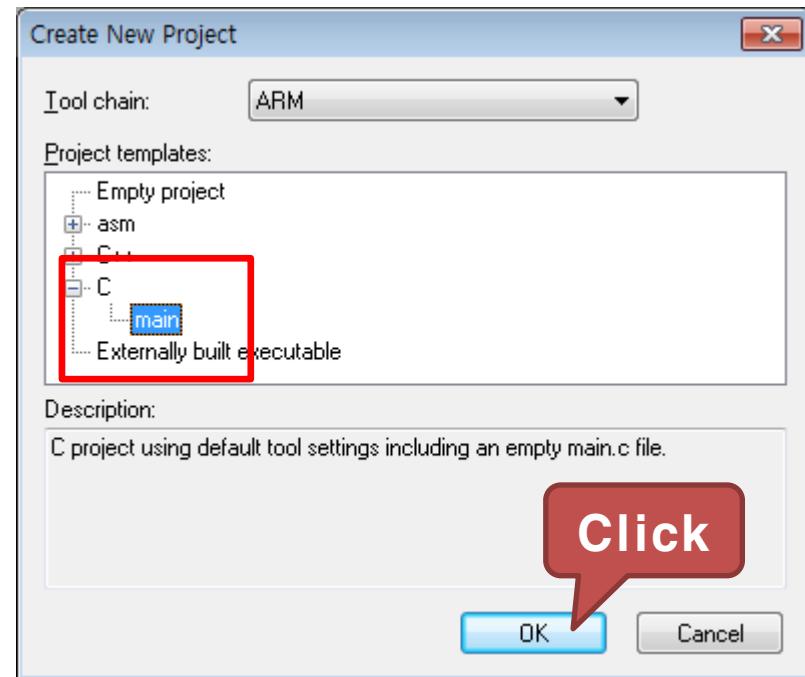
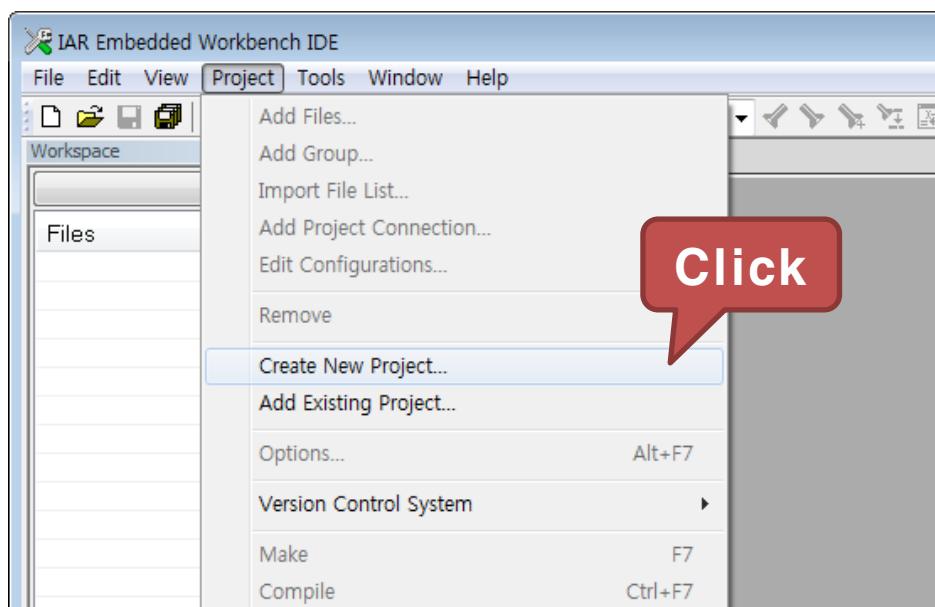
실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

- 구동 프로그램 : PWM 동작 모드 설정
 - 사용할 타이머/카운터 결정
 - 여기서는 타이머/카운터 2를 사용
 - 동작 모드 결정
 - 여기서는 PWM 모드 사용
 - 카운트 방향을 업카운트(0 부터 TIMx-ARR까지)로 선택
 - TIM2_CR1레지스터의 DIR을 0으로 클리어
 - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
 - 여기서는 클럭(84MHz)을 사용
 - 타이머 2의 클럭은 메인 클럭(168Mhz)의 $\frac{1}{2}$ 을 사용
 - 샘플링 클럭의 분주비는 1:1로 설정
 - 타이머 클럭 프리스케일러는 41로 설정(TIM2_CR1/CKD:00, TIM2_PSC: 41)
 - 84MHz 클럭의 1분주비와 41 프리스케일러로 나누어 지므로 2MHz(0.5us)로 동작
 - PWM 주파수 결정
 - 20KHz를 만들기 위해 0.05ms을 타이머 주기로 결정
 - TIM2_ARR레지스터의 값을 100으로 설정
 - 듀티비는 TIM2_CCR4 에 70을 입력하여 듀티비를 70% 로 설정

실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch11” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “dcMotorPwm1”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

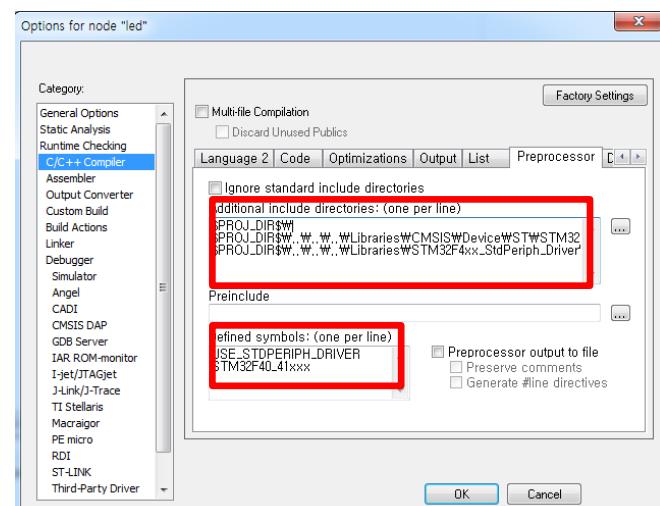
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch11\dcMotorPwm1	system_stm32f4xx.c
Cortex_Example\Projects\ch11\dcMotorPwm1	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c

실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

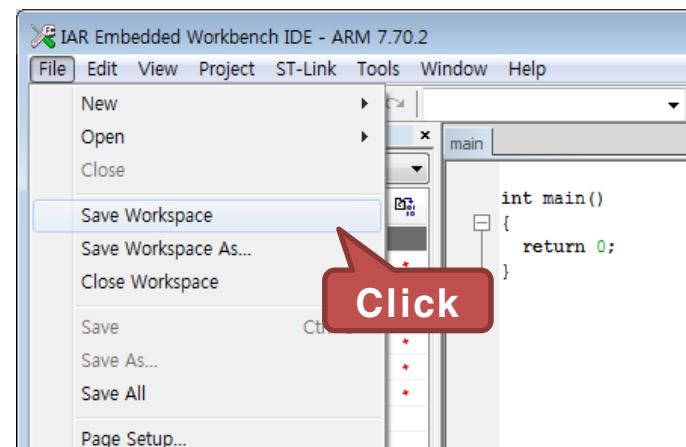
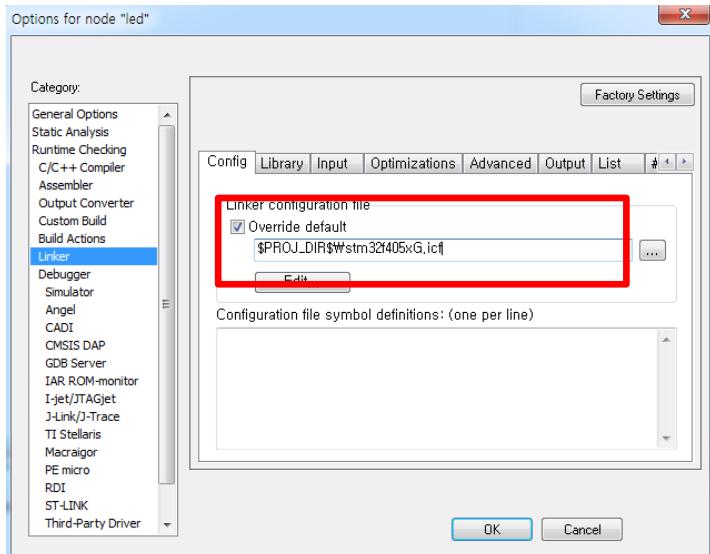
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCInitTypeDef  TIM_OCInitStructure;
    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
```

실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

- 구동 프로그램

- main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;  
// MOTOR1_IN1, MOTOR1_IN2  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;  
GPIO_Init(GPIOB, &GPIO_InitStructure);  
  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3; // MOTOR1_EN(TIM2_CH4)  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
  
GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_TIM2);  
// (168Mhz/2)/42 = 2MHz(0.5us)  
TIM_TimeBaseStructure.TIM_Prescaler = 42-1;  
// (0.5us x 100(99+1) = 50us(20KHz))  
TIM_TimeBaseStructure.TIM_Period = 100-1;
```

실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

- 구동 프로그램

- main.c 코드 작성

```
//...  
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;  
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);  
  
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;  
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;  
TIM_OCInitStructure.TIM_Pulse = 70;  
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;  
TIM_OC4Init(TIM2, &TIM_OCInitStructure);  
  
// 타이머2를 동작  
TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Enable);  
TIM_ARRPreloadConfig(TIM2, ENABLE);  
TIM_Cmd(TIM2, ENABLE);
```

실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

- 구동 프로그램
 - main.c 코드 작성

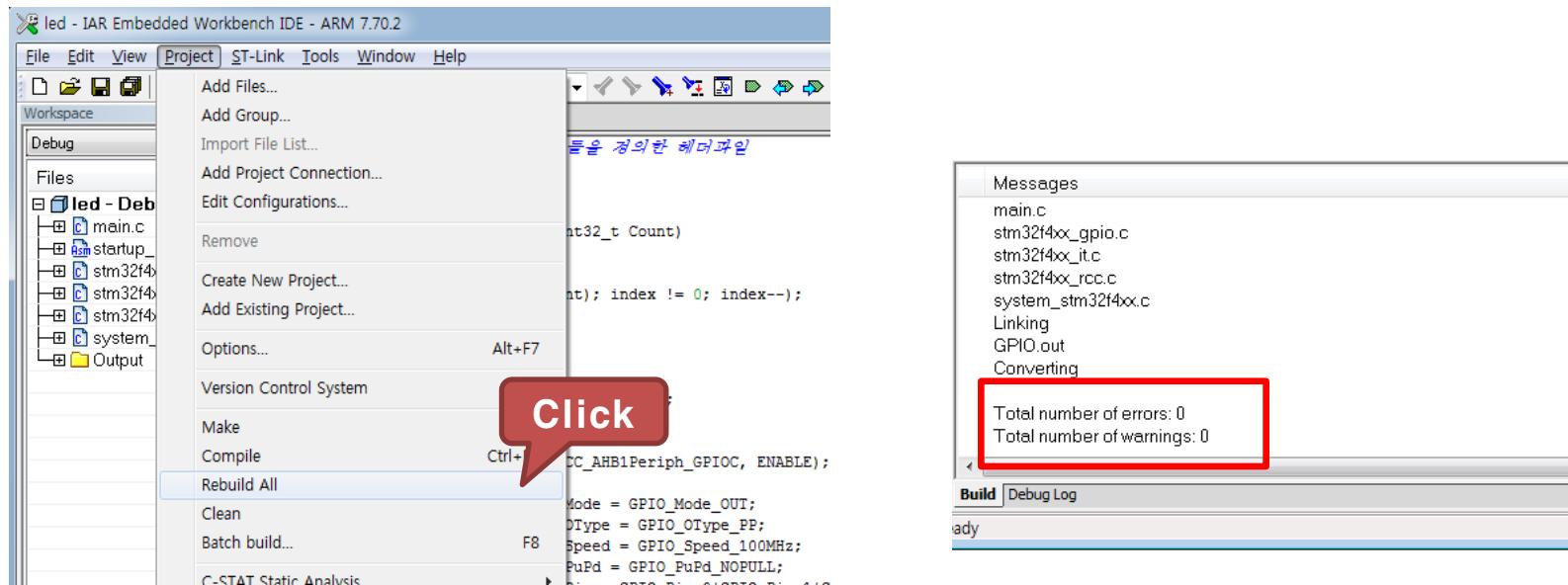
```
//...
while(1) {
    // DC Motor 정회전
    GPIO_SetBits(GPIOB, GPIO_Pin_8);      // Motor 1
    GPIO_ResetBits(GPIOB, GPIO_Pin_9);     // Motor 2
    Delay(1000);                         // 1초간 시간지연

    // DC Motor 역회전
    GPIO_ResetBits(GPIOB, GPIO_Pin_8);     // Motor 1
    GPIO_SetBits(GPIOB, GPIO_Pin_9);       // Motor 2
    Delay(1000);                         // 1초간 시간지연
}
}
```

실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

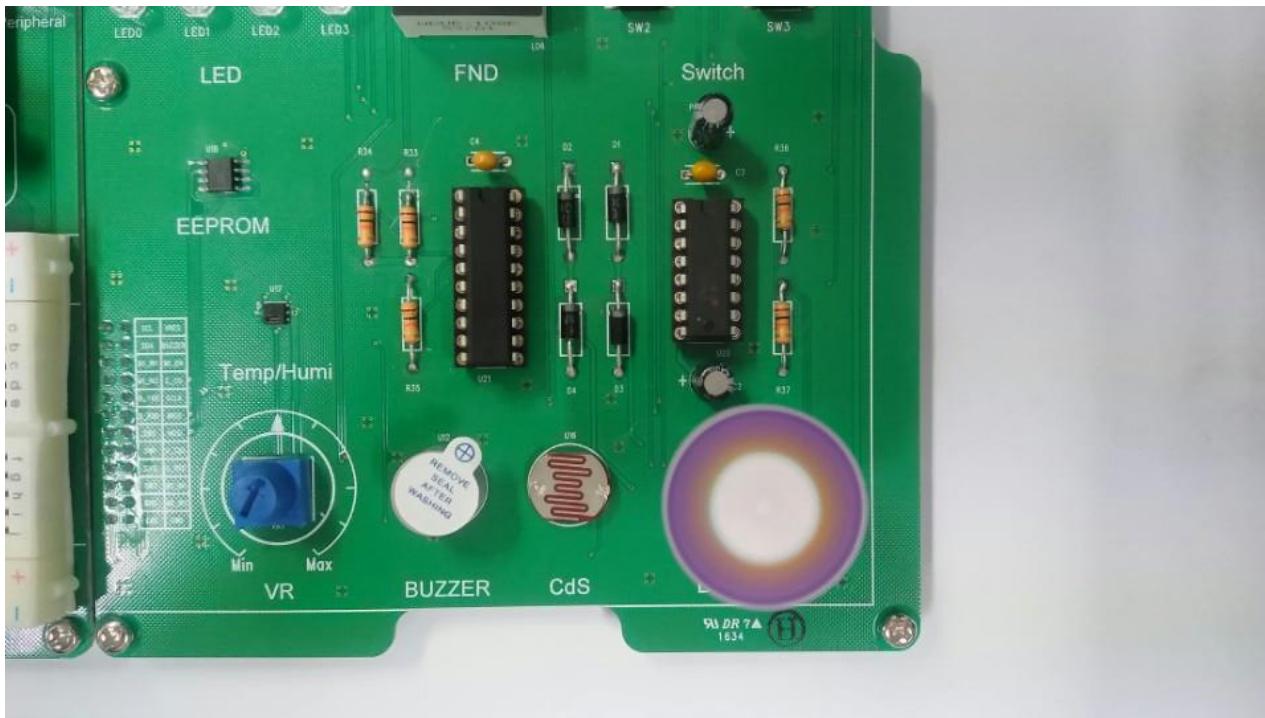
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 dcMotorPwm1 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 3 : PWM을 이용하여 DC 모터 속도 제어하기 1

- 실행 결과
 - 모터가 반복해서 정회전, 역회전함
 - 실습1과 비교해서 회전속도가 느림



실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 실습 개요
 - 타이머2의 PWM 기능을 이용하여 DC 모터를 제어
 - PWM 모드를 이용하여 20KHz의 PWM신호를 발생시켜 DC 모터를 회전 시키기
 - 시간에 따라 PWM 듀티비를 증가시키고 감소시켜서 DC 모터의 속도를 변화를 관찰
 - 변화하는 듀티비는 Text LCD에 출력
- 실습 목표
 - STM32F405의 PWM 핀에 대한 프로그램 방법 습득
 - L293DN을 사용한 DC Motor 제어의 원리 이해
 - 타이머2의 PWM 기능 동작원리 이해
 - Text LCD의 사용법 이해

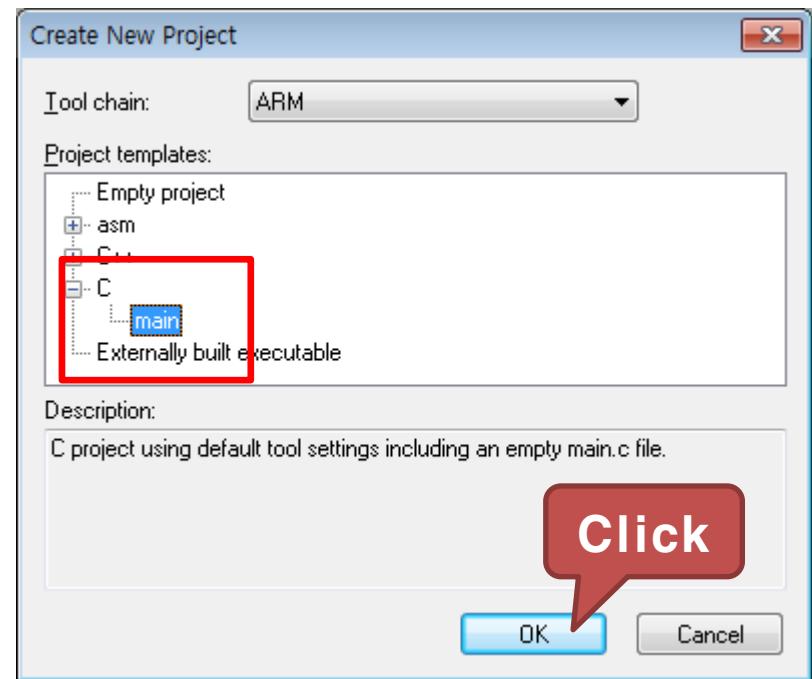
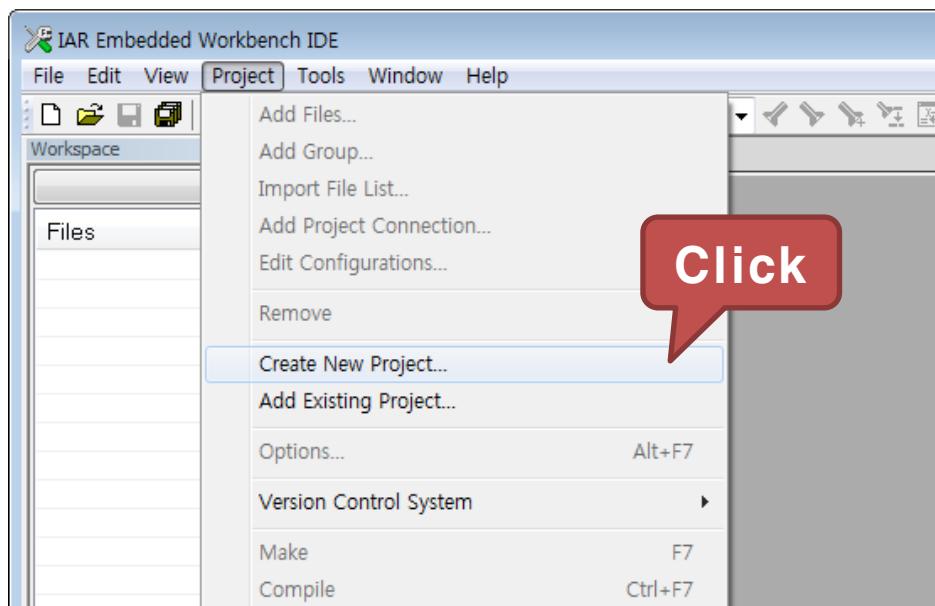
실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 구동 프로그램 : PWM 동작 모드 설정
 - 사용할 타이머/카운터 결정
 - 여기서는 타이머/카운터 2를 사용
 - 동작 모드 결정
 - 여기서는 PWM 모드 사용
 - 카운트 방향을 업카운트(0 부터 TIMx-ARR까지)로 선택
 - TIM2_CR1레지스터의 DIR을 0으로 클리어
 - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
 - 여기서는 클럭(84MHz)을 사용
 - 타이머 2의 클럭은 메인 클럭(168Mhz)의 $\frac{1}{2}$ 을 사용
 - 샘플링 클럭의 분주비는 1:1로 설정
 - 타이머 클럭 프리스케일러는 41로 설정(TIM2_CR1/CKD:00, TIM2_PSC: 41)
 - 84MHz 클럭의 1분주비와 41 프리스케일러로 나누어 지므로 2MHz(0.5us)로 동작
 - PWM 주파수 결정
 - 20KHz를 만들기 위해 0.05ms을 타이머 주기로 결정
 - TIM2_ARR레지스터의 값을 100으로 설정
 - 듀티비는 TIM2_CCR4 에 70을 입력하여 듀티비를 70% 로 설정

실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

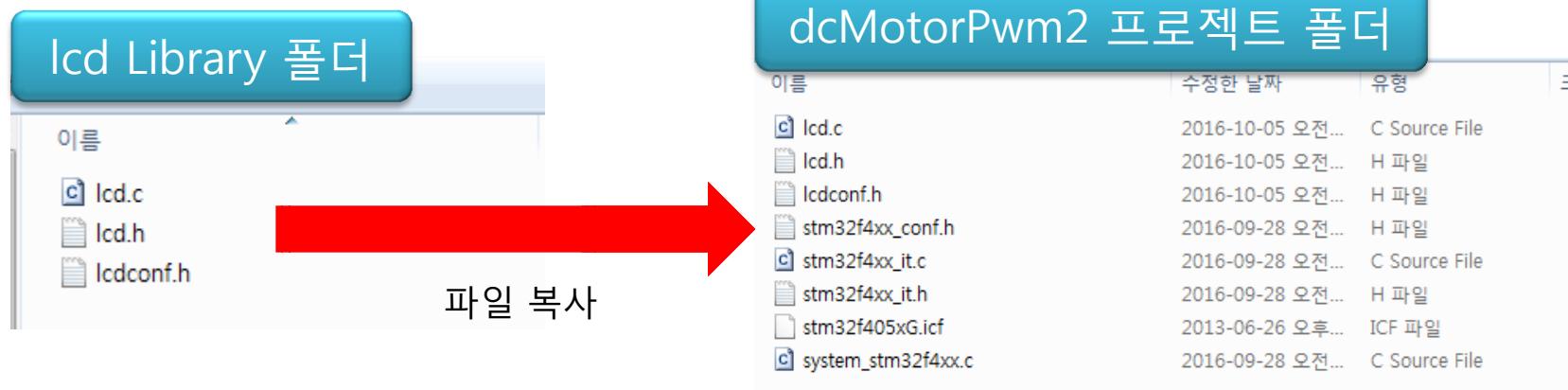
● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch11” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “dcMotorPwm2”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 구동 프로그램 : EWARM에서 라이브러리 함수 추가하는 방법
 - 라이브러리 함수파일을 프로젝트내로 복사
 - lcd.c, lcd.h, lcdconf.h 파일은 “Cortex Example\Projects\library\lcd” 폴더에 존재
 - 라이브러리 파일은 EWARM에서 새 프로젝트를 생성한 후 생성된 폴더에 복사



실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

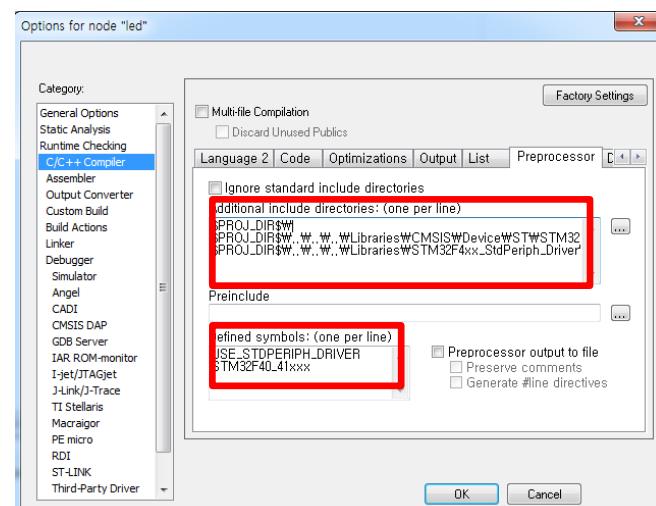
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch11\dcMotorPwm2	system_stm32f4xx.c
Cortex_Example\Projects\ch11\dcMotorPwm2	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c

실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

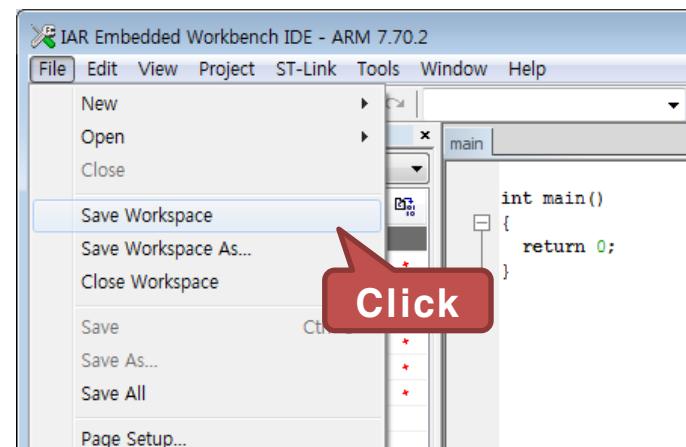
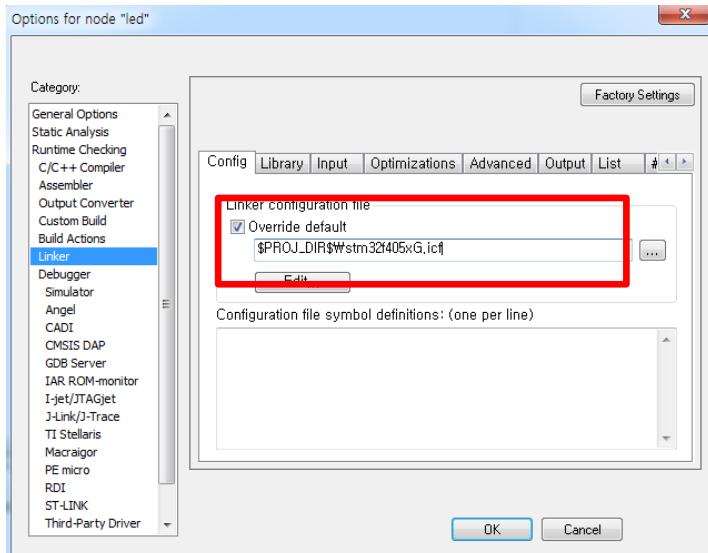
● 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"
#include "lcd.h"          // Text LCD를 사용하기 위한 헤더 파일

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCInitTypeDef  TIM_OCInitStructure;
    unsigned int pwmduty = 0, cnt_dir = 0;
    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
```

실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 구동 프로그램

- main.c 코드 작성

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
// MOTOR1_IN1, MOTOR1_IN2
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;
GPIO_Init(GPIOB, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3; // MOTOR1_EN(TIM2_CH4)
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_TIM2);
// (168Mhz/2)/42 = 2MHz(0.5us)
TIM_TimeBaseStructure.TIM_Prescaler = 42-1;
// (0.5us x 100(99+1) = 50us(20KHz))
TIM_TimeBaseStructure.TIM_Period = 100-1;
```

실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 구동 프로그램

- main.c 코드 작성

```
//...
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 70;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OC4Init(TIM2, &TIM_OCInitStructure);
// 타이머2를 동작
TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Enable);
TIM_ARRPreloadConfig(TIM2, ENABLE);
TIM_Cmd(TIM2, ENABLE);
```

실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 구동 프로그램

- main.c 코드 작성

```
//...
// DC Motor 정회전
GPIO_SetBits(GPIOB, GPIO_Pin_8);          // Motor 1
GPIO_ResetBits(GPIOB, GPIO_Pin_9);         // Motor 2

lcdInit();                      // Text LCD를 초기화
lcdGotoXY(0,0);                // 커서위치를 첫번째 줄 첫번째 칸으로 이동
lcdPrintData(" Duty: ",7);      // " Duty: " 출력

while(1) {
    if(cnt_dir) {
        pwmduty--;           // pwmduty가 1씩 감소
        // pwmduty가 0이 되면 pwmduty가 증가하기 시작
        if(pwmduty == 0) cnt_dir = 0;
    }
}
```

실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 구동 프로그램

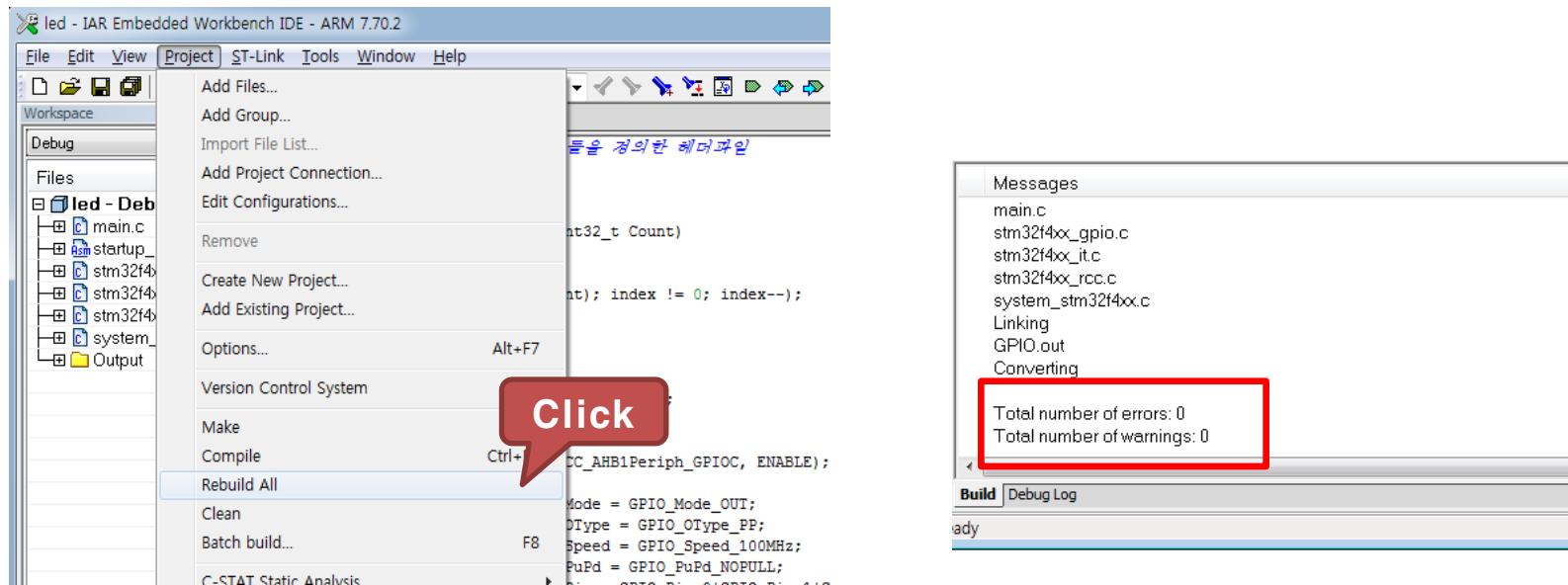
- main.c 코드 작성

```
//...
else {
    pwmduty++;          // pwmduty가 1씩 증가
    // pwmduty가 100이 되면 pwmduty가 감소하기 시작
    if(pwmduty == 99) cnt_dir = 1;
}
TIM_Cmd(TIM2, DISABLE);
TIM_SetCompare4(TIM2,pwmduty);          // 듀티비가 1%씩 변화
TIM_Cmd(TIM2, ENABLE);
lcdGotoXY(7,0);
lcdDataWrite((pwmduty/10)%10 + '0'); // 10의 자리 출력
lcdDataWrite((pwmduty)%10 + '0');    // 1의 자리 출력
lcdDataWrite('%');                  // % 출력
Delay(1000);
}
```

실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

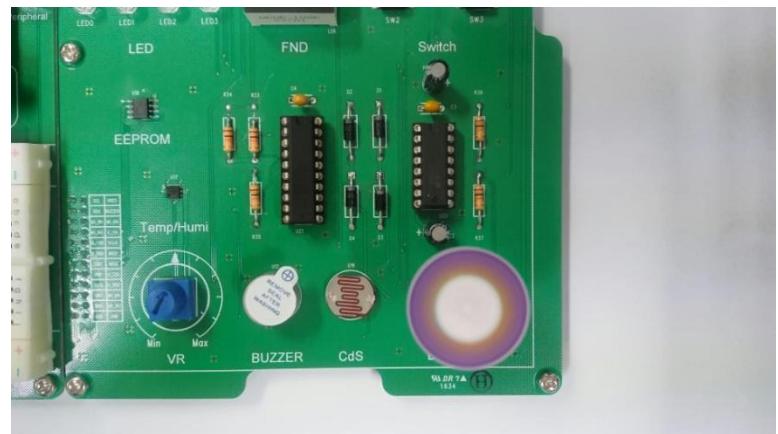
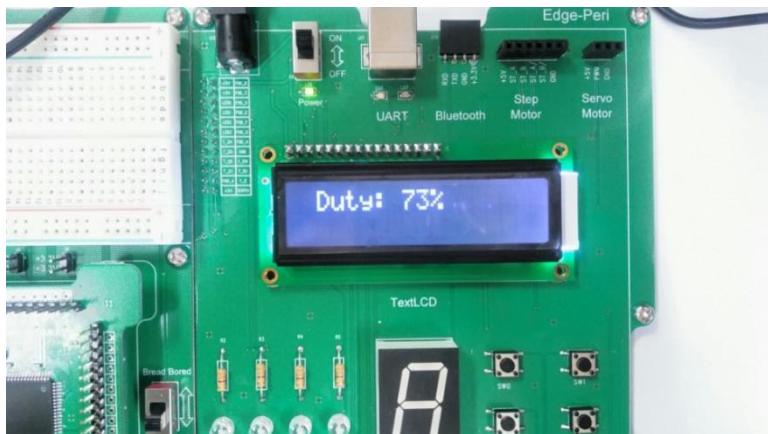
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 dcMotorPwm2 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 4 : PWM을 이용하여 DC 모터 속도 제어하기 2

- 실행 결과
 - 특정 듀티비 이상이면 모터가 회전하며, 이하이면 모터 정지





STEP MOTOR

- 스텝 모터(Step Motor)
- 1상 여자 방식으로 스텝 모터 돌리기
- 2상 여자 방식으로 스텝 모터 돌리기
- 1-2상 여자 방식으로 스텝 모터 돌리기 1
- 1-2상 여자 방식으로 스텝 모터 돌리기 2



엣지아이랩

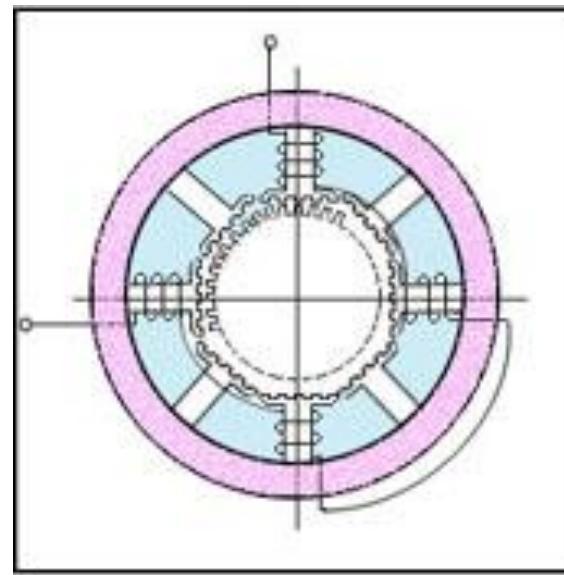
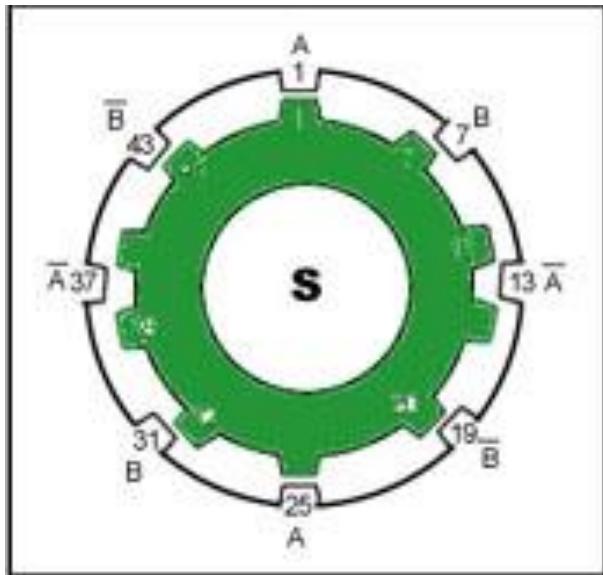
스텝 모터(Step Motor)

- 스텝 모터(스텝모터, 펄스 모터)

- 1902년 영국에서 처음 개발되어, 1960년대 일본에서 NC공작기계에 처음 도입
- AC 서보, DC 서보 모터에 비하여 값이 싸고 정확한 각도제어에 유리
- 기계적인 이동량을 정밀하게 제어하는 곳에 스텝 모터가 많이 사용
- 펄스에 의해 디지털적으로 제어하는 것이 가능하므로 마이크로컨트롤러에서 사용하기에 적합한 모터
- 샤프트(축)의 위치를 검출하기 위한 별도의 피드백 신호없이 정해진 각도를 회전하고 (Open Loop Control), 상당히 높은 정확도로 정지할 수 있음
- 다른 모터에 비해 정지시 매우 큰 유지 토크가 있기 때문에 전자 브레이크 등의 위치 유지 기구를 필요로 하지 않음
- 회전 속도가 펄스 속도에 비례하므로 간편하게 제어 할 수 있음

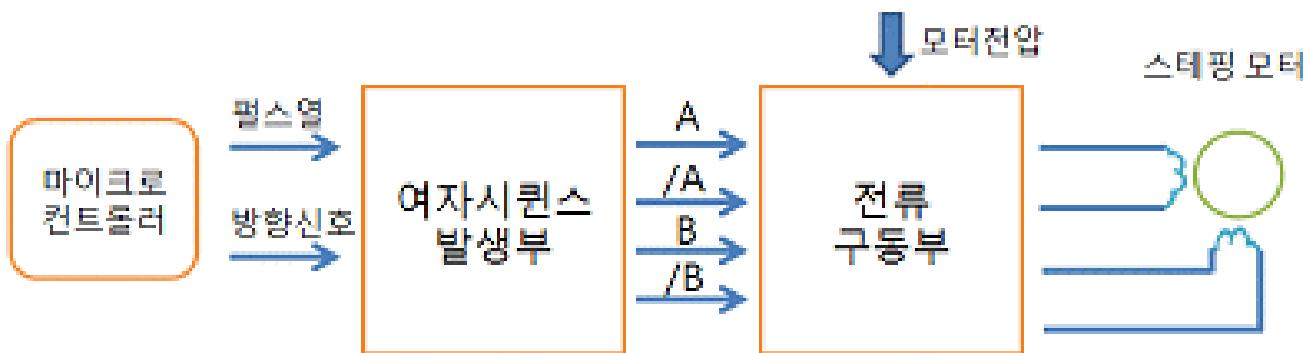
스텝 모터(Step Motor)

- 스텝 모터의 구조



스텝 모터(Step Motor)

- 스텝 모터 구동방법
 - 스텝 모터 제어 회로
 - 여자 신호 발생부 : 펄스를 인가하여 각상의 여자 신호를 발생함
 - 구동 회로부 : 신호를 받아서 권선에 여자 전류를 흘려줌
 - 마이크로 컨트롤러
 - 방향 신호와 펄스열을 발생



스텝 모터(Step Motor)

- 1상 여자 방식 (Full step)

- 구동 방법

- 스텝 모터를 구동하기 위한 최소한의 구동 방법
 - $\text{STEP_A} \rightarrow \text{STEP_A} \rightarrow \text{STEP_B} \rightarrow \text{STEP_B} \rightarrow \text{STEP_A} \Rightarrow$ 정회전

- 특징

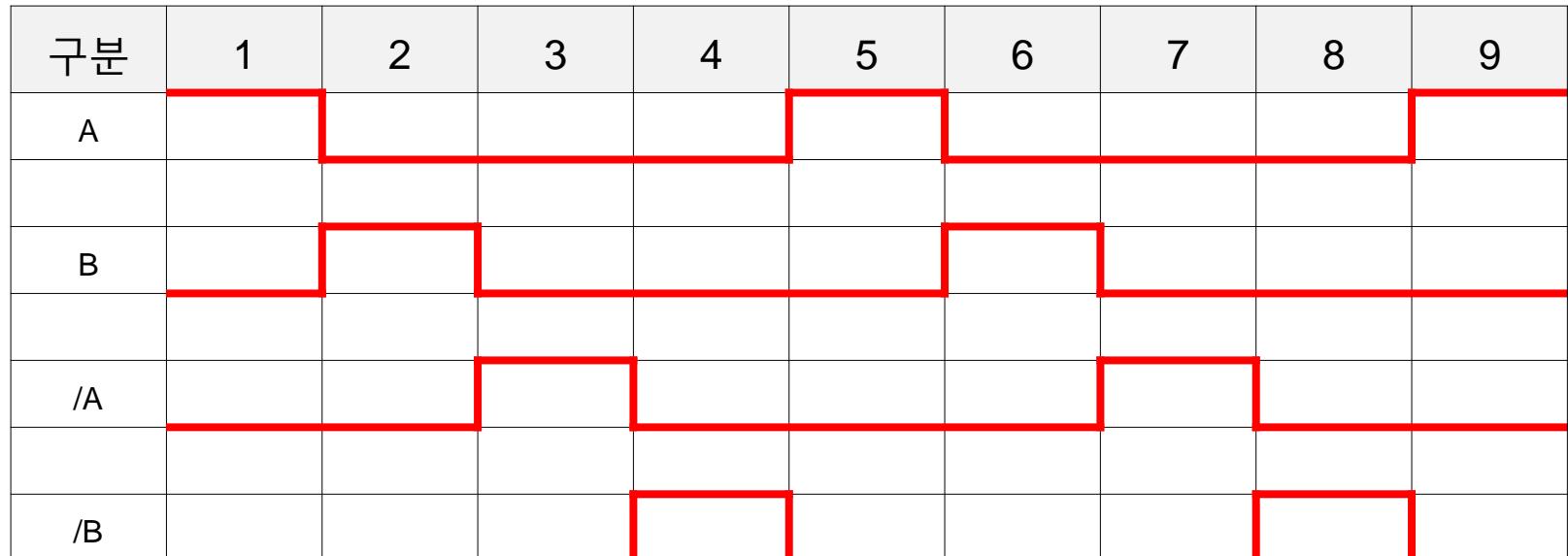
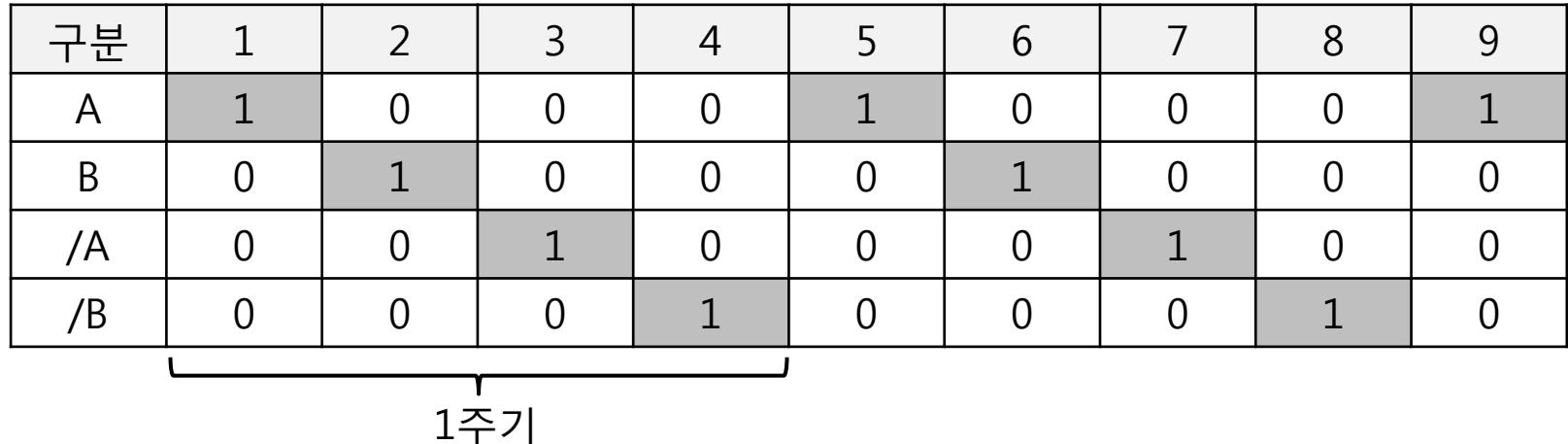
- 1개의 코일 만을 차례로 여자 하는 방식
 - 소비 전력이 낮고 1스텝당 각 정밀도가 높음
 - 감쇠 진동이 크고 탈조 하기 쉬움

구분	1	2	3	4	5	6	7	8	9
A	1	0	0	0	1	0	0	0	1
B	0	1	0	0	0	1	0	0	0
/A	0	0	1	0	0	0	1	0	0
/B	0	0	0	1	0	0	0	1	0

1주기

스텝 모터(Step Motor)

- 1상 여자 방식 (Full step)
 - 1상 여자 방식 동작 타이밍



스텝 모터(Step Motor)

- 2상 여자 방식

- 구동 방법

- 항상 2상이 여자 되므로 기동 토크가 주어져 난조가 일어나기 어렵고 항상 2개의 상에서 전류가 흐르게 하도록 해야 함
 - STEP_A,STEP/_A → STEP/_A, STEP_B → STEP_B, STEP/_B → STEP/_B,STEP_A
⇒ 정회전

- 특징

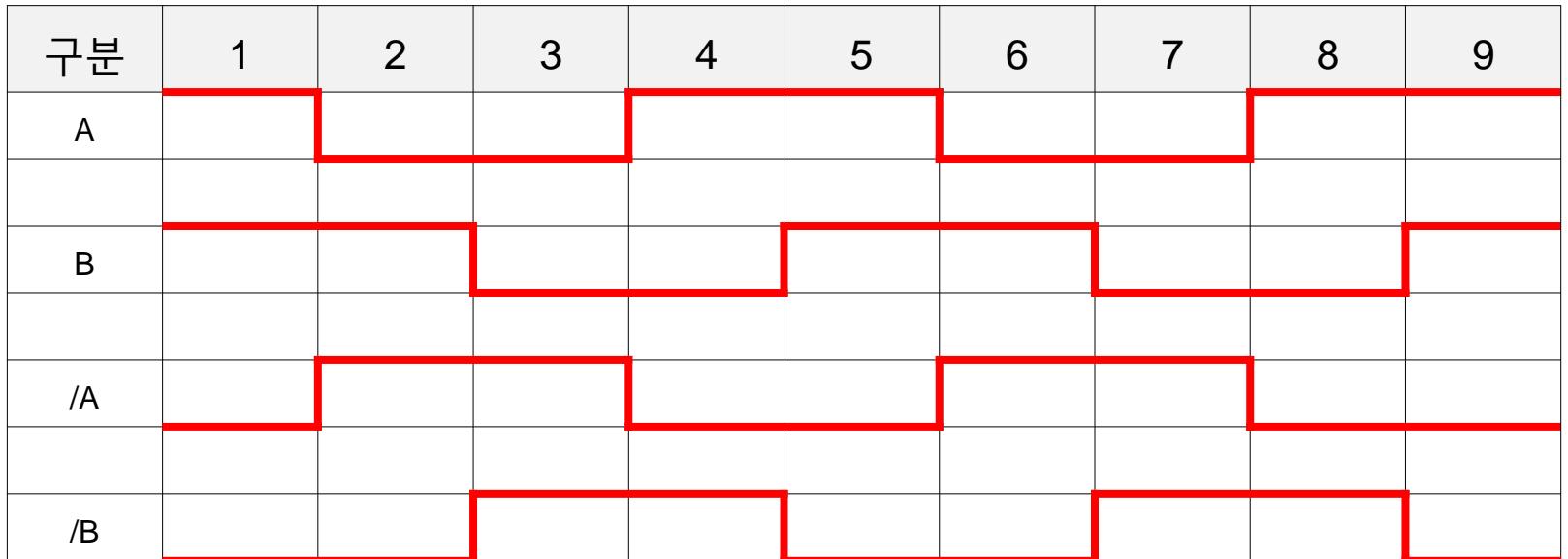
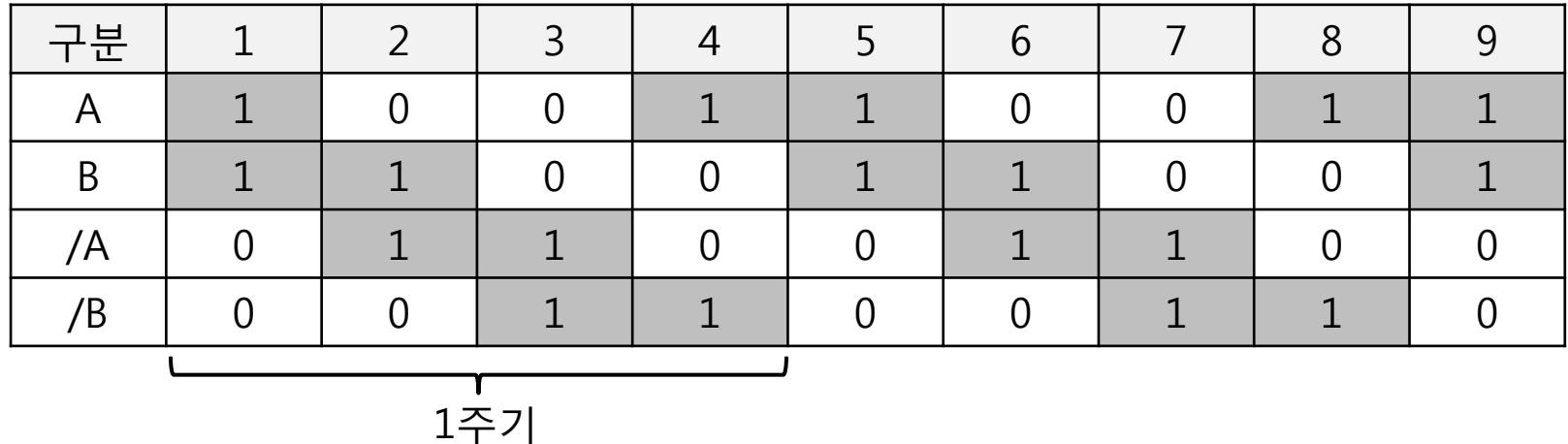
- 2개의 코일을 동시에 여자 하는 방식
 - 1상 여자 구동에 비해 2배의 전류가 필요하지만 토크가 크고 감쇠진동이 적음
 - 주파수(damping)특성이 양호하여 가장 널리 이용되는 방법

구분	1	2	3	4	5	6	7	8	9
A	1	0	0	1	1	0	0	1	1
B	1	1	0	0	1	1	0	0	1
/A	0	1	1	0	0	1	1	0	0
/B	0	0	1	1	0	0	1	1	0

1주기

스텝 모터(Step Motor)

- 2상 여자 방식
 - 2상 여자 방식 동작 타이밍



스텝 모터(Step Motor)

- 1-2상 여자 방식

- 구동 방법

- 하나의 상과 두개의 상에 교대로 전류를 흐르게 하는 방식
 - 스텝각은 1,2상 여자 방식의 1/2이며 응답 스텝 비율은 1,2 상의 2배

- 특징

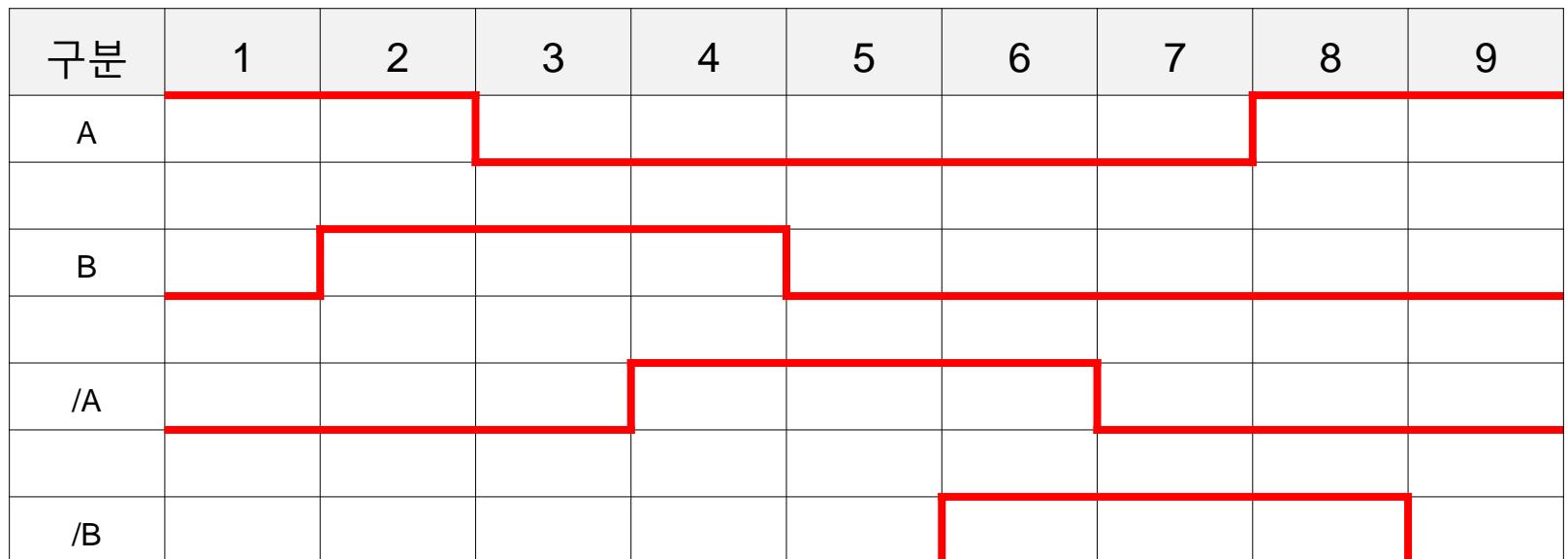
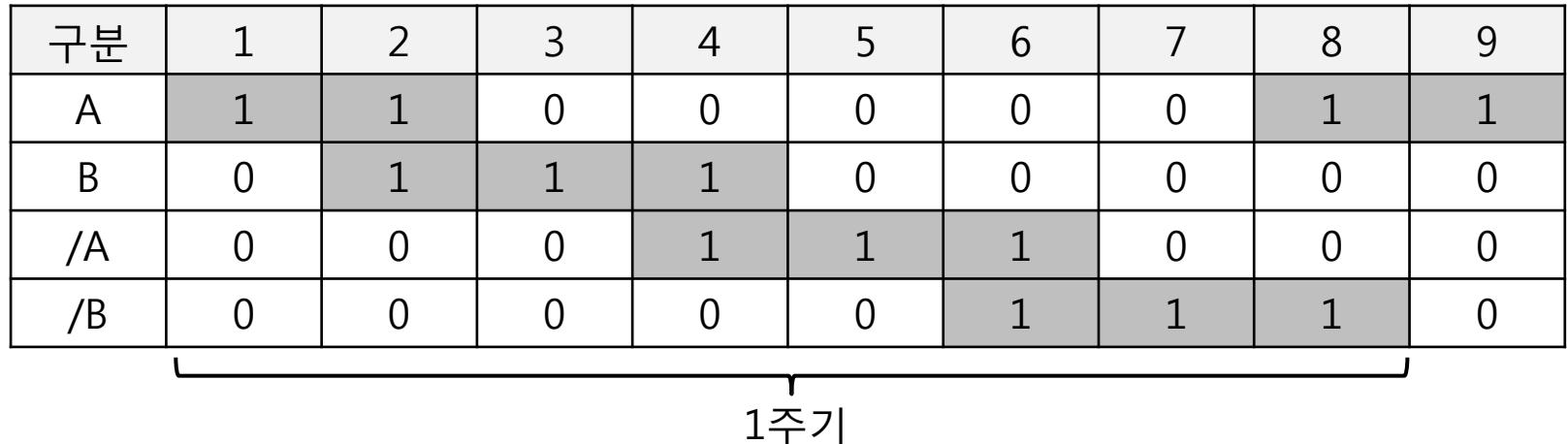
- 1상 여자 구동에 비해 1.5배의 전류가 필요
 - 1펄스에 대한 스텝 각은 1상 여자와 2상 여자에 의한 스텝 각의 1/2
 - 각도를 정밀하게 제어하는 경우에 사용

구분	1	2	3	4	5	6	7	8	9
A	1	1	0	0	0	0	0	1	1
B	0	1	1	1	0	0	0	0	0
/A	0	0	0	1	1	1	0	0	0
/B	0	0	0	0	0	1	1	1	0

1주기

스텝 모터(Step Motor)

- 1-2상 여자 방식
 - 1-2상 여자 방식 동작 타이밍



실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

● 실습 개요

- STM32F405의 GPIO핀에 스텝 모터의 제어 신호를 연결하여, 스텝 모터를 회전 시키기
- 1상 여자 방식을 사용하여 반 시계방향으로 무한 회전
- 본 실험에서 사용하고 있는 Step 모터는 1상, 2상 여자 방식일 때 펄스당 7.5도, 1-2상 여자 방식일 때 펄스당 3.75도 회전

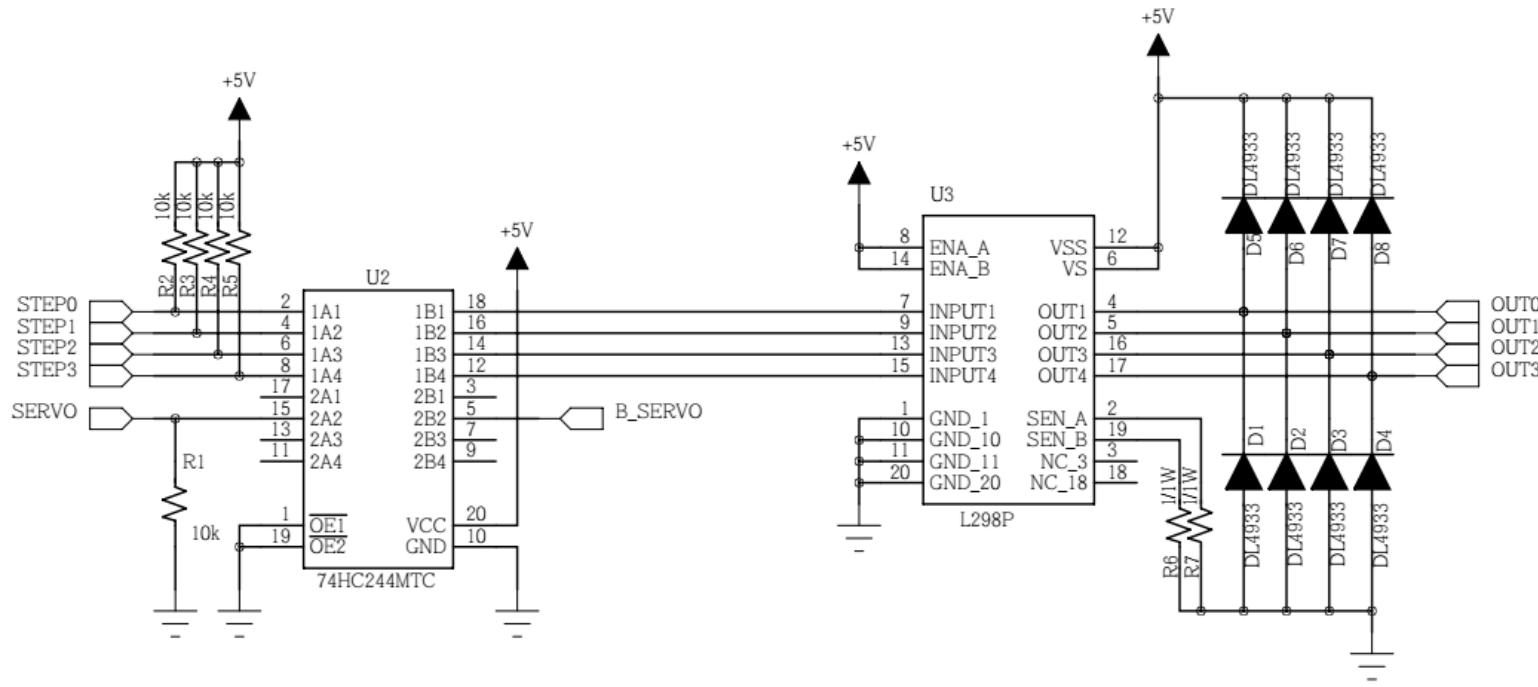
● 실습 목표

- 스텝 모터의 동작원리를 이해
- STM32F405의 GPIO를 이용한 스텝 모터 구동 방법 습득

실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

- 사용 모듈

- 스텝 모터 모듈의 회로
 - 제어 신호
 - OUT_0 → STEP_A, OUT_1 → STEP_B
 - OUT_2 → STEP_A, OUT_3 → STEP_B



실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

- 구동 프로그램 : 사전지식
 - 스텝 모터의 1상 여자 방식 신호 만들기
 - B(9:8)에 차례대로, 0x0100, 0x0200, 0x0000, 0x0000, 동시에 A(12:11)에 차례대로, 0x0000, 0x0000, 0x0800, 0x1000 을 한 주기로 하여 계속 보냄, 반대방향으로 돌리고 싶다면, 역순서로 데이터를 보내면 됨
- 1상 여자 방식 제어 신호(반 시계 방향)

신호	1	2	3	4	5	6	7	8	9
PB8(A)	1	0	0	0	1	0	0	0	1
PB9(B)	0	1	0	0	0	1	0	0	0
PA11(A)	0	0	1	0	0	0	1	0	0
PA12(B)	0	0	0	1	0	0	0	1	0
B 포트	0x0100	0x0200	0x0000	0x0000	0x0100	0x0200	0x0000	0x0000	0x0100
A 포트	0x0000	0x0000	0x0800	0x1000	0x0000	0x0000	0x0800	0x1000	0x0000

실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

- 구동 프로그램 : 사전지식

- GPIO 사용

- OUT_0(STEP_A) → PORTB(PB8)
 - OUT_1(STEP_B) → PORTB(PB9)
 - OUT_2(STEP_A) → PORTA(PA11)
 - OUT_3(STEP_B) → PORTA(PA12)

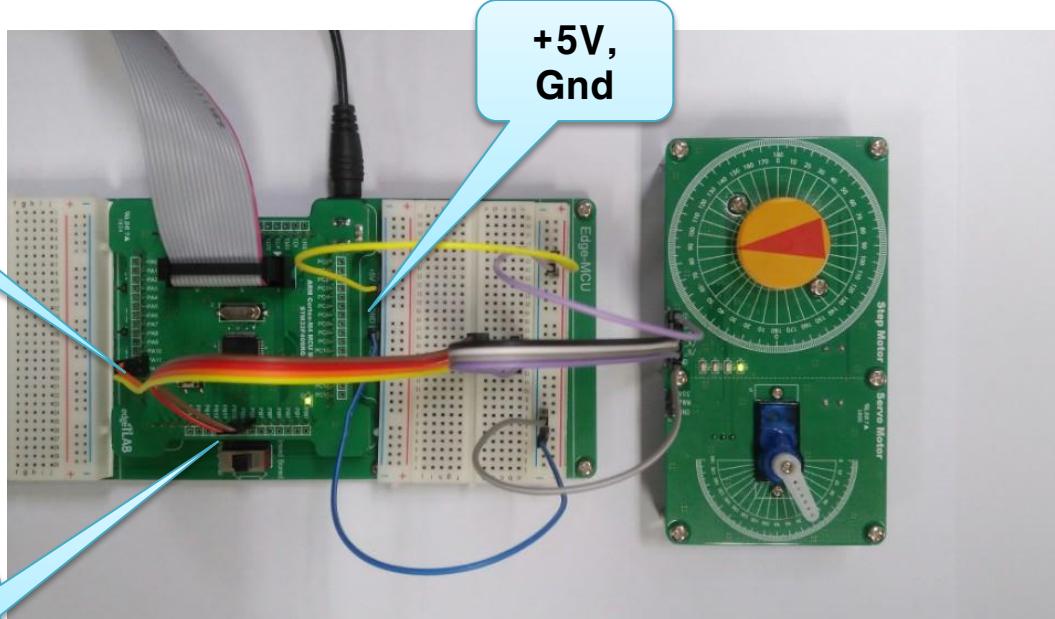
- 시간 지연 함수 사용

- 1상 여자 방식 신호 만들기를 위해서는 B(9:8)/A(12:11)에 데이터를 보내는 시간 간격은 일정하게 유지
 - 시간 지연 함수를 이용하여 10ms 마다 상을 변화시킴

실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

● 실습 준비

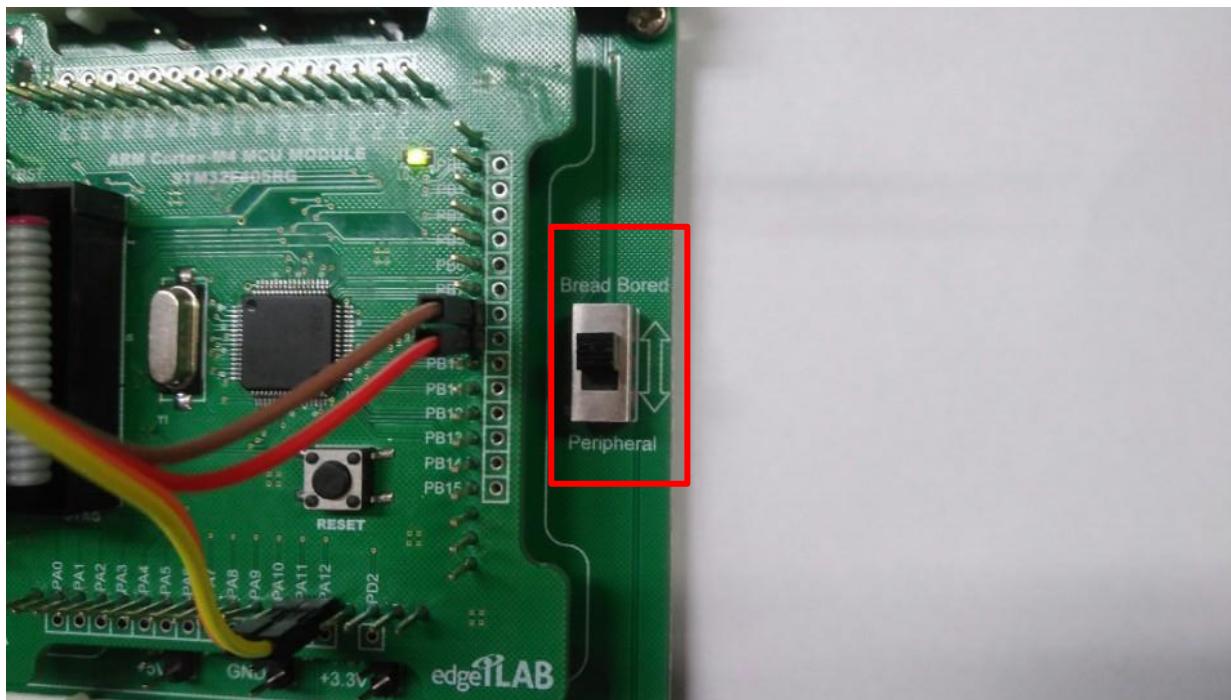
- 사용보드를 다음 그림과 같이 연결
 - Edge-MCU보드의 **+5V** → Bread보드 (+) → Step 모터 보드의 **VCC**
 - Edge-MCU보드의 **PB8** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_A**
 - Edge-MCU보드의 **PB9** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_B**
 - Edge-MCU보드의 **PA11** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_A**
 - Edge-MCU보드의 **PA12** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_B**
 - Edge-MCU보드의 **GND** → Bread보드 (-) → Edge-Peri 보드의 **GND**



실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

- 실습 준비

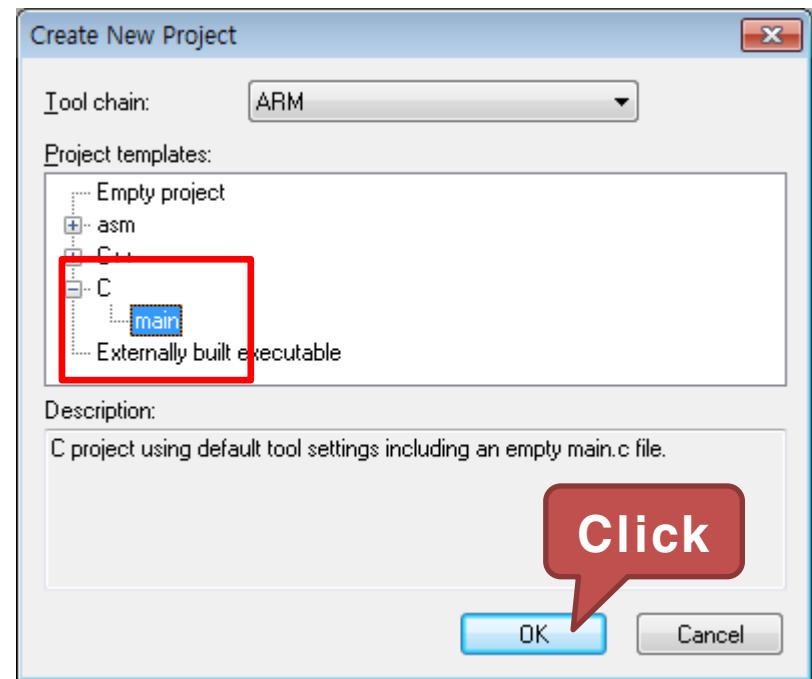
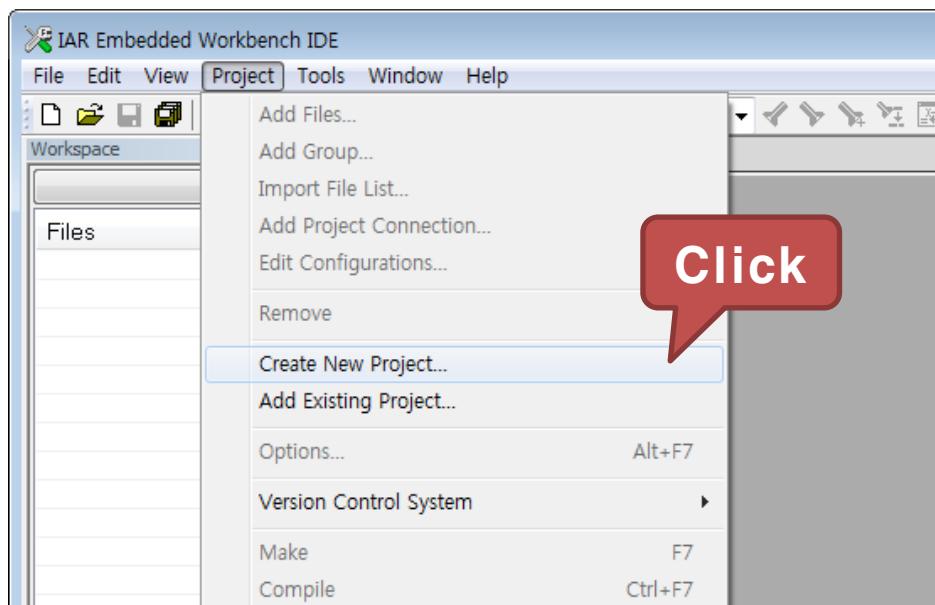
- Edge-MCU보드 오른쪽의 스위치를 Bread Board로 올림



실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch12” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “step1Phase”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

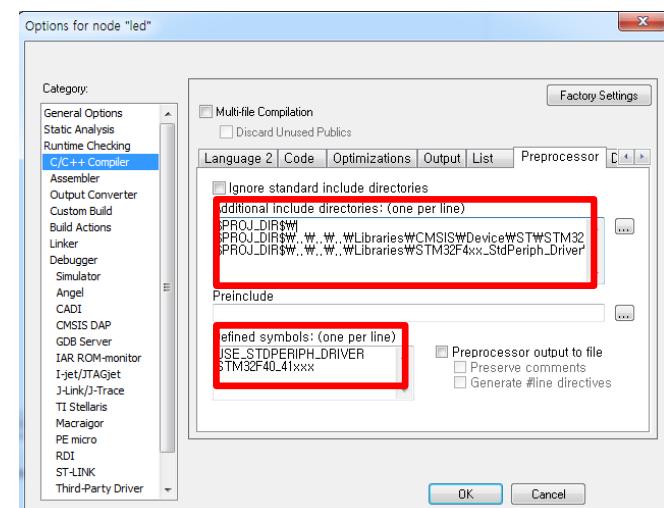
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch12\step1Phase	system_stm32f4xx.c
Cortex_Example\Projects\ch12\step1Phase	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

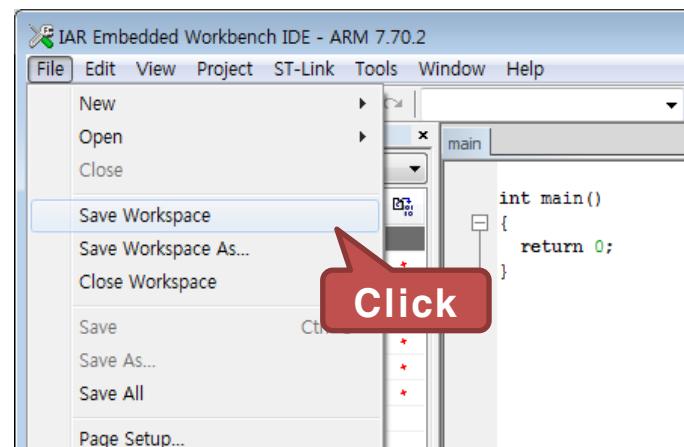
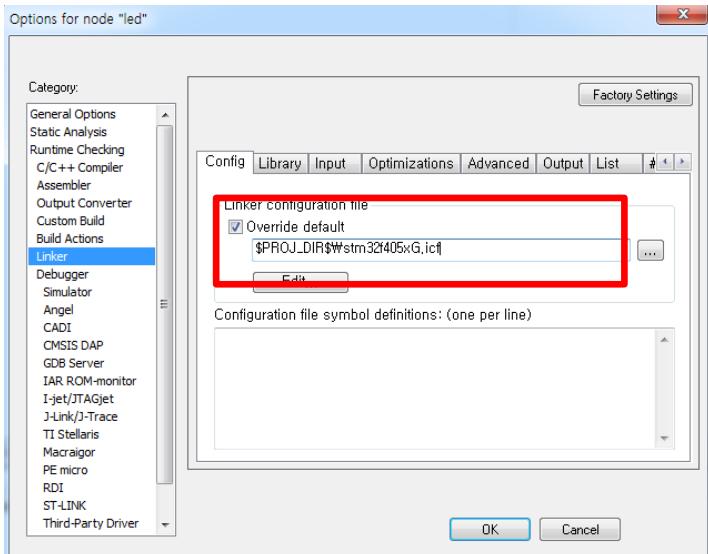
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {          // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE);
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
```

실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

- 구동 프로그램
 - main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9; // A, B  
GPIO_Init(GPIOB, &GPIO_InitStructure);  
// MOTOR1_EN, /A, /B  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3|GPIO_Pin_11|GPIO_Pin_12;  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
  
GPIO_ResetBits(GPIOA, GPIO_Pin_3); // M1 Disable, DC 모터 정지
```

실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

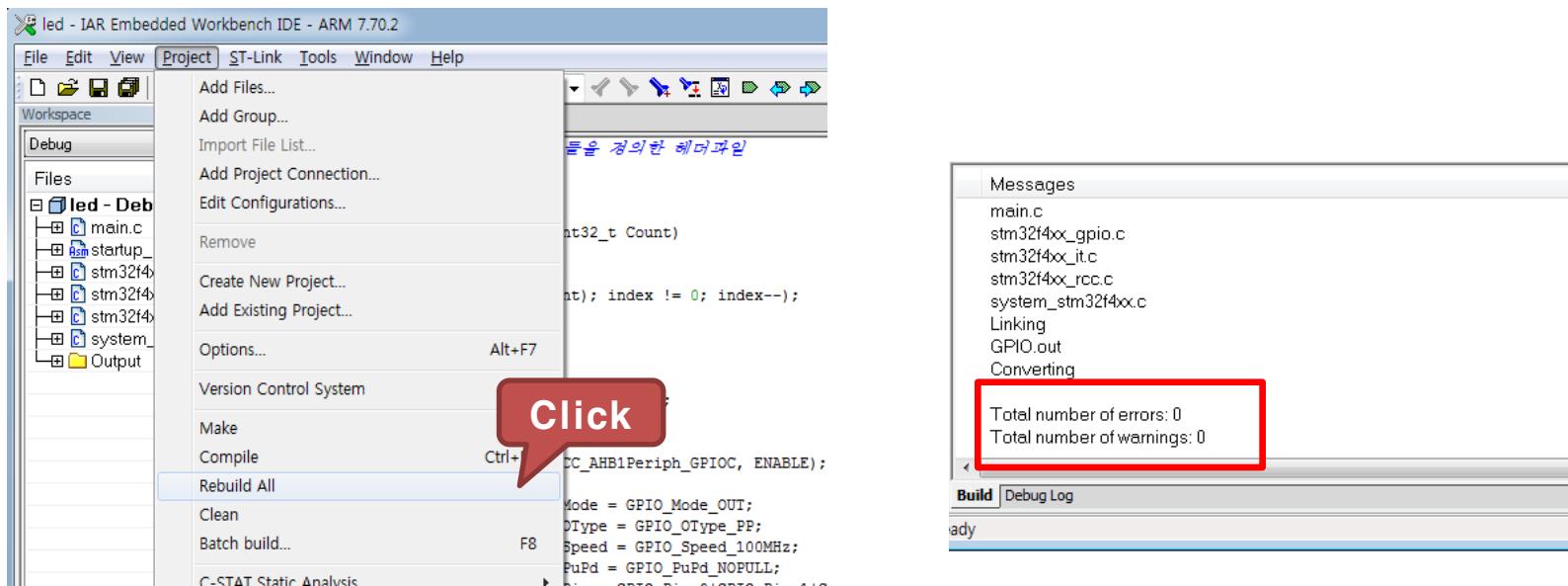
- 구동 프로그램
 - main.c 코드 작성

```
//...
while(1) {
    GPIO_Write(GPIOB, 0x0100); // 1 step
    GPIO_Write(GPIOA, 0x0000);
    Delay(10);
    GPIO_Write(GPIOB, 0x0200); // 2 step
    GPIO_Write(GPIOA, 0x0000);
    Delay(10);
    GPIO_Write(GPIOB, 0x0000); // 3 step
    GPIO_Write(GPIOA, 0x0800);
    Delay(10);
    GPIO_Write(GPIOB, 0x0000); // 4 step
    GPIO_Write(GPIOA, 0x1000);
    Delay(10);
}
}
```

실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

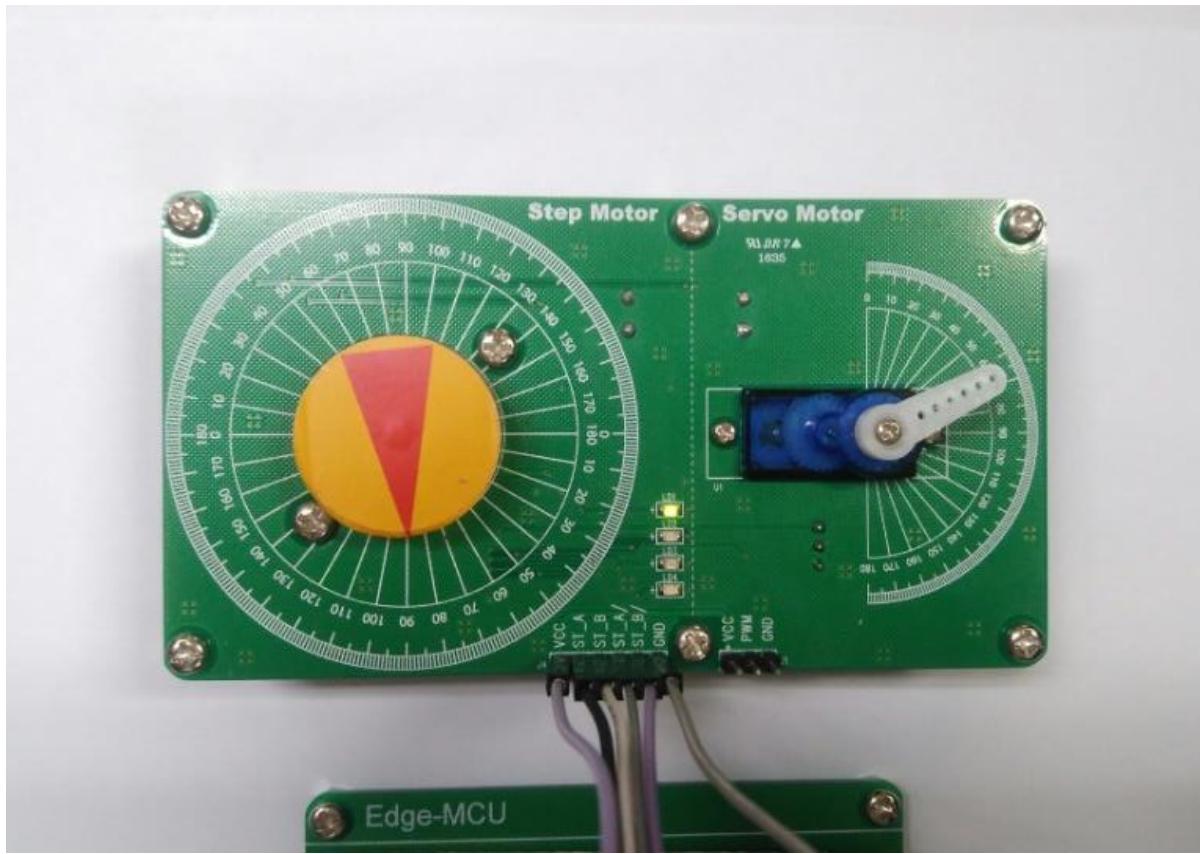
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 step1Phase 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 1 : 1상 여자 방식으로 스텝 모터 돌리기

- 실행 결과
 - 스텝 모터가 1상 여자 방식으로 시계 방향으로 한바퀴 회전 후 1초간 대기



실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

- 실습 개요
 - STM32F405의 GPIO핀에 스텝 모터의 제어 신호를 연결하여, 스텝 모터를 회전 시키기
 - 2상 여자 방식을 사용하여 시계방향으로 1바퀴 돌고 1초 정지 시키는 것을 무한반복
 - 본 실험에서 사용하고 있는 Step 모터는 1상, 2상 여자 방식일 때 펄스당 7.5도, 1-2상 여자 방식일 때 펄스당 3.75도 회전
- 실습 목표
 - 스텝 모터의 동작원리를 이해
 - STM32F405의 GPIO를 이용한 스텝 모터 구동 방법 습득

실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

- 구동 프로그램 : 사전지식
 - 스텝 모터의 2상 여자 방식 신호 만들기
 - 2상 여자 방식이므로 스텝 모터는 펄스당 7.5도가 회전
 - 시계 방향일 경우는 B(9:8)에 차례대로, 0x0300, 0x0200, 0x0000, 0x0100, 동시에 A(12:11)에 차례대로, 0x0000, 0x0800, 0x1800, 0x1000, 이러한 명령이 총 $360/7.5 = 48$, 즉 48개의 펄스가 입력되어야 함
 - 한 주기가 4개의 명령으로 이루어 지므로 12 주기의 명령이 포트 A, B로 입력되어야 스텝 모터는 1회전하게 됨
- 2상 여자 방식 제어 신호(반 시계 방향)

신호	1	2	3	4	5	6	7	8	9
PB8(A)	1	0	0	1	1	0	0	1	1
PB9(B)	1	1	0	0	1	1	0	0	1
PA11(/A)	0	1	1	0	0	1	1	0	0
PA12(/B)	0	0	1	1	0	0	1	1	0
B 포트	0x0300	0x0200	0x0000	0x0100	0x0300	0x0200	0x0000	0x0100	0x0300
A 포트	0x0000	0x0800	0x1800	0x1000	0x0000	0x0800	0x1800	0x1000	0x0000

실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

- 구동 프로그램 : 사전지식

- GPIO 사용

- OUT_0(STEP_A) → PORTB(PB8)
 - OUT_1(STEP_B) → PORTB(PB9)
 - OUT_2(STEP_A) → PORTA(PA11)
 - OUT_3(STEP_B) → PORTA(PA12)

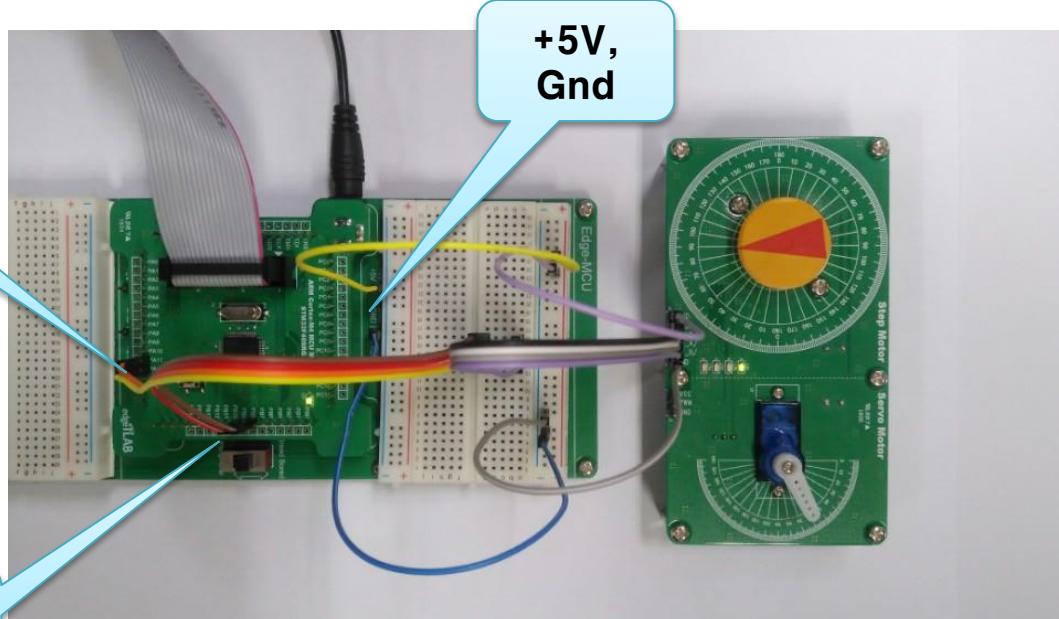
- 시간 지연 함수 사용

- 2상 여자 방식 신호의 4 스텝을 12번 실행 시켜 48 스텝을 진행
 - 2상 여자 방식 신호 만들기를 위해서는 B(9:8)/A(12:11)에 데이터를 보내는 시간 간격은 일정하게 유지
 - 시간 지연 함수를 이용하여 10ms 마다 상을 변화시킴

실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

● 실습 준비

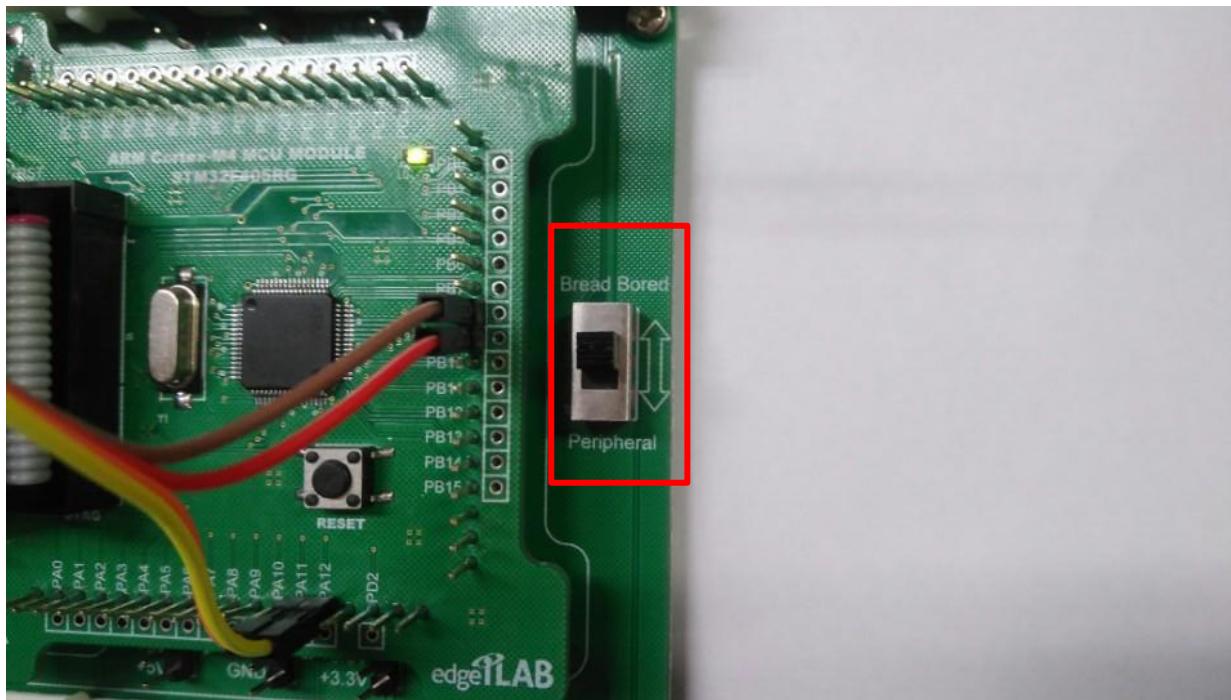
- 사용보드를 다음 그림과 같이 연결
 - Edge-MCU보드의 **+5V** → Bread보드 (+) → Step 모터 보드의 **VCC**
 - Edge-MCU보드의 **PB8** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_A**
 - Edge-MCU보드의 **PB9** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_B**
 - Edge-MCU보드의 **PA11** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_A**
 - Edge-MCU보드의 **PA12** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_B**
 - Edge-MCU보드의 **GND** → Bread보드 (-) → Edge-Peri 보드의 **GND**



실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

- 실습 준비

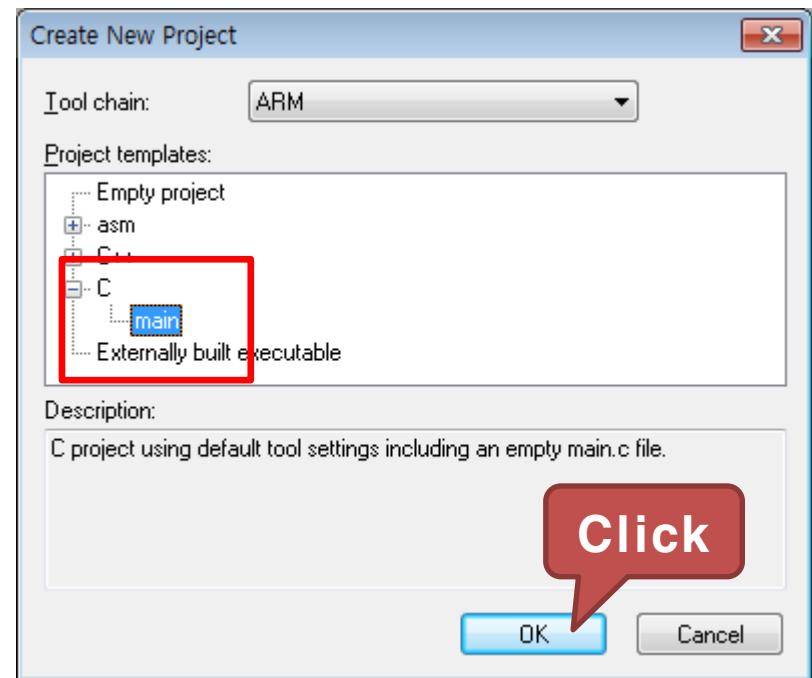
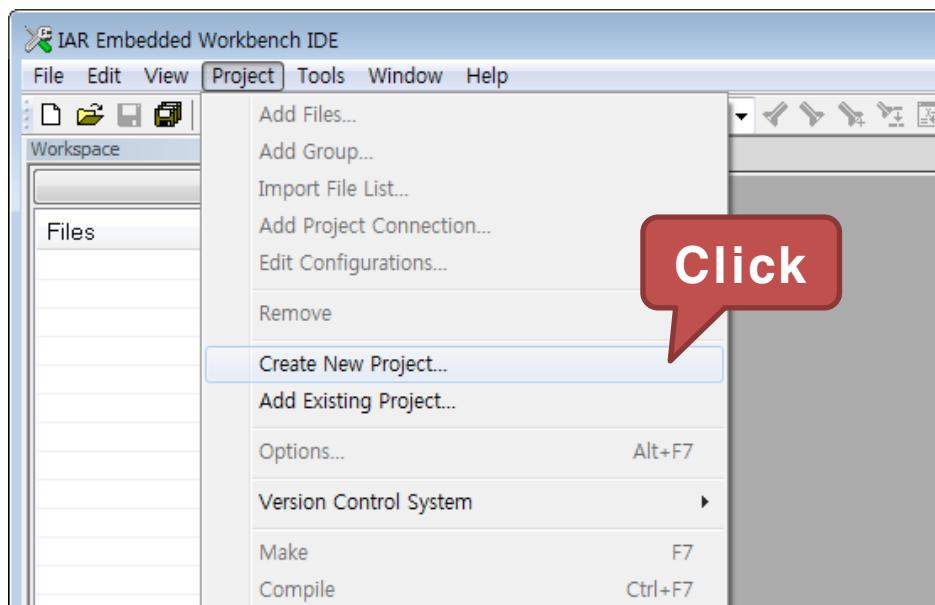
- Edge-MCU보드 오른쪽의 스위치를 Bread Board로 올림



실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch12” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “step2Phase”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

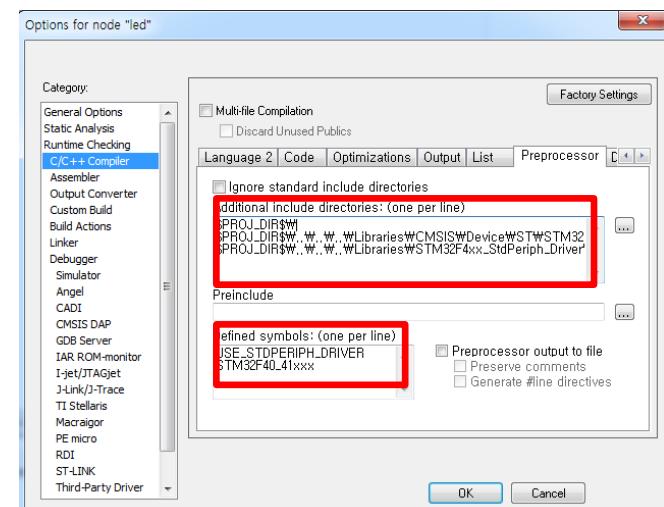
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch12\step2Phase	system_stm32f4xx.c
Cortex_Example\Projects\ch12\step2Phase	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

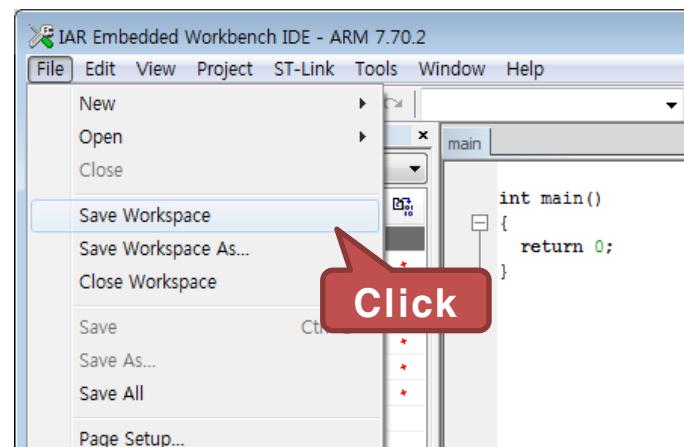
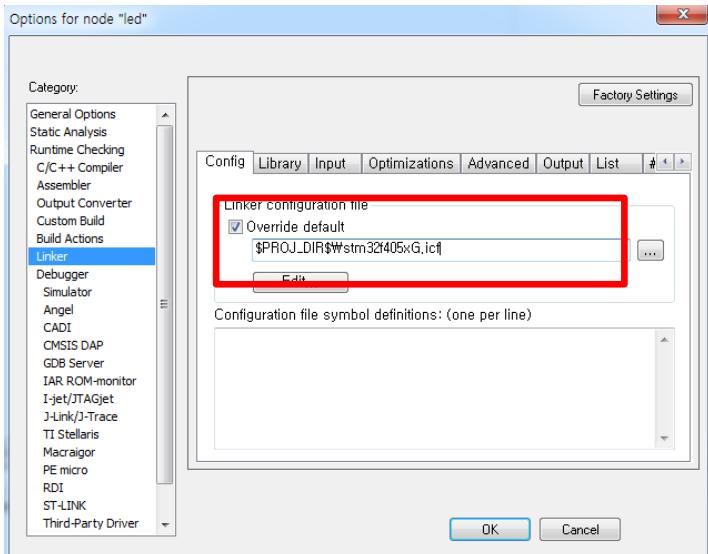
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

- 구동 프로그램
 - main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {          // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    unsigned char i;

    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE);
}
```

실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

- 구동 프로그램

- main.c 코드 작성

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;      // A, B
GPIO_Init(GPIOB, &GPIO_InitStructure);
// MOTOR1_EN, /A, /B
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3|GPIO_Pin_11|GPIO_Pin_12;
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_ResetBits(GPIOA, GPIO_Pin_3); // M1 Disable, DC 모터 정지

while(1) {
    for ( i = 0; i < 12 ; i++ ) {          // 48 스텝 실행
        GPIO_Write(GPIOB, 0x0300); // 1 step
        GPIO_Write(GPIOA, 0x0000);
        Delay(10);
```

실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

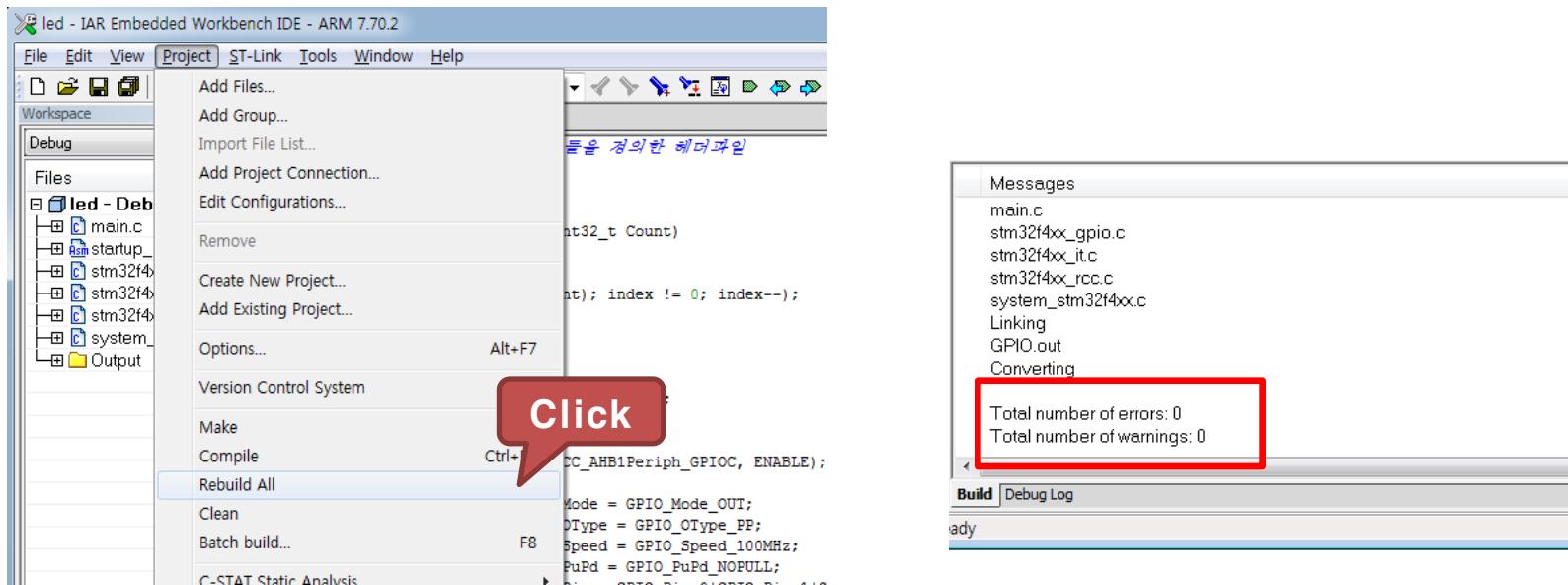
- 구동 프로그램
 - main.c 코드 작성

```
//...
GPIO_Write(GPIOB, 0x0100);           // 2 step
GPIO_Write(GPIOA, 0x1000);
Delay(10);
GPIO_Write(GPIOB, 0x0000);           // 3 step
GPIO_Write(GPIOA, 0x1800);
Delay(10);
GPIO_Write(GPIOB, 0x0200);           // 4 step
GPIO_Write(GPIOA, 0x0800);
Delay(10);
}
Delay(1000);
}
```

실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

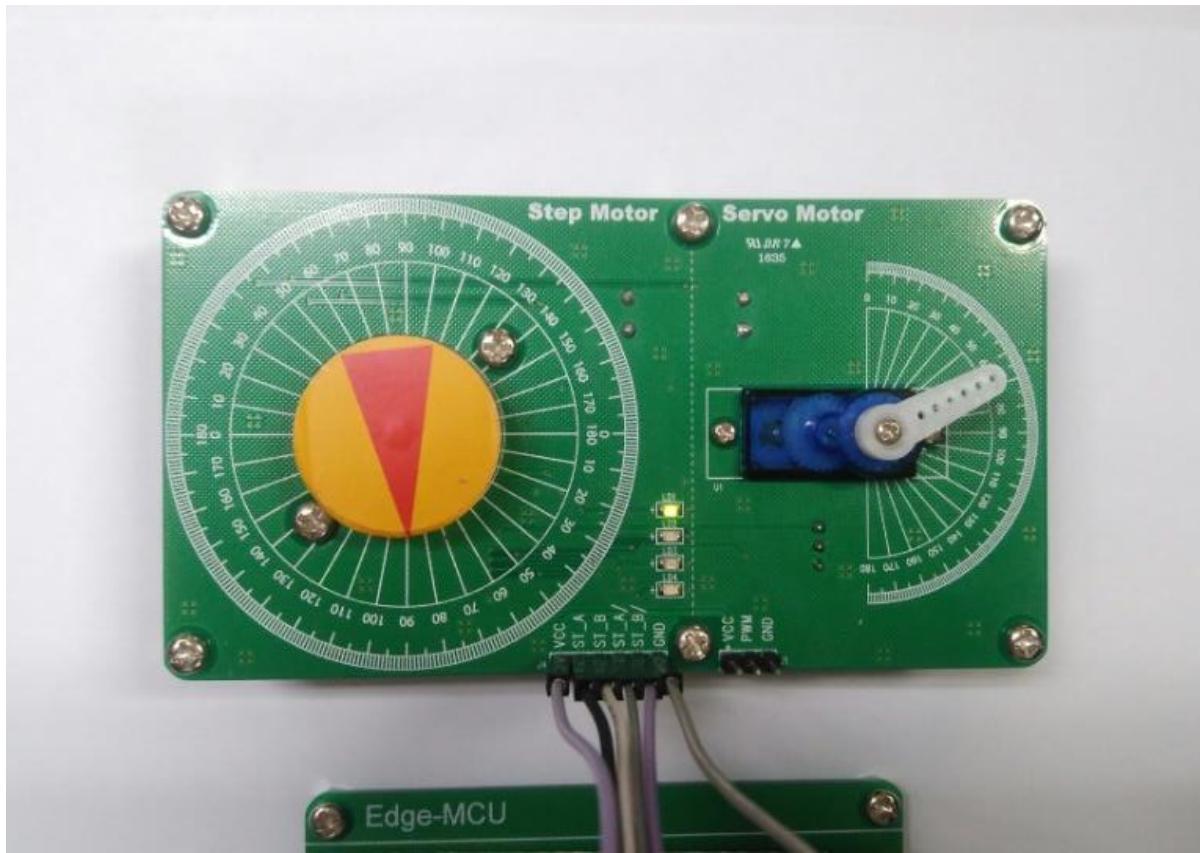
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 step2Phase 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 2 : 2상 여자 방식으로 스텝 모터 돌리기

- 실행 결과
 - 스텝 모터가 2상 여자 방식으로 시계 방향으로 한바퀴 회전 후 1초간 대기



- 실습 개요
 - STM32F405의 GPIO핀에 스텝 모터의 제어 신호를 연결하여, 스텝 모터를 회전 시키기
 - 1-2상 여자 방식을 사용하여 시계방향으로 90도 회전하고 1초 정지 시키는 것을 무한 반복
 - 본 실험에서 사용하고 있는 Step 모터는 1상, 2상 여자 방식일 때 펄스당 7.5도, 1-2상 여자 방식일 때 펄스당 3.75도 회전
- 실습 목표
 - 스텝 모터의 동작원리를 이해
 - STM32F405의 GPIO를 이용한 스텝 모터 구동 방법 습득

실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 구동 프로그램 : 사전지식
 - 스텝 모터의 1-2상 여자 방식 신호 만들기
 - 1-2상 여자 방식이므로 스텝 모터는 펄스당 3.75도가 회전
 - 반 시계 방향일 경우는 B(9:8)에 차례대로, 0x0100, 0x0300, 0x0200, 0x0200, 0x0000, 0x0000, 0x0000, 0x0100, 동시에 A(12:11)에 차례대로, 0x0000, 0x0000, 0x0000, 0x0800, 0x0800, 0x1800, 0x1000, 0x1000, 이러한 명령이 총 $360/3.75 = 96$, 즉 96개의 펄스가 입력
 - 90도만 회전 하려면 $96/4 = 24$, 즉 24개의 스텝이 필요
- 1-2상 여자 방식 제어 신호(반 시계 방향)

신호	1	2	3	4	5	6	7	8	9
PB8(A)	1	1	0	0	0	0	0	1	1
PB9(B)	0	1	1	1	0	0	0	0	0
PA11(/A)	0	0	0	1	1	1	0	0	0
PA12(/B)	0	0	0	0	0	1	1	1	0
B 포트	0x0100	0x0300	0x0200	0x0200	0x0000	0x0000	0x0000	0x0100	0x0100
A 포트	0x0000	0x0000	0x0000	0x0800	0x0800	0x1800	0x1000	0x1000	0x0000

- 구동 프로그램 : 사전지식

- GPIO 사용

- OUT_0(STEP_A) → PORTB(PB8)
 - OUT_1(STEP_B) → PORTB(PB9)
 - OUT_2(STEP_A) → PORTA(PA11)
 - OUT_3(STEP_B) → PORTA(PA12)

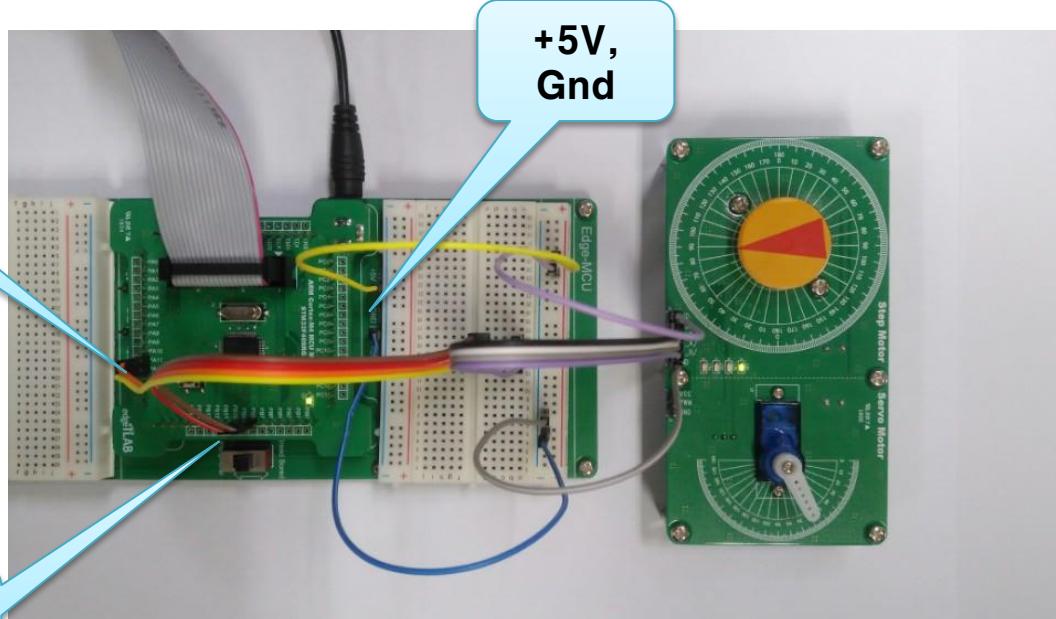
- 시간 지연 함수 사용

- 1-2상 여자 방식 신호의 스텝을 24 스텝 실행
 - 1-2상 여자 방식 신호 만들기를 위해서는 B(9:8)/A(12:11) 에 데이터를 보내는 시간 간격은 일정하게 유지
 - 시간 지연 함수를 이용하여 10ms 마다 상을 변화시킴

실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 실습 준비

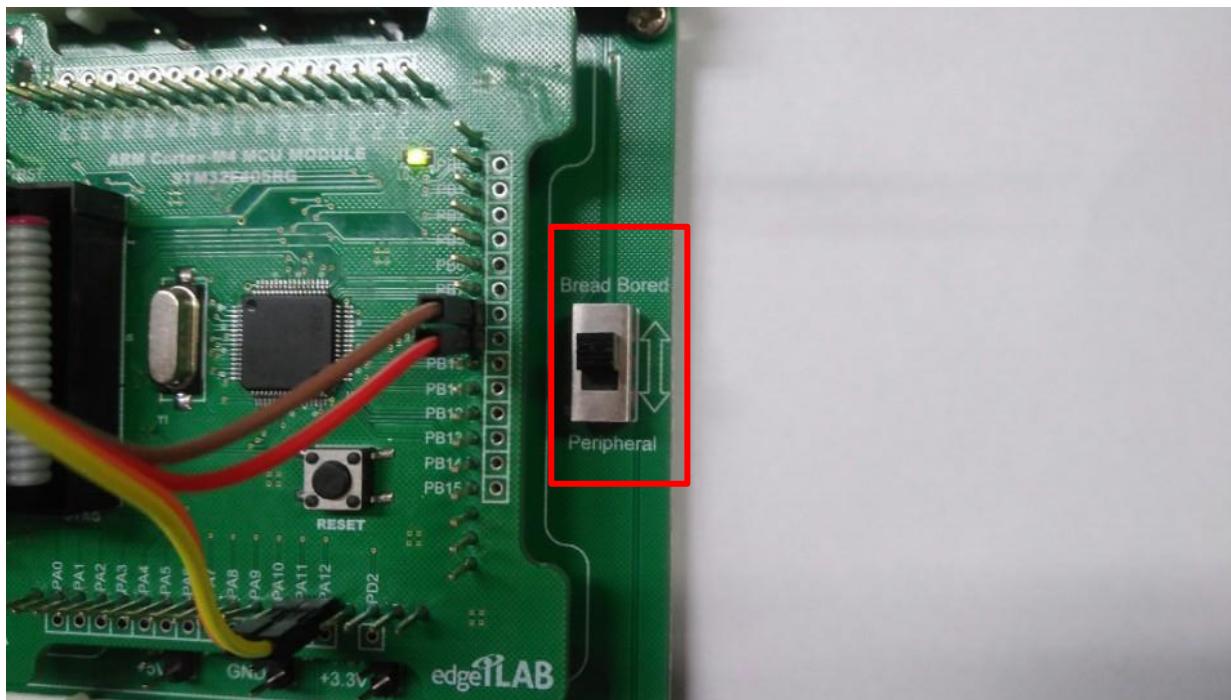
- 사용보드를 다음 그림과 같이 연결
 - Edge-MCU보드의 **+5V** → Bread보드 (+) → Step 모터 보드의 **VCC**
 - Edge-MCU보드의 **PB8** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_A**
 - Edge-MCU보드의 **PB9** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_B**
 - Edge-MCU보드의 **PA11** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_A**
 - Edge-MCU보드의 **PA12** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_B**
 - Edge-MCU보드의 GND → Bread보드 (-) → Edge-Peri 보드의 GND



실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 실습 준비

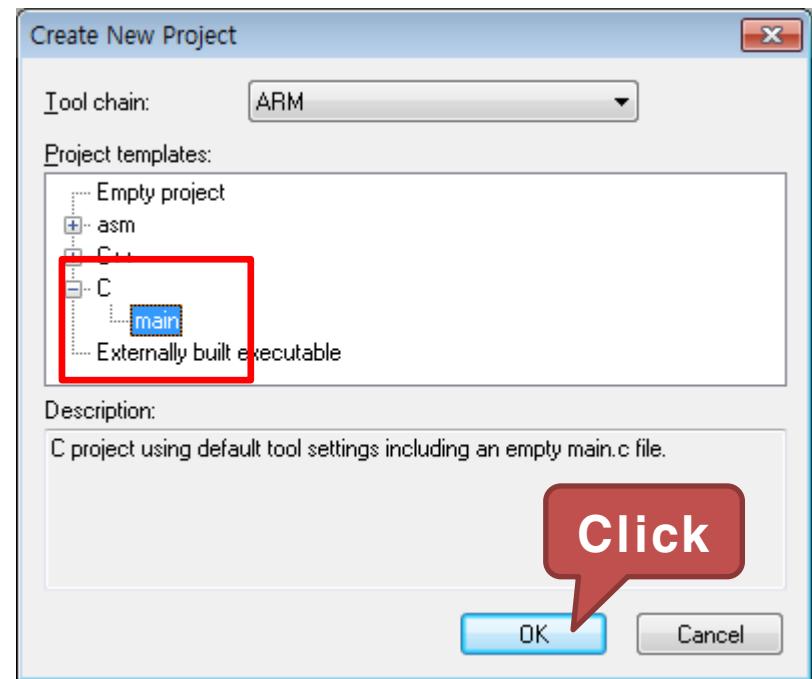
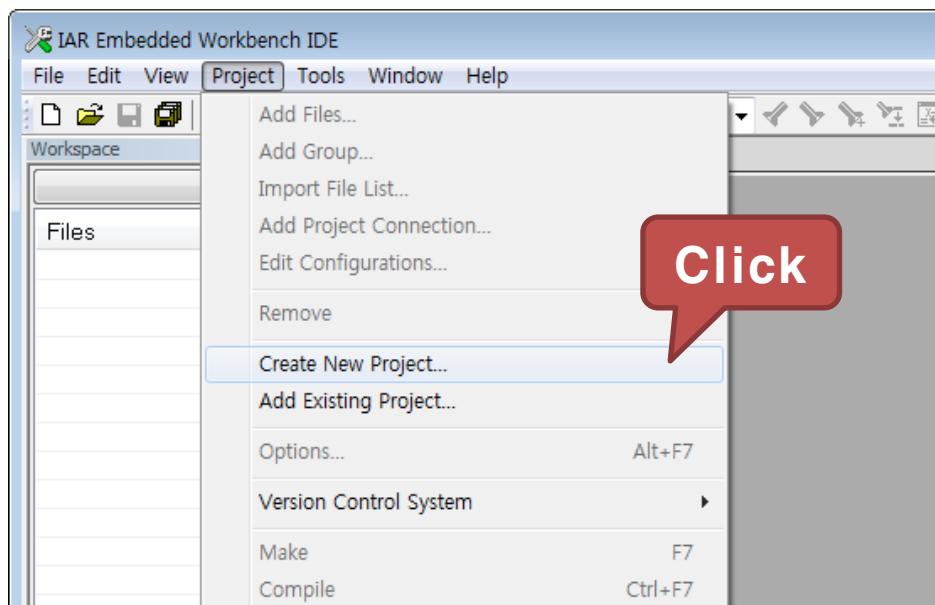
- Edge-MCU보드 오른쪽의 스위치를 Bread Board로 올림



실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch12” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “step12Phase1”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 라이브러리 파일 추가

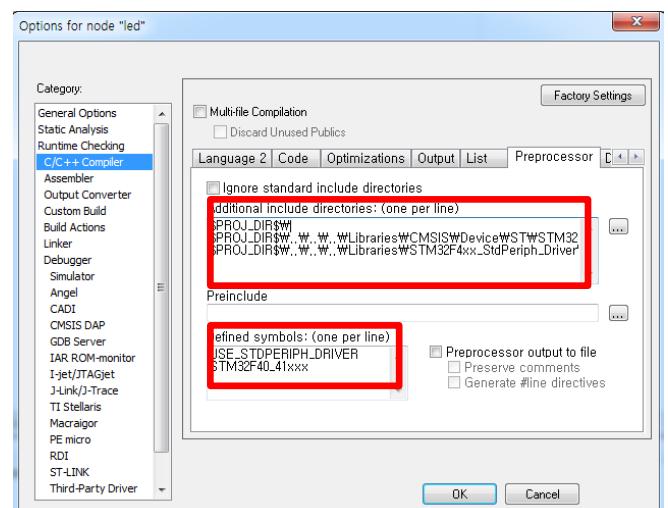
- GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
- 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
- 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch12\step12Phase1	system_stm32f4xx.c
Cortex_Example\Projects\ch12\step12Phase1	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

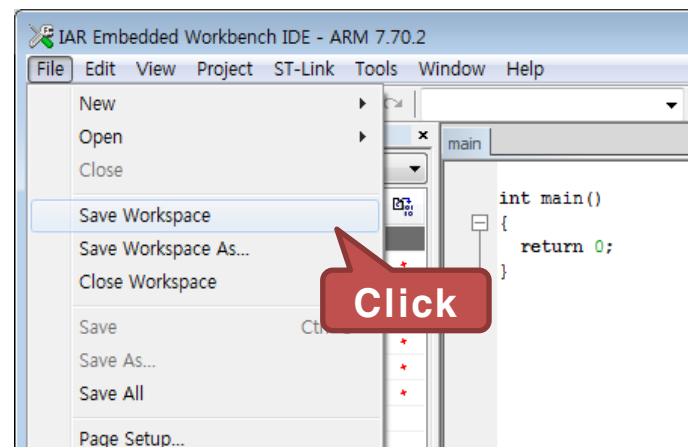
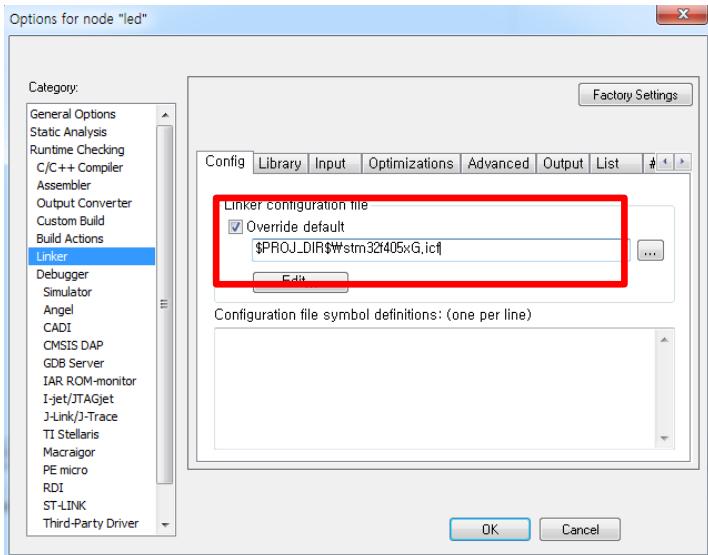
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

// 1-2상 여자 방식 제어 신호를 저장하고 있는 배열
unsigned int StepB[8] =
{0x0100,0x0000,0x0000,0x0000,0x0200,0x0200,0x0300,0x0100};
unsigned int StepA[8] =
{0x1000,0x1000,0x1800,0x0800,0x0800,0x0000,0x0000, 0x0000};
```

실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 구동 프로그램
 - main.c 코드 작성

```
int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    unsigned char i, t;

    RCC_AHB1PeriphClockCmd(
        RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9; // A, B
    GPIO_Init(GPIOB, &GPIO_InitStructure);
```

실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 구동 프로그램

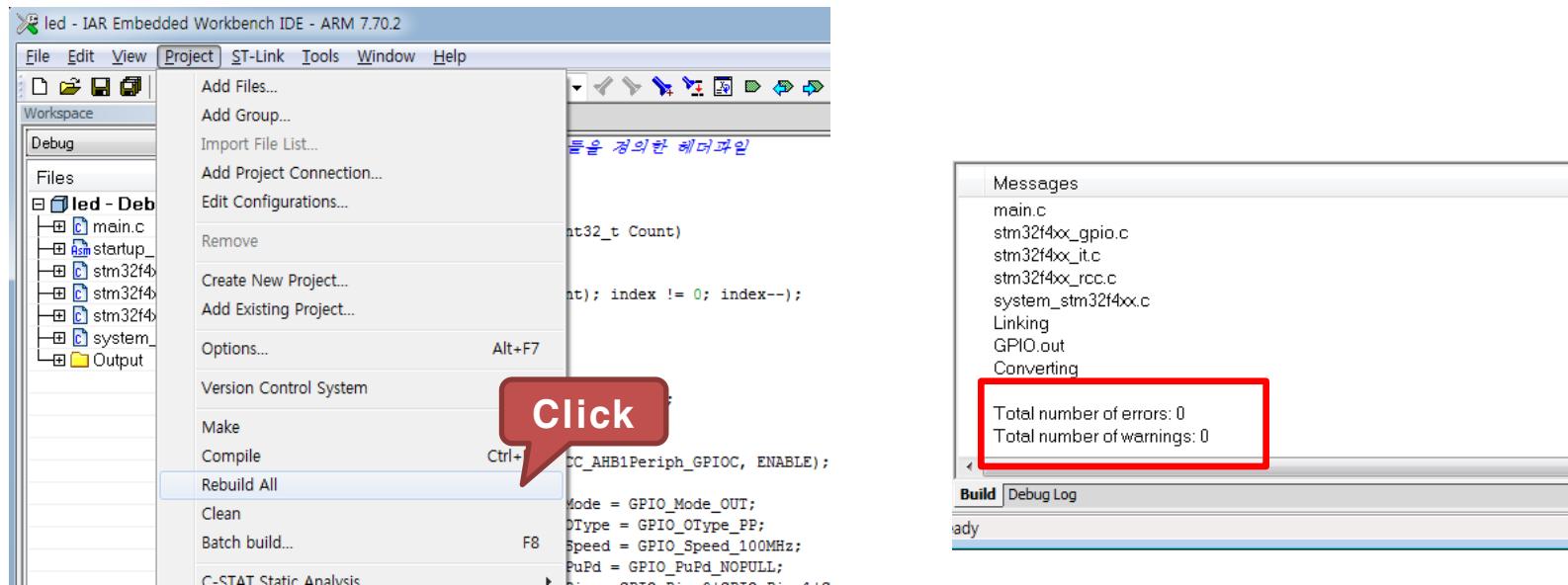
- main.c 코드 작성

```
// MOTOR1_EN, /A, /B  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3|GPIO_Pin_11|GPIO_Pin_12;  
GPIO_Init(GPIOA, &GPIO_InitStructure);  
GPIO_ResetBits(GPIOA, GPIO_Pin_3); // M1 Disable, DC 모터 정지  
while(1) {  
    // 명령당 3.75도 회전, 90도 회전하려면 총 24개의 명령이 필요  
    for ( i = 0; i < 24 ; i++ ) {  
        GPIO_Write(GPIOB, StepB[t]); // 한 스텝 실행  
        GPIO_Write(GPIOA, StepA[t]);  
        t++; // 다음 스텝 실행을 위해 t 증가  
        if(t > 7) t = 0; // 8 스텝을 초과하지 않도록 초기화  
        Delay(10);  
    }  
    Delay(1000);  
}
```

실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

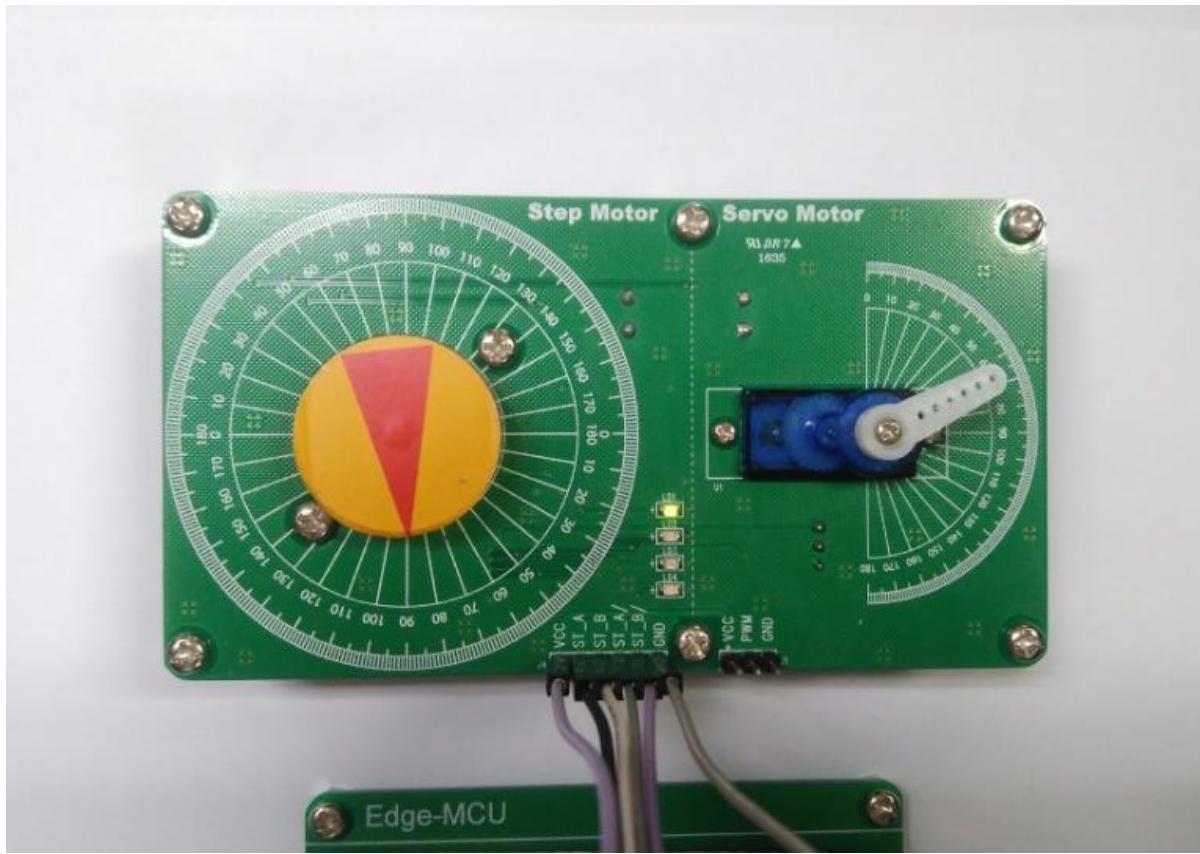
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 step12Phase1 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 3 : 1-2상 여자 방식으로 스텝 모터 돌리기 1

- 실행 결과
 - 스텝 모터가 1-2상 여자 방식으로 시계방향으로 90도 회전한 후 1초간 대기



실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

● 실습 개요

- STM32F405의 GPIO핀에 스텝 모터의 제어 신호를 연결하여, 스텝 모터를 회전 시키기
- 타이머를 이용하여 2초마다 모터가 방향을 전환
- 스텝 모터의 구동 방식은 1-2상 여자 방식
- 앞에서 배운 타이머의 기능을 복합적으로 이용

● 실습 목표

- 스텝 모터의 동작원리를 이해
- STM32F405의 GPIO를 이용한 스텝 모터 구동 방법 습득
- 타이머, 인터럽트, GPIO 제어의 복합적인 프로그램 능력 배양

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 구동 프로그램 : 사전지식
 - 스텝 모터의 1-2상 여자 방식 신호 만들기
 - 1-2상 여자 방식이므로 스텝 모터는 펄스당 3.75도가 회전
 - 반 시계 방향일 경우는 B(9:8)에 차례대로, 0x0100, 0x0300, 0x0200, 0x0200, 0x0000, 0x0000, 0x0000, 0x0100, 동시에 A(12:11)에 차례대로, 0x0000, 0x0000, 0x0000, 0x0800, 0x0800, 0x1800, 0x1000, 0x1000, 이러한 명령이 총 $360/3.75 = 96$, 즉 96개의 펄스가 입력
 - 90도만 회전 하려면 $96/4 = 24$, 즉 24개의 스텝이 필요
- 1-2상 여자 방식 제어 신호(반 시계 방향)

신호	1	2	3	4	5	6	7	8	9
PB8(A)	1	1	0	0	0	0	0	1	1
PB9(B)	0	1	1	1	0	0	0	0	0
PA11(/A)	0	0	0	1	1	1	0	0	0
PA12(/B)	0	0	0	0	0	1	1	1	0
B 포트	0x0100	0x0300	0x0200	0x0200	0x0000	0x0000	0x0000	0x0100	0x0100
A 포트	0x0000	0x0000	0x0000	0x0800	0x0800	0x1800	0x1000	0x1000	0x0000

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 구동 프로그램 : 사전지식

- GPIO 사용

- OUT_0(STEP_A) → PORTB(PB8)
 - OUT_1(STEP_B) → PORTB(PB9)
 - OUT_2(STEP_A) → PORTA(PA11)
 - OUT_3(STEP_B) → PORTA(PA12)

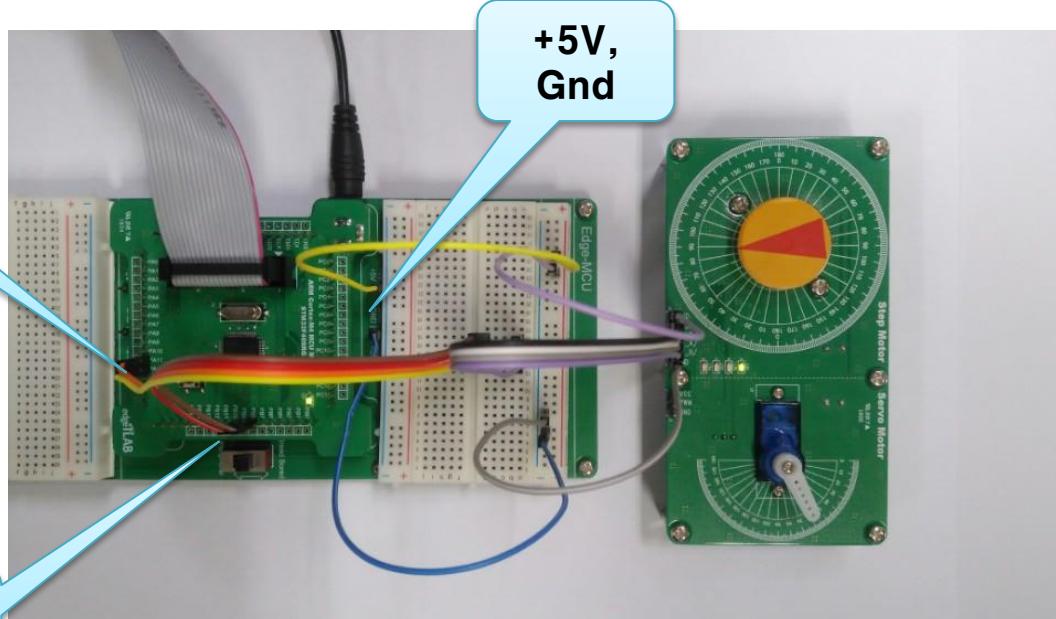
- 타이머 사용

- 1-2상 여자 방식 신호 만들기를 위해서는 B(9:8)/A(12:11) 에 데이터를 보내는 시간 간격은 일정하게 유지
 - 이를 위한 타이머가 필요 여기서는 타이머/카운터 0를 사용하며, 타이머의 주기는 45KHz

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 실습 준비

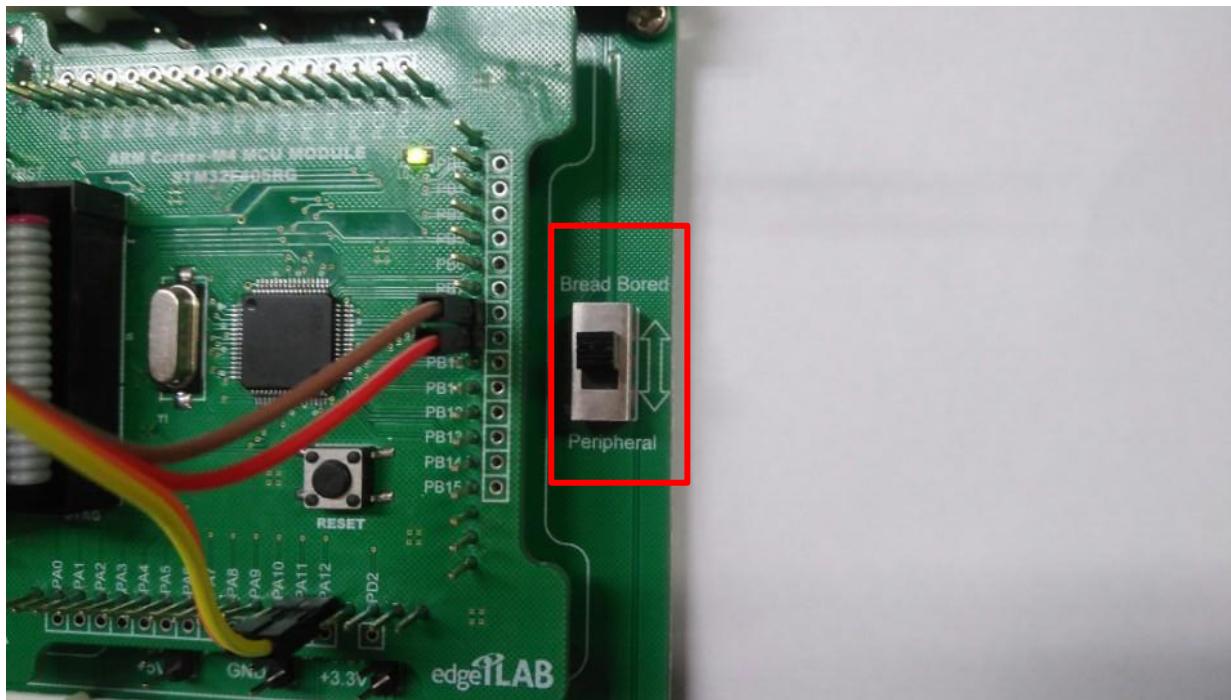
- 사용보드를 다음 그림과 같이 연결
 - Edge-MCU보드의 **+5V** → Bread보드 (+) → Step 모터 보드의 **VCC**
 - Edge-MCU보드의 **PB8** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_A**
 - Edge-MCU보드의 **PB9** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_B**
 - Edge-MCU보드의 **PA11** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_A**
 - Edge-MCU보드의 **PA12** → Bread보드 IC영역 → Edge-Peri 보드의 **ST_B**
 - Edge-MCU보드의 GND → Bread보드 (-) → Edge-Peri 보드의 GND



실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 실습 준비

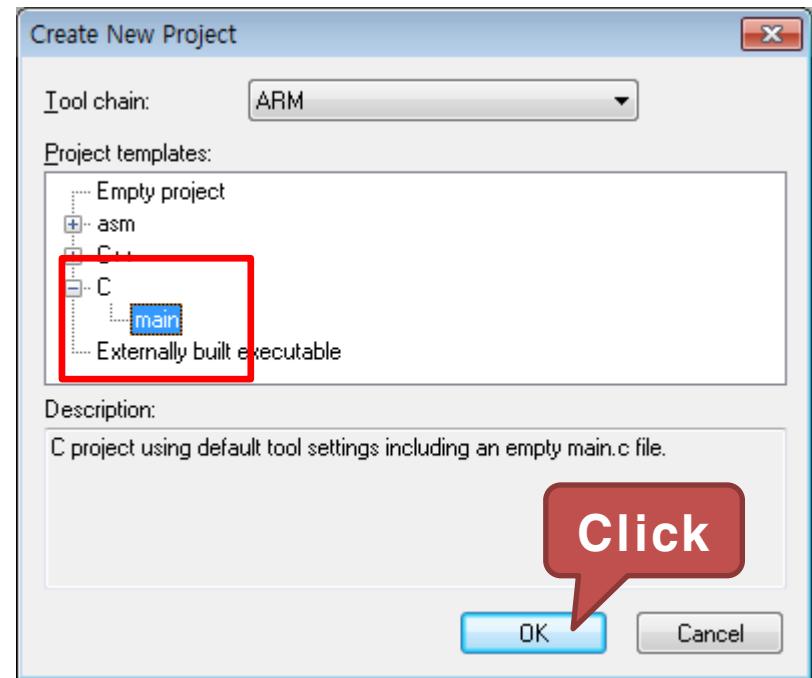
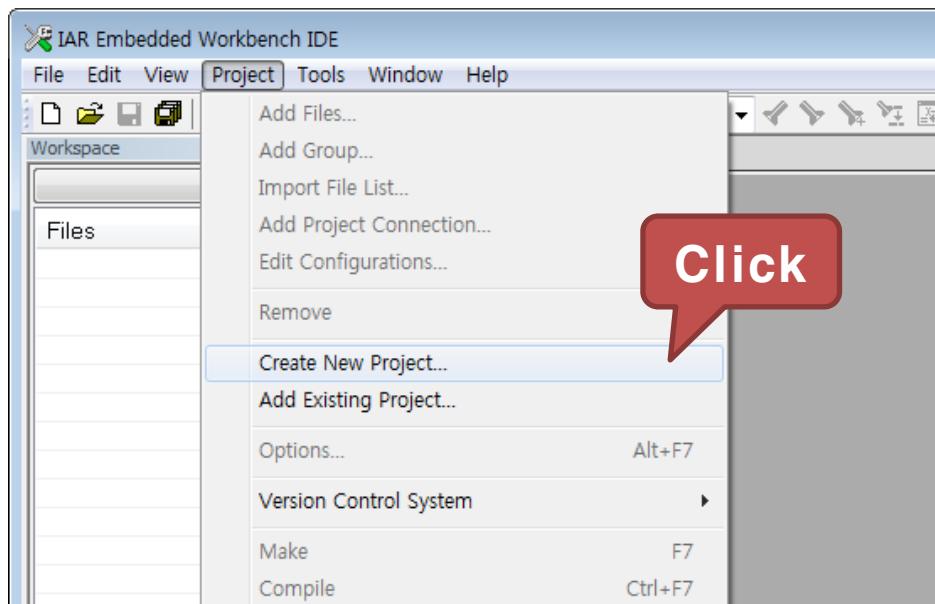
- Edge-MCU보드 오른쪽의 스위치를 Bread Board로 올림



실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch12” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “step12Phase2”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 라이브러리 파일 추가

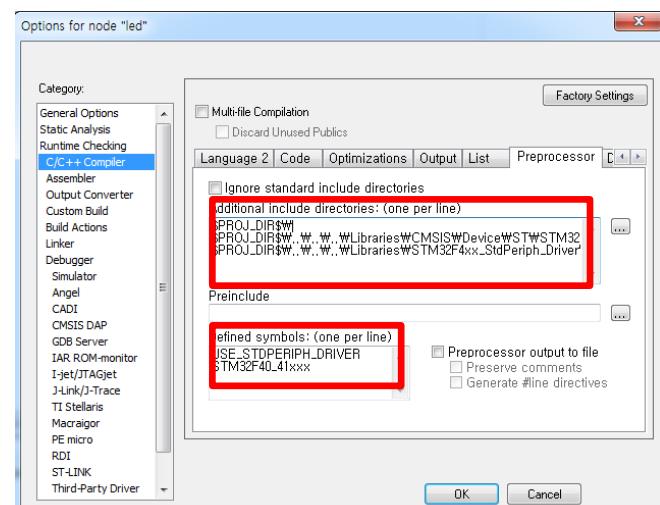
- GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
- 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
- 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch12\step12Phase1	system_stm32f4xx.c
Cortex_Example\Projects\ch12\step12Phase1	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_misc.c

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

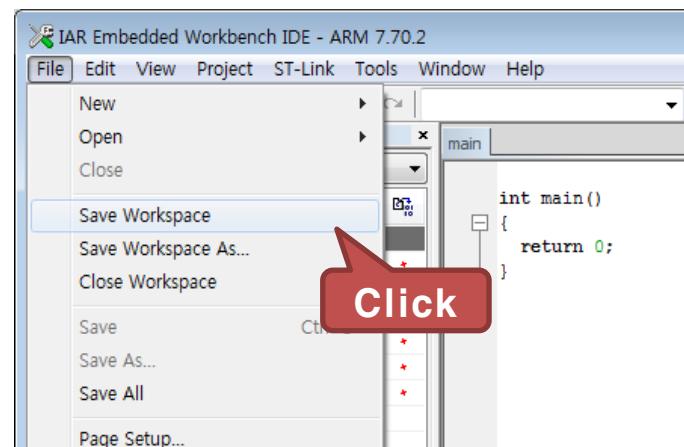
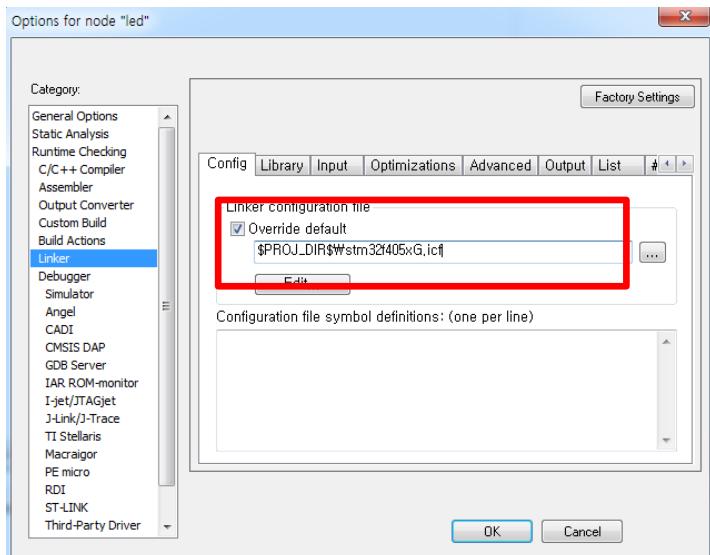
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"
#define DIR_L 0
#define DIR_R 1
unsigned char mot_cnt=0;
volatile unsigned char dir=DIR_R;      // 처음방향은 우측
//1-2상 여자 방식 제어 신호를 저장 하고 있는 배열
unsigned int StepB[8] =
    {0x0100,0x0000,0x0000,0x0000,0x0200,0x0200,0x0300,0x0100};
unsigned int StepA[8] =
    {0x1000,0x1000,0x1800,0x0800,0x0800,0x0000,0x0000, 0x0000};

int main(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    NVIC_InitTypeDef      NVIC_InitStructure;
```

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 구동 프로그램

- main.c 코드 작성

```
//...
RCC_AHB1PeriphClockCmd(
    RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOB, ENABLE);
RCC_APB1PeriphClockCmd(
    RCC_APB1Periph_TIM3|RCC_APB1Periph_TIM4, ENABLE);
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9; // A, B
GPIO_Init(GPIOB, &GPIO_InitStructure);

// MOTOR1_EN, /A, /B
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3|GPIO_Pin_11|GPIO_Pin_12;
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_ResetBits(GPIOA, GPIO_Pin_3); // M1 Disable, DC 모터 정지
```

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 구동 프로그램

- main.c 코드 작성

```
// (168Mhz/2)/8400 = 10KHz(1ms)
TIM_TimeBaseStructure.TIM_Prescaler = 8400-1;
TIM_TimeBaseStructure.TIM_Period = 20000-1;
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

// (168Mhz/2)/8400 = 10KHz(1ms)
TIM_TimeBaseStructure.TIM_Prescaler = 8400-1;
TIM_TimeBaseStructure.TIM_Period = 220-1;
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
```

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 구동 프로그램

- main.c 코드 작성

```
// 타이머 3, 4 인터럽트의 우선순위를 설정  
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);  
  
NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;  
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;  
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x01;  
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;  
NVIC_Init(&NVIC_InitStructure);  
  
NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;  
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x02;  
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x02;  
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;  
NVIC_Init(&NVIC_InitStructure);
```

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 구동 프로그램

- main.c 코드 작성

```
//...
TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);
TIM_Cmd(TIM3,ENABLE);
TIM_Cmd(TIM4,ENABLE);

while(1) {}

void TIM3_IRQHandler(void) {      // 1s
    if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) {
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
        dir ^=1;      // 방향 전환
    }
}
```

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 구동 프로그램

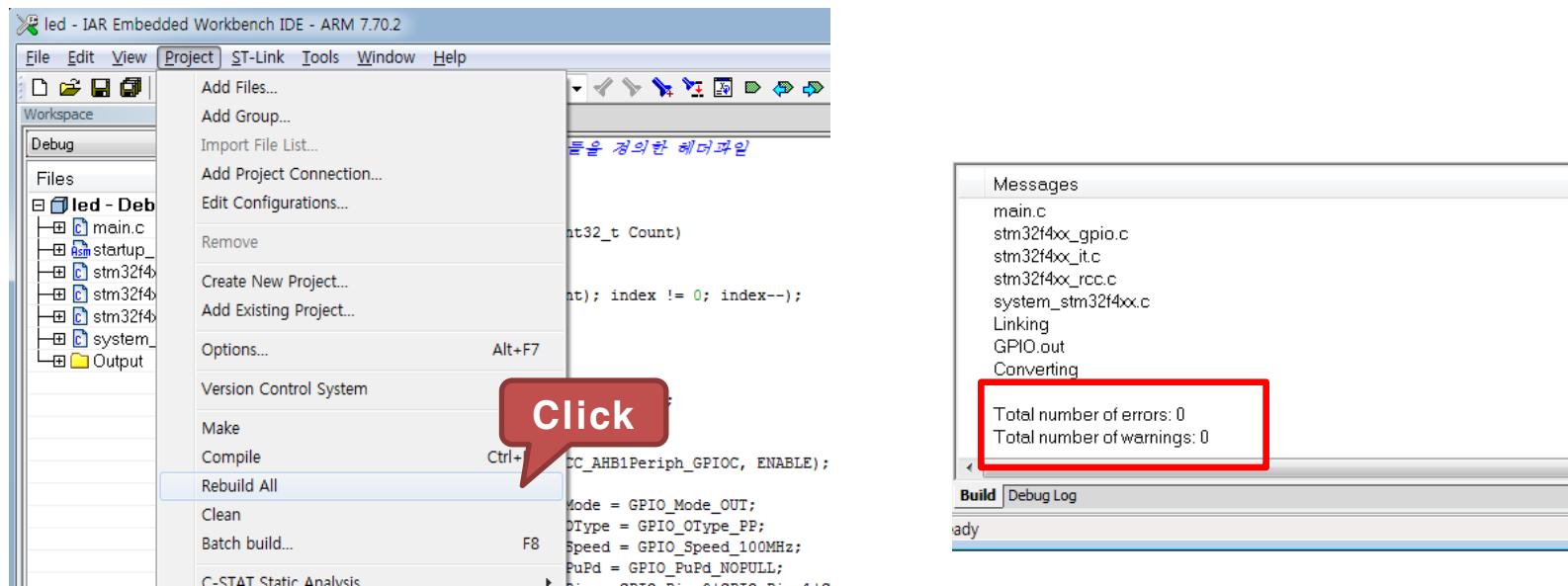
- main.c 코드 작성

```
void TIM4_IRQHandler(void) {      // 22ms -> 45.4545Hz
    if(TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET) {
        TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
        GPIO_Write(GPIOB, StepB[mot_cnt]); // 1-2상 여자 방식한 스텝 진행
        GPIO_Write(GPIOA, StepA[mot_cnt]);
        if(dir==DIR_R) {           // 회전 방향이 우측 방향이면
            if(mot_cnt++==7) mot_cnt=0;          // 스텝 카운터 증가
        }
        else if(mot_cnt--==0) mot_cnt=7;          // 스텝 카운터 감소
    }
}
```

실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

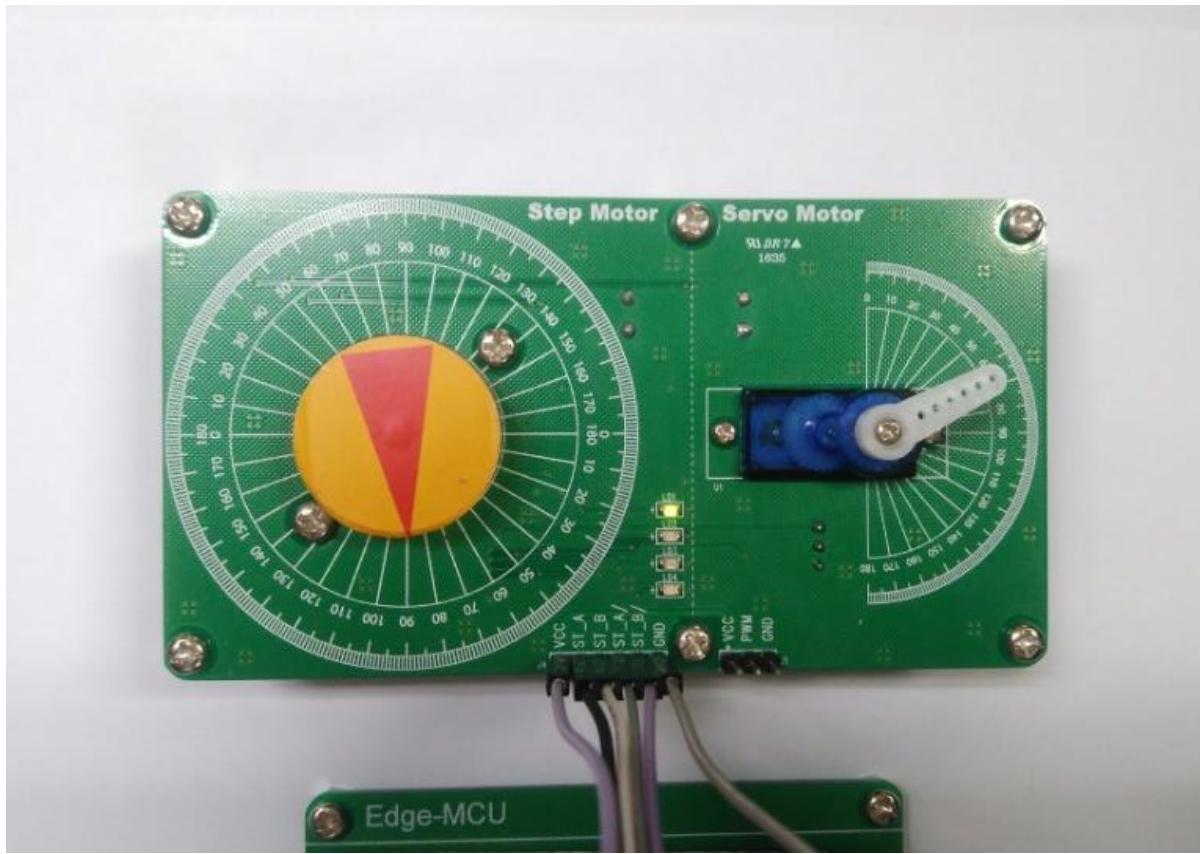
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 step12Phase2 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 4 : 1-2상 여자 방식으로 스텝 모터 돌리기 2

- 실행 결과
 - 스텝 모터가 1-2상 여자 방식으로 회전하며 2초마다 방향 전환





SERVO MOTOR

- 서보 모터(Servo Motor)
- 서보 모터 위치 제어 1
- 서보 모터 위치 제어 2
- 서보 모터 위치 제어 3
- 서보 모터 위치 제어 4



엣지아이랩

서보 모터(Servo Motor)

- 서보의 어원
 - 서보기구라는 용어는 1934년에 H.L.Hazen 교수에 의해 처음 쓰여졌으나, Servo의 어원은 라틴어의 *Servue*(영어의 Slave : 노예)라고 함
 - 노예의 역할이 주인의 명령을 충실히 따르고 육체노동을 하는 것이므로 그러한 역할을 해내는 장치로 명명
- 서보의 역사
 - 본래의 목적인 위치에 최초로 응용한 것은 어뢰
 - 어뢰의 자동 조정에 이어 배와 항공기의 자동 조정이 실현
 - 자동제어는 속도, 서보의 순으로 발달하여 1920년경부터 프로세스제어에도 실현
 - 레이더로 항공기를 자동적으로 추정하는 연구하던 중 시스템으로서 서보계 하드웨어의 개발에 성공하면서 이러한 제어계를 설계, 조정하는 실용적 제어이론의 개발에 성공 피드백 제어이론이 체계화
 - 오일쇼크로 인해 유압서보에서 전기서보로 바뀜
 - 처음에는 DC서보를 사용했으나 교류서보 전동기를 이용한 AC서보의 성능이 향상되면서 더 효율적인 AC서보를 더 많이 사용하게 됨

서보 모터(Servo Motor)

● 서보 모터의 종류

DC 서보모터	AC 서보 모터
브러시 모터(Bushed Motor)	브러시리스 모터(Brushless Motor)
제어구조가 간단하고 쉽다	제어구조가 복잡하고 어렵다
단상으로 제어한다	3상을 제어한다
회전 전기자형	회전 자계형
회전자가 권선으로 방열 나쁘다	고정자가 권성으로 방열이 쉽다
브러시의 유지 보수가 필요하다	브러시의 유지 보수가 필요없다
기계적 구조로 최대 속도가 낮다	전기적 구조로 최대 속도가 높다
정격 용량을 크게 하기 어렵다	정격 용량을 크게 하기 어렵다

서보 모터(Servo Motor)

- 서보 모터의 종류에 따른 장단점

종류	장점	단점
DC서보모터	<ul style="list-style-type: none"> -기동토크가 크다. -크기에 비해 큰 토크 발생. -효율이 높다. -제어기 용이하다. -속도제어 범위가 넓다. -비교적 가격이 저렴하다. 	<ul style="list-style-type: none"> -브러쉬 마찰로 기계적 손실이 큼 -브러쉬의 보수가 필요 -접촉부의 신뢰성이 떨어진다. -정류속도에 한계가 있다. -사용환경에 제한이 있다. -방열이 나쁘다.
동기기형 AC서보모터	<ul style="list-style-type: none"> -브러쉬가 없어 보수가 용이 -내환경성이 우수 -정류속도에 한계가 없다. -신뢰성이 우수하다. -고속, 고 토크의 사용이 가능 -방열이 좋다. 	<ul style="list-style-type: none"> -시스템이 복잡하고 고가이다. -전기적 시정수가 크다. -회전 검출기가 필요 -영구자석을 고정자로 사용하므로 대용량의 모터 제작이 어렵다.
유도기형 AC서보모터	<ul style="list-style-type: none"> -브러쉬가 없어 보수가 용이 -내환경성이 우수 -자석을 사용하지 않는다. -방열이 좋다. -회전검출기가 불필요 	<ul style="list-style-type: none"> -시스템이 복잡하고 고가이다. -전기적 시정수가 크다.

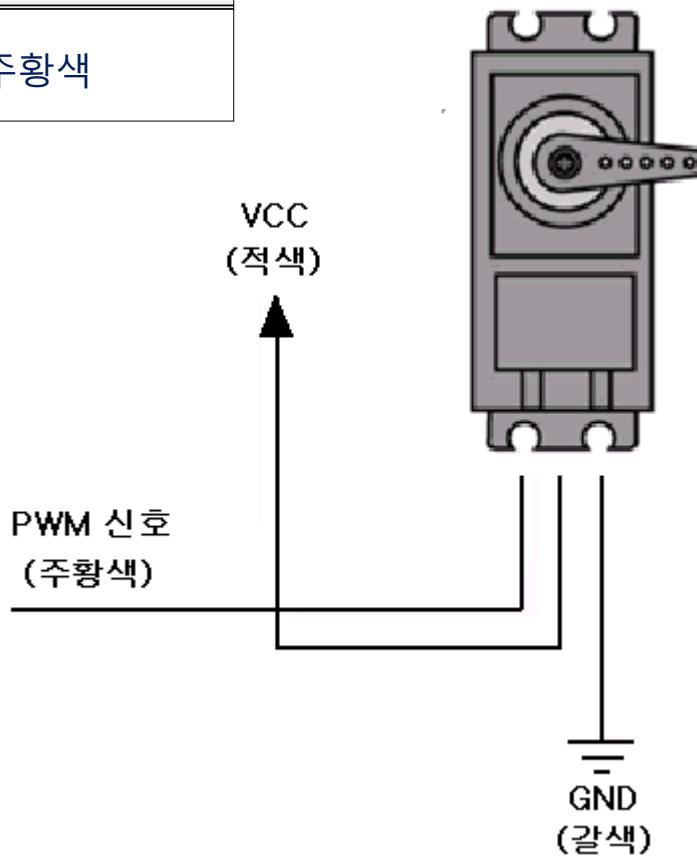
서보 모터(Servo Motor)

- DC 서보 모터의 제어원리
 - DC 서보 모터는 전기자 전류에 대하여 발생 토크의 관계가 직선성으로 우수하고, OA에서부터 FA분야까지 광범위하게 사용
 - 서보 모터는 자동화시스템에서 Actuator로 사용되는 중요한 요소
 - 서보모터는 경량, 소형, 설치의 용이성, 고효율성, 정확한 제어성, 유지, 보수가 용이한 특징
 - 서보모터는 DC, 스텝 모터와는 다르게 각도에 보통 180° 의 제한

서보 모터(Servo Motor)

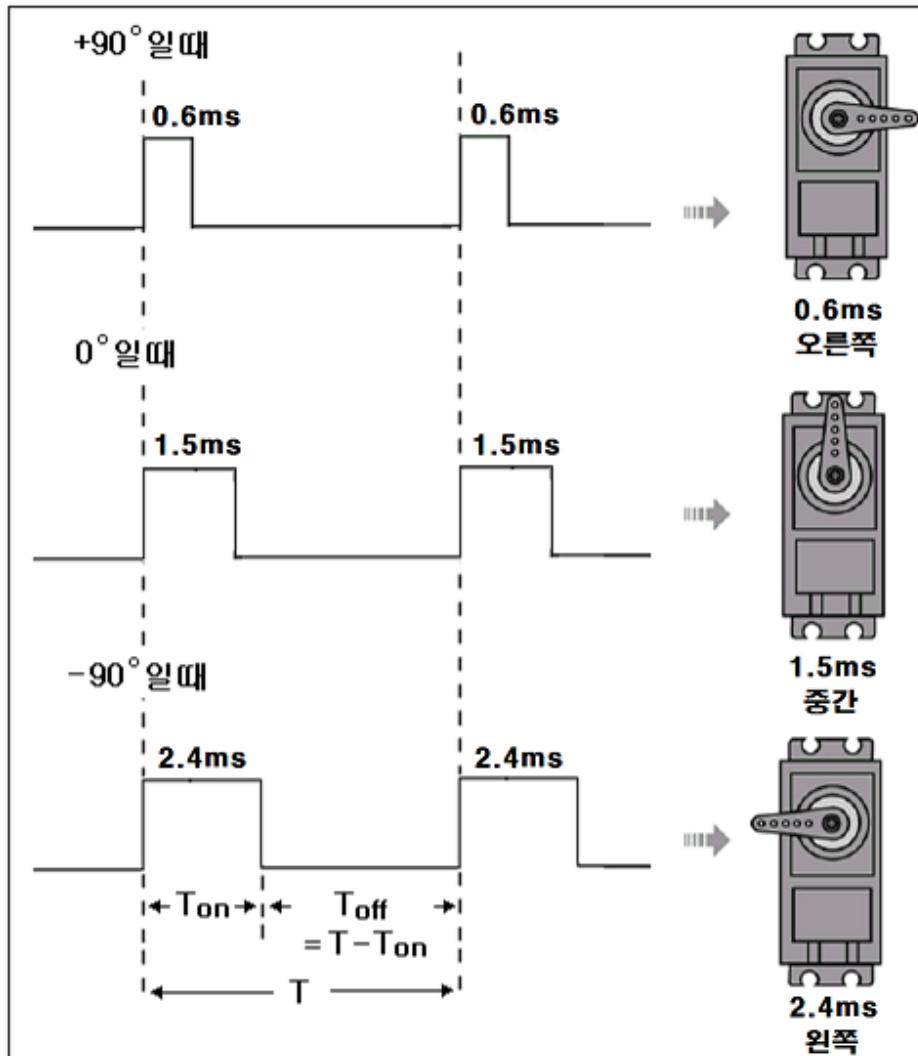
- 서보모터는 펄스 입력선으로 일정주기의 펄스신호를 주면 모터가 0~180도 사이에서 동작

GND	VCC	PWM 신호 입력선
갈색	적색	주황색



서보 모터(Servo Motor)

- 서보모터 PWM의 일반적인 주기는 20[ms]
 - High 부분의 펄스 폭에 따라 결정

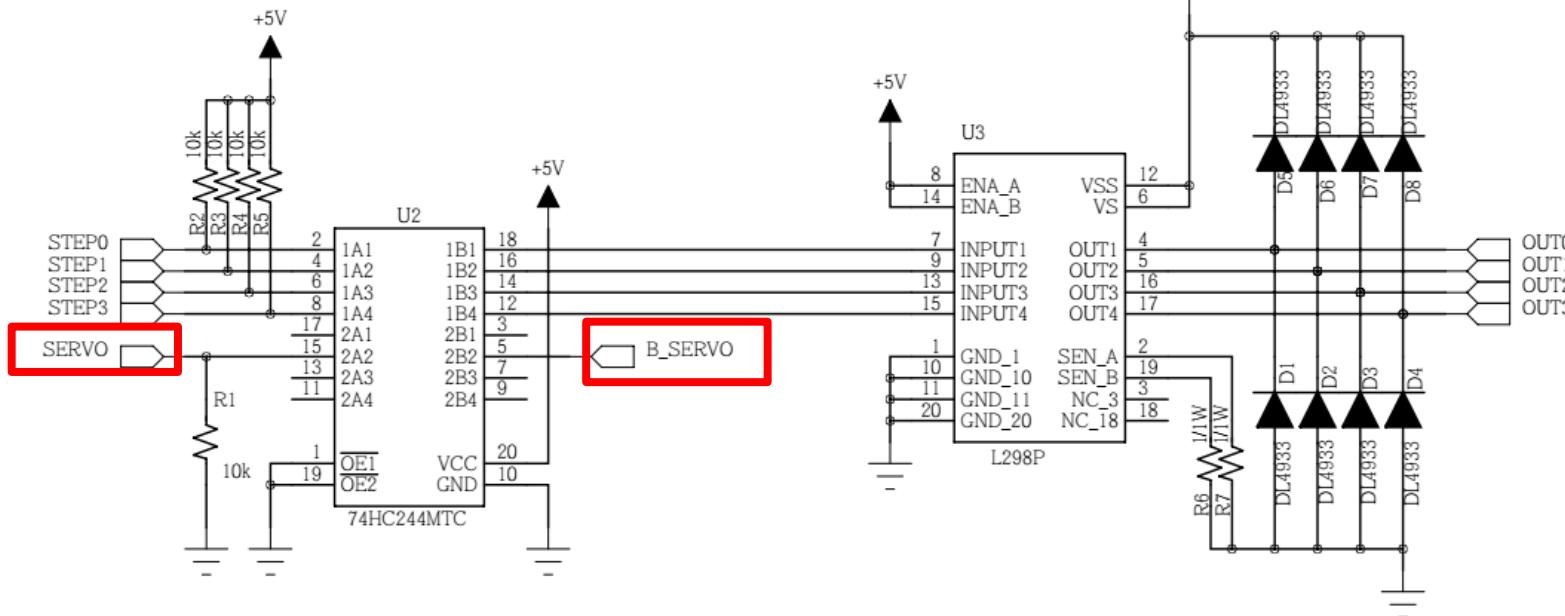


실습 1 : 서보 모터 위치 제어 1

- 실습 개요
 - STM32F405의 GPIO핀에 서보 모터의 제어신호를 연결하여, 서보 모터의 각도를 제어
 - 시간 지연 함수를 사용하여 서보의 위치를 영점(0° 위치)에 오도록 함
- 실습 목표
 - 서보 모터의 동작원리를 이해
 - STM32F405의 GPIO를 이용한 서보 모터 구동 방법 습득

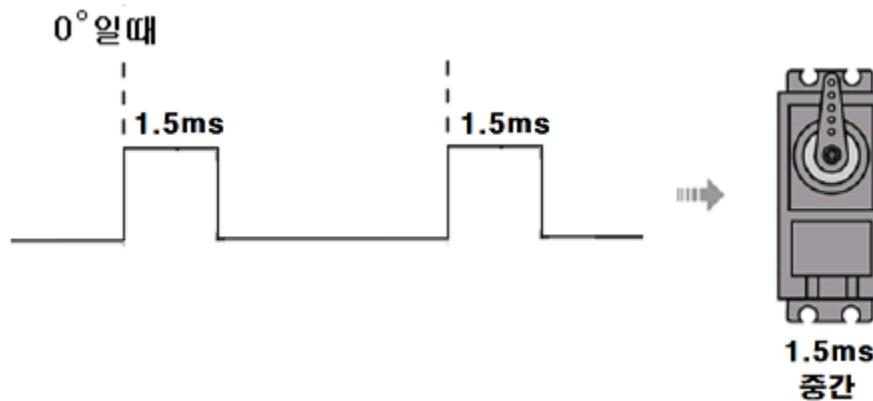
실습 1 : 서보 모터 위치 제어 1

- 사용 모듈
 - 서보 모터 모듈의 회로



실습 1 : 서보 모터 위치 제어 1

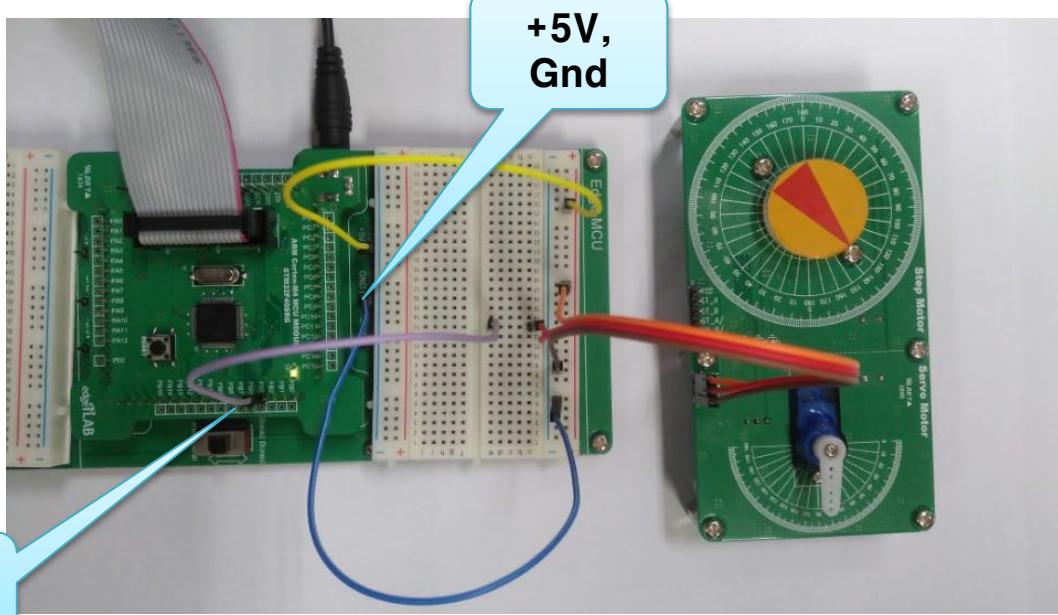
- 구동 프로그램 : 사전지식
 - GPIO 사용
 - B_SERVO → PORTB(PB5)
 - 시간 지연 함수 사용
 - 서보 모터의 제어 펄스는 주기가 20ms이며, High 부분이 1.5ms



실습 1 : 서보 모터 위치 제어 1

● 실습 준비

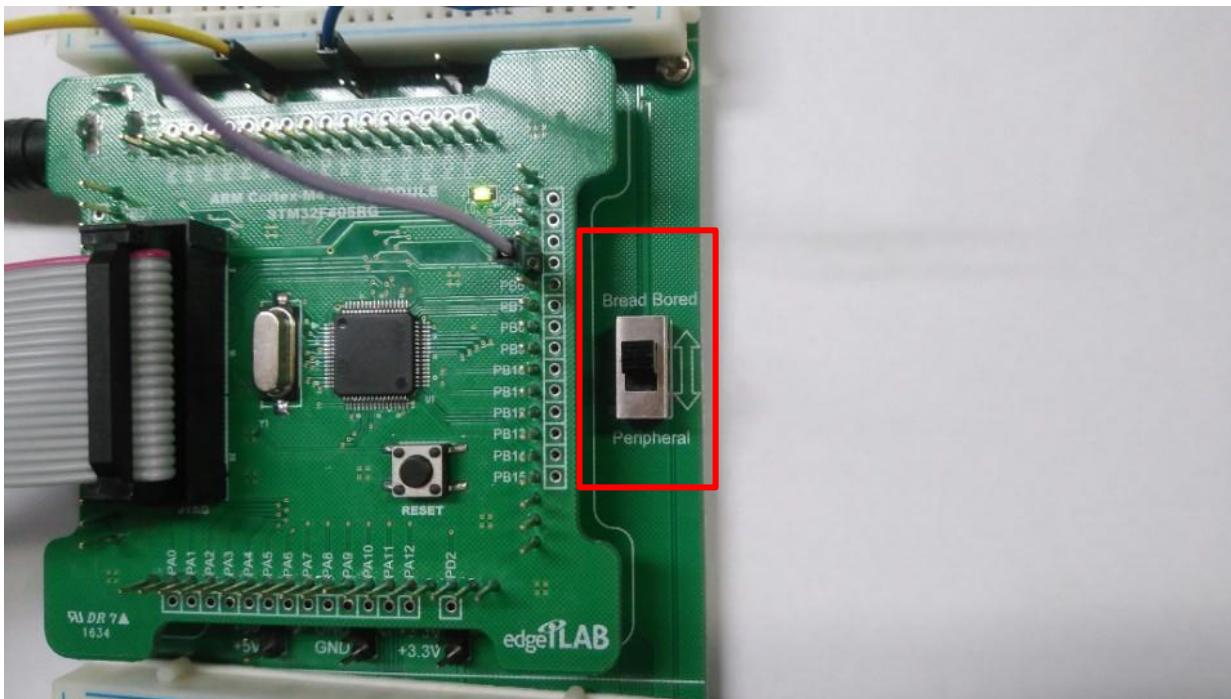
- 사용보드를 다음 그림과 같이 연결
 - Edge-MCU보드의 **+5V** → Bread보드 (+) → Step 모터 보드의 **VCC**
 - Edge-MCU보드의 **PB5** → Bread보드 IC영역 → Edge-Peri 보드의 **PWM**
 - Edge-MCU보드의 GND → Bread보드 (-) → Edge-Peri 보드의 GND



실습 1 : 서보 모터 위치 제어 1

- 실습 준비

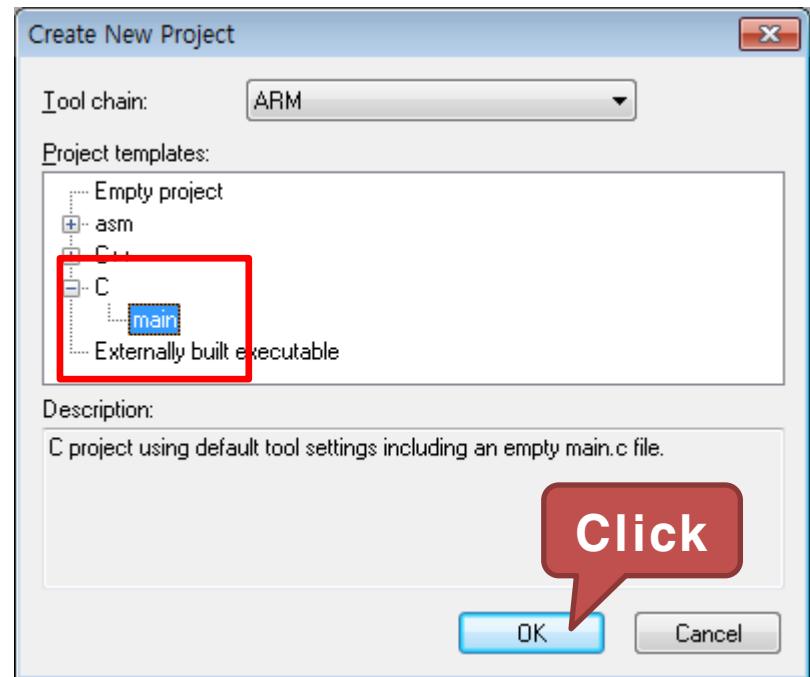
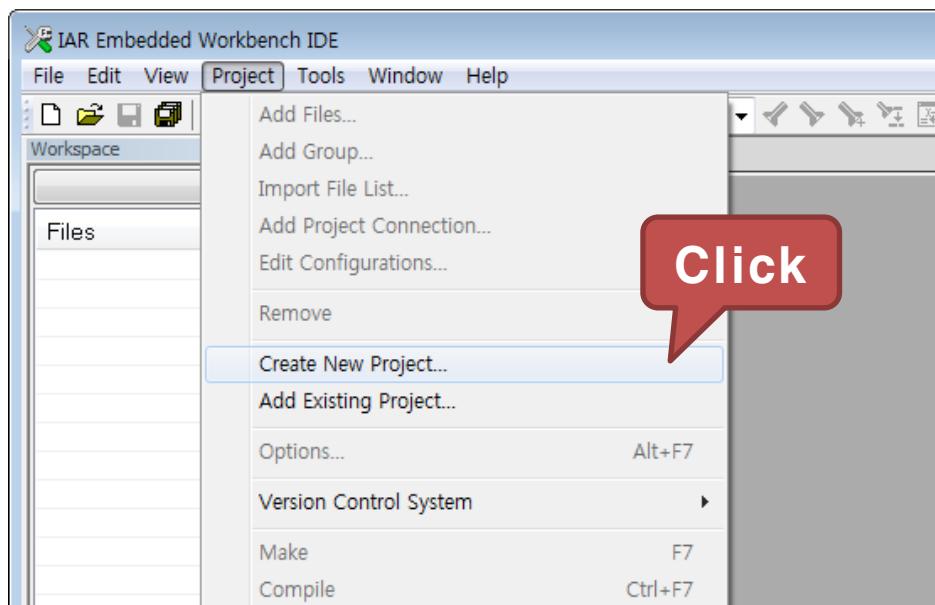
- Edge-MCU보드 오른쪽의 스위치를 Bread Board로 올림



실습 1 : 서보 모터 위치 제어 1

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch13” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “servo1”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 1 : 서보 모터 위치 제어 1

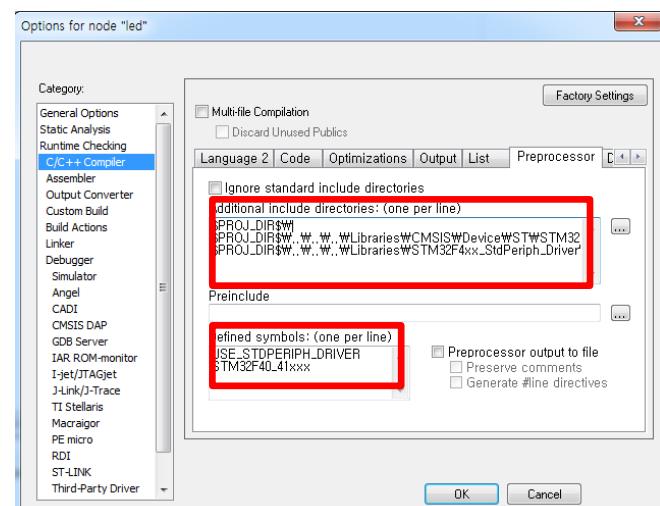
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch13\servo1	system_stm32f4xx.c
Cortex_Example\Projects\ch13\servo1	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 1 : 서보 모터 위치 제어 1

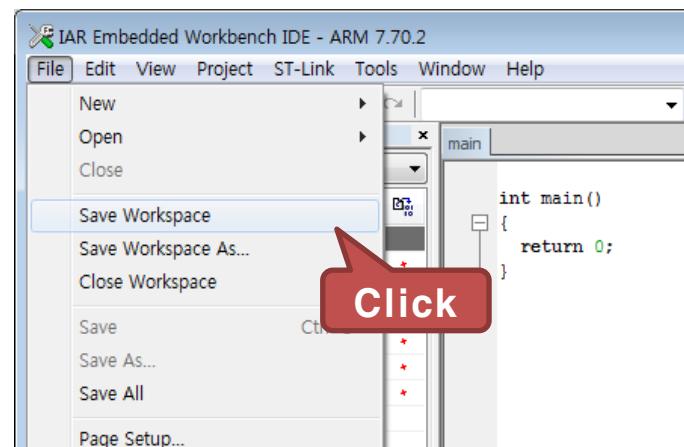
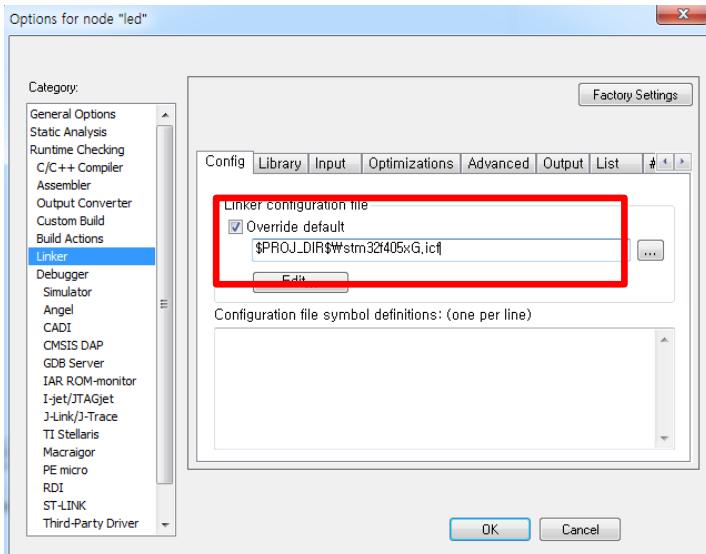
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 1 : 서보 모터 위치 제어 1

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 1 : 서보 모터 위치 제어 1

- 구동 프로그램
 - main.c 코드 작성

```
#include "stm32f4xx.h"

// delay 함수
static void Delay_us(const uint32_t Count) {
    __IO uint32_t index = 0;
    for(index = (16 * Count); index != 0; index--);
}

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
```

실습 1 : 서보 모터 위치 제어 1

- 구동 프로그램

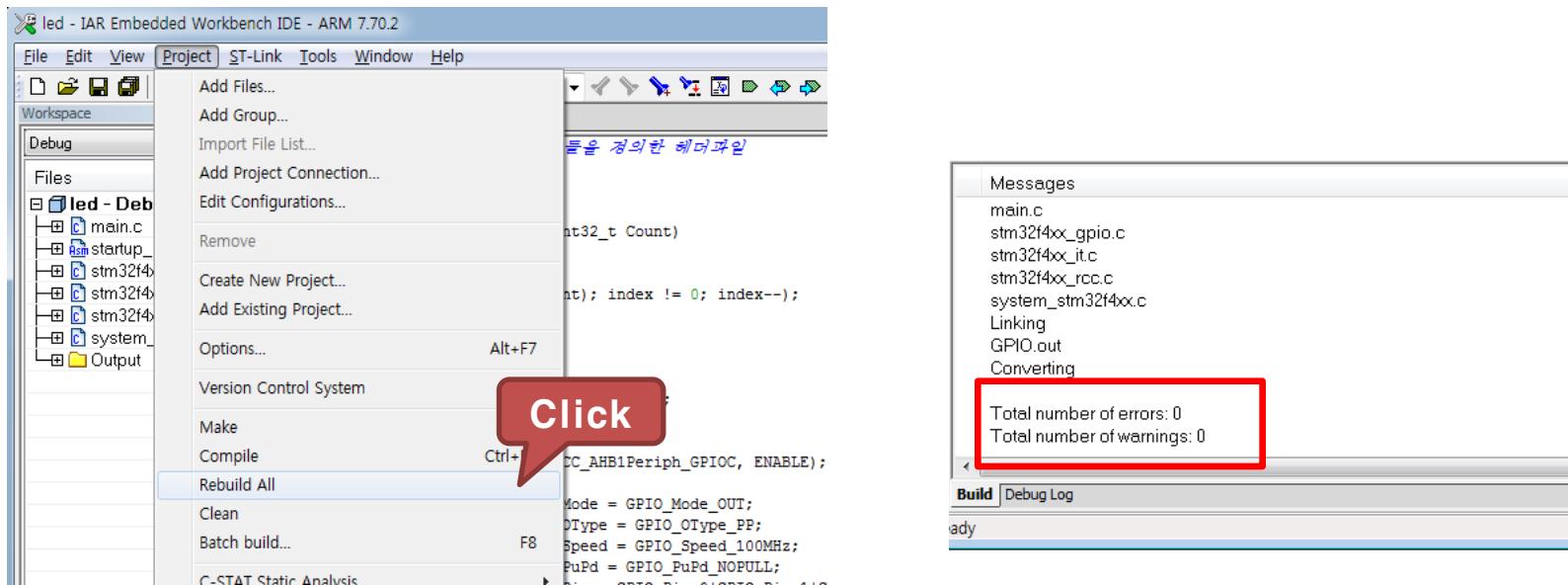
- main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;  
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;      // SERVO  
GPIO_Init(GPIOB, &GPIO_InitStructure);  
  
while(1) {  
    // PWM 주기를 20[ms]  
    GPIO_SetBits(GPIOB, GPIO_Pin_5);  
    Delay_us(1500);        // ON Time 1.5[ms]  
    GPIO_ResetBits(GPIOB, GPIO_Pin_5);  
    Delay_us(18500);       // OFF Time 18.5[ms]  
}  
}
```

실습 1 : 서보 모터 위치 제어 1

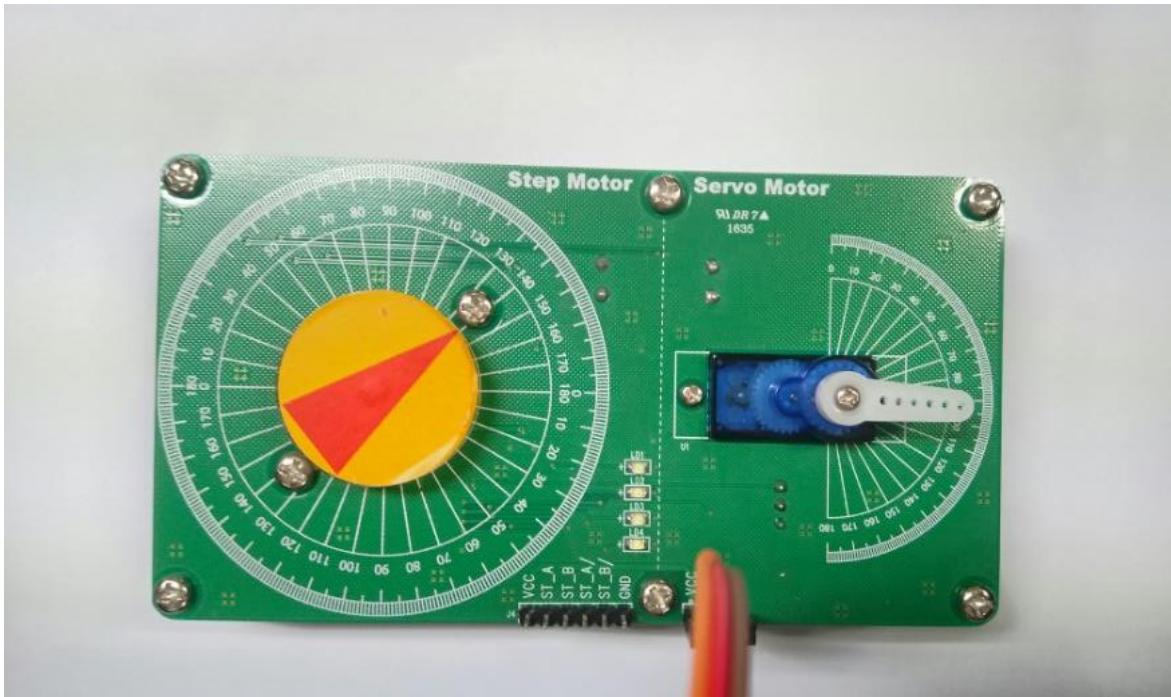
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 step1Phase 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 1 : 서보 모터 위치 제어 1

- 실행 결과
 - 서보모터는 초기 상태인 0도로 이동



실습 2 : 서보 모터 위치 제어 2

- 실습 개요
 - STM32F405의 GPIO핀에 서보 모터의 제어 신호를 연결하여, 서보 모터의 각도를 제어
 - 시간 지연 함수를 사용하여 서보의 위치를 $+90^\circ \rightarrow 0^\circ \rightarrow -90^\circ \rightarrow$ 를 왕복하는 프로그램을 구현
- 실습 목표
 - 서보 모터의 동작원리를 이해
 - STM32F405의 GPIO를 이용한 서보 모터 구동 방법 습득

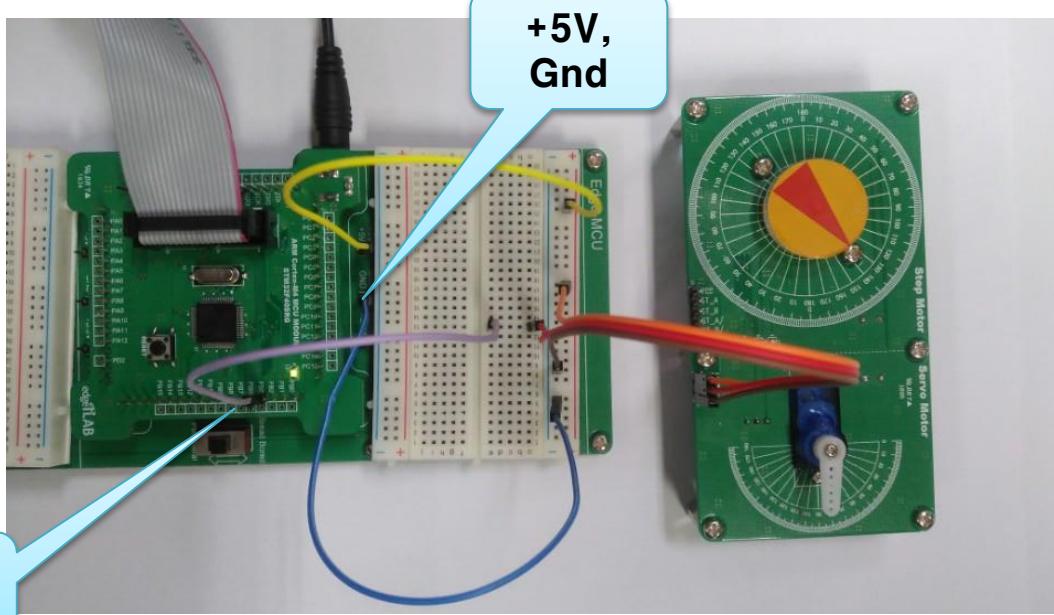
실습 2 : 서보 모터 위치 제어 2

- 구동 프로그램 : 사전지식
 - GPIO 사용
 - B_SERVO → PORTB(PB5)
 - 시간 지연 함수 사용
 - 서보 모터의 제어 펄스는 주기가 20ms
 - $+90^\circ$ 일때 High 부분이 0.6ms
 - 0° 일때 High 부분이 1.5ms
 - -90° 일때 High 부분이 2.4ms

실습 2 : 서보 모터 위치 제어 2

● 실습 준비

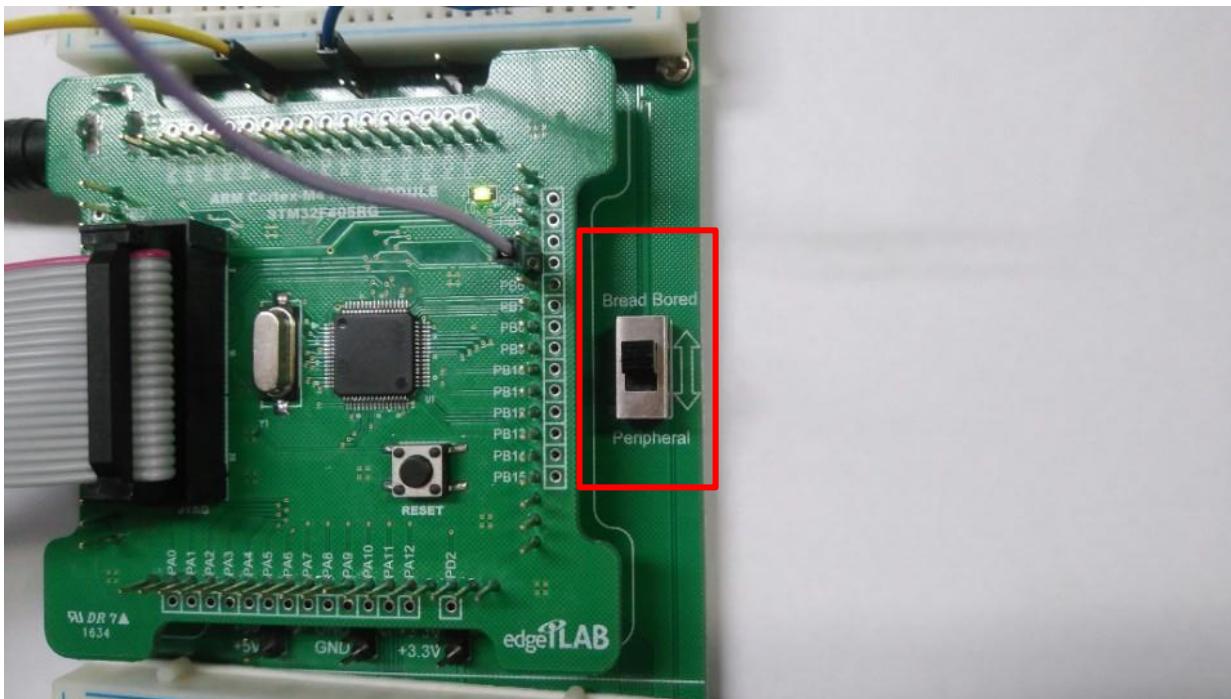
- 사용보드를 다음 그림과 같이 연결
 - Edge-MCU보드의 **+5V** → Bread보드 (+) → Step 모터 보드의 **VCC**
 - Edge-MCU보드의 **PB5** → Bread보드 IC영역 → Edge-Peri 보드의 **PWM**
 - Edge-MCU보드의 GND → Bread보드 (-) → Edge-Peri 보드의 GND



실습 2 : 서보 모터 위치 제어 2

● 실습 준비

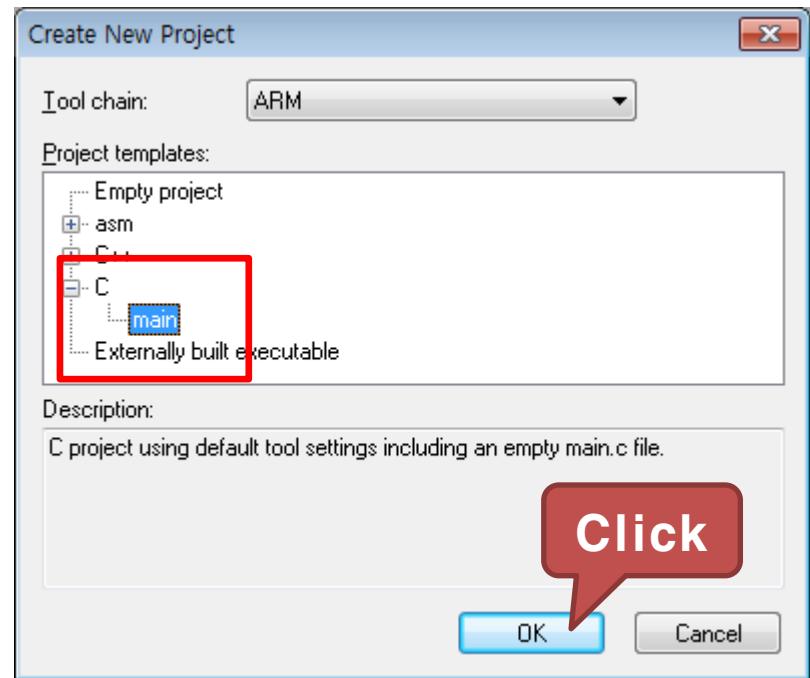
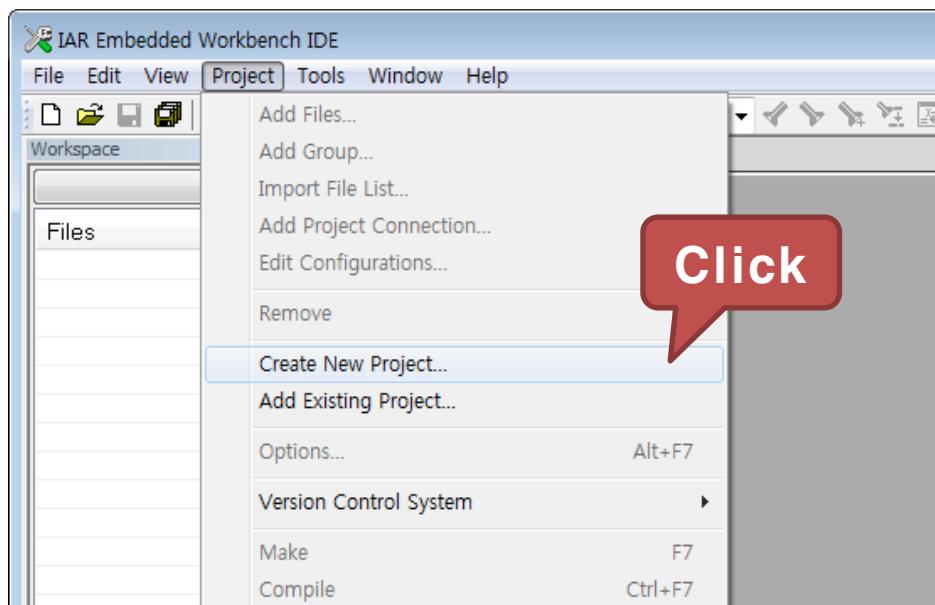
- Edge-MCU보드 오른쪽의 스위치를 Bread Board로 올림



실습 2 : 서보 모터 위치 제어 2

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch13” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “servo2”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 2 : 서보 모터 위치 제어 2

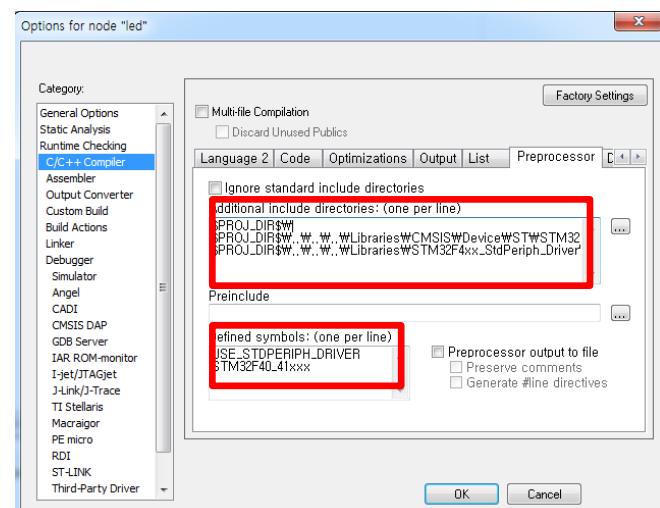
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch13\servo2	system_stm32f4xx.c
Cortex_Example\Projects\ch13\servo2	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c

실습 2 : 서보 모터 위치 제어 2

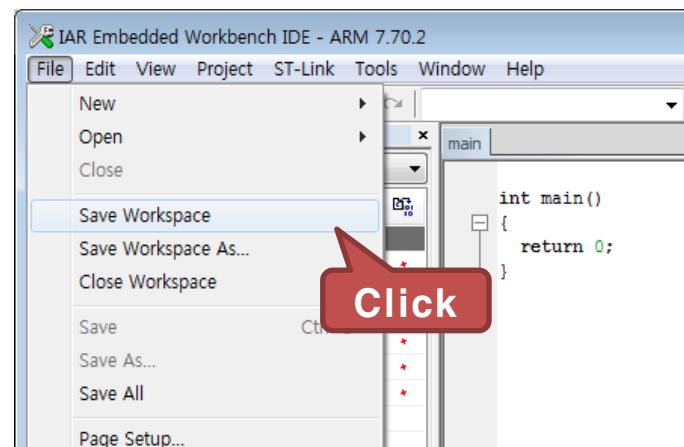
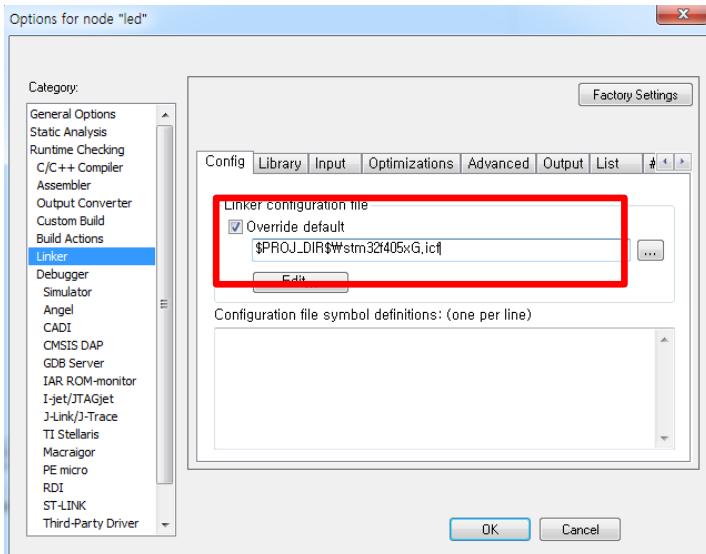
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 2 : 서보 모터 위치 제어 2

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 2 : 서보 모터 위치 제어 2

- 구동 프로그램
 - main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay_us(const uint32_t Count) {      // delay 함수
    __IO uint32_t index = 0;
    for(index = (16 * Count); index != 0; index--);
}

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    int i;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
```

실습 2 : 서보 모터 위치 제어 2

- 구동 프로그램

- main.c 코드 작성

```
//...  
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;  
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;          // SERVO  
GPIO_Init(GPIOB, &GPIO_InitStructure);  
  
while(1) {  
    for(i = 0; i < 50 ; i++ ) { // +90도 동작  
        GPIO_SetBits(GPIOB, GPIO_Pin_5);  
        Delay_us(600);      // 0.6[ms]  
  
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);  
        Delay_us(19400);   // 19.4[ms]  
    }  
}
```

실습 2 : 서보 모터 위치 제어 2

- 구동 프로그램

- main.c 코드 작성

```
for(i = 0; i < 50 ; i++ ) { // 0도 동작
    GPIO_SetBits(GPIOB, GPIO_Pin_5);
    Delay_us(1500);           // 1.5[ms]

    GPIO_ResetBits(GPIOB, GPIO_Pin_5);
    Delay_us(18500);          // 18.5[ms]
}

for(i = 0; i < 50 ; i++ ) { // -90도 동작
    GPIO_SetBits(GPIOB, GPIO_Pin_5);
    Delay_us(2400);           // 2.4[ms]

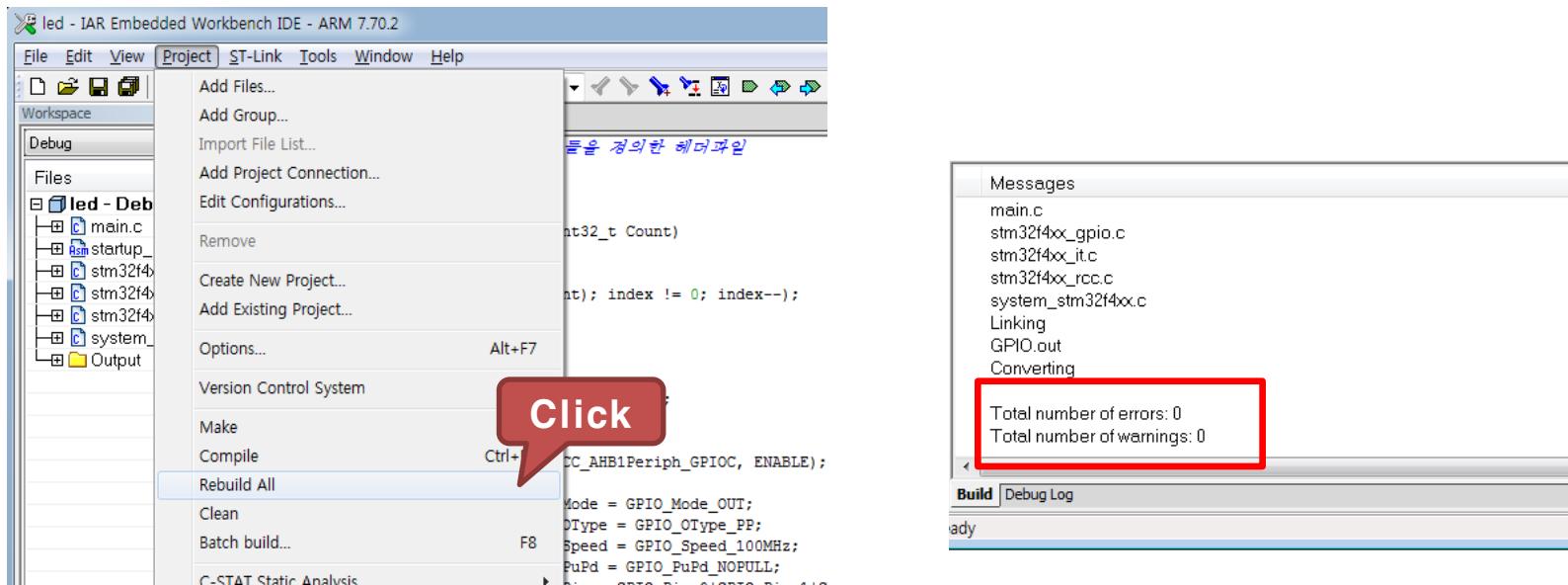
    GPIO_ResetBits(GPIOB, GPIO_Pin_5);
    Delay_us(17600);          // 17.6[ms]
}

}
```

실습 2 : 서보 모터 위치 제어 2

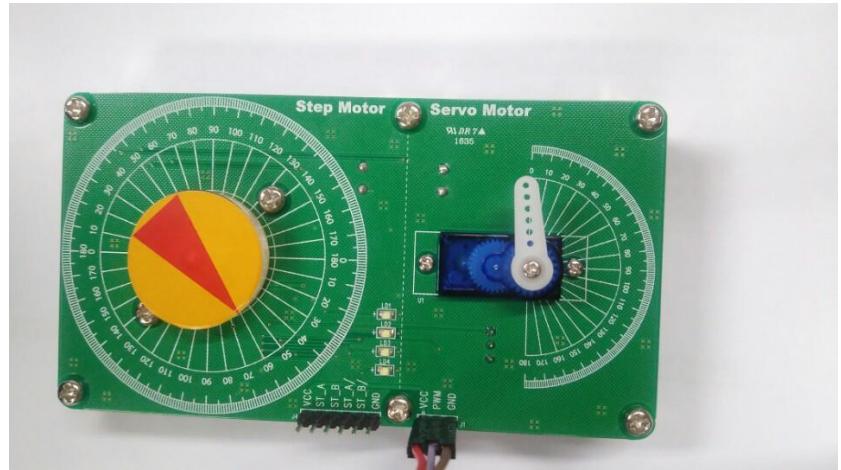
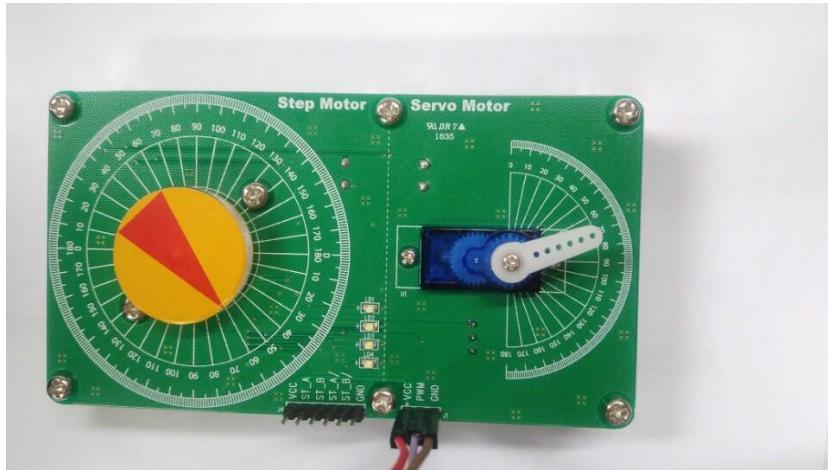
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 servo2 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 2 : 서보 모터 위치 제어 2

- 실행 결과
 - 서보 모터는 $+90^\circ \rightarrow 0^\circ \rightarrow -90^\circ$ 로 반복해서 이동



실습 3 : 서보 모터 위치 제어 3

- 실습 개요
 - STM32F405의 GPIO핀에 서보 모터의 제어신호를 연결하여, 서보 모터의 각도를 제어
 - 타이머를 사용하여 서보 제어 신호를 만들고, 서보의 위치를 $+90^\circ \rightarrow 0^\circ \rightarrow -90^\circ$ 를 왕복하는 프로그램을 구현
- 실습 목표
 - 서보 모터의 동작원리를 이해
 - STM32F405의 GPIO를 이용한 서보 모터 구동 방법 습득
 - STM32F405의 타이머 사용법 습득

실습 3 : 서보 모터 위치 제어 3

- 구동 프로그램 : 사전지식
 - GPIO 사용
 - B_SERVO → PORTB(PB5)
 - 타이머 사용
 - 서보 모터의 제어 펄스는 주기가 20ms
 - 타이머/카운터 0이 약 0.1[ms] 마다 오버플로가 발생하기 위해서는 먼저 타이머/카운터 0의 소스 클록을 몇 분주로 할 것인지 TCCR0 레지스터를 설정
 - CLK/8 분주로 설정

	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
TCCR0	0	0	0	0	0	0	1	0

- 카운트 TCNT0의 초기 값을 설정
 - $TCNT0 = 0xff(\text{카운트 최댓값}) + 1 - \text{클록 카운트 값}(0xB8) = 0x48$
 - $TCNT0 = 0x48$ 로 설정하면 $256(0x100) - 72(0x48) = 184(0xB8)$ 즉, 184번 클록을 카운트 한 다음에 인터럽트가 발생

실습 3 : 서보 모터 위치 제어 3

- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
 - GPIO 사용
 - B_SERVO → PORTB(PB5)
 - 사용할 타이머/카운터 결정
 - 여기서는 타이머/카운터 4를 사용
 - 동작 모드 결정
 - 여기서는 일반 동작 모드 사용
 - 카운트 방향을 업카운트(0 부터 TIMx-ARR까지)로 선택
 - TIM4_CR1레지스터의 DIR을 0으로 클리어

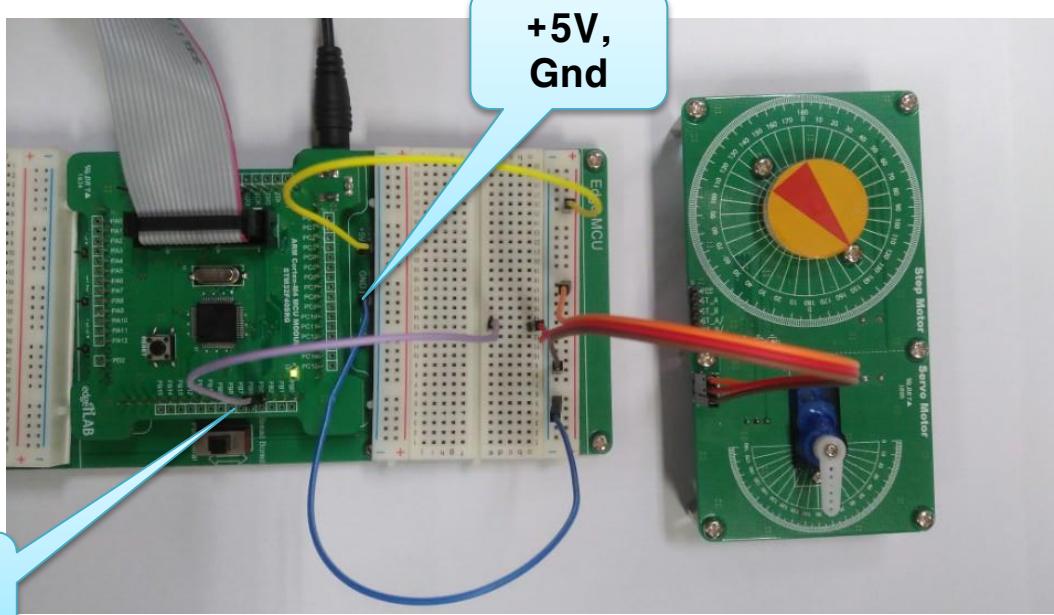
실습 3 : 서보 모터 위치 제어 3

- 구동 프로그램 : 타이머/카운터 를 동작시키기 위한 사전 지식
 - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
 - 여기서는 클럭(84MHz)을 사용
 - 타이머 4의 클럭은 메인 클럭(168Mhz)의 $\frac{1}{2}$ 을 사용
 - 샘플링 클럭의 분주비는 1:1로 설정
 - 타이머 클럭 프리스케일러는 83으로 설정(TIM4_CR1/CKD:00, TIM4_PSC: 83)
 - 84MHz 클럭의 1분주비와 83 프리스케일러로 나누어 지므로 1MHz(1us)로 동작
 - 타이머 주기 결정
 - 100us를 타이머 주기로 결정
 - TIM4_ARR레지스터의 값을 100으로 설정
 - 타이머 주기 * 카운터 수의 공식에 의해 100s마다 Update(여기서는 오버플로우)가 발생하여 100us마다 인터럽트가 발생

실습 3 : 서보 모터 위치 제어 3

● 실습 준비

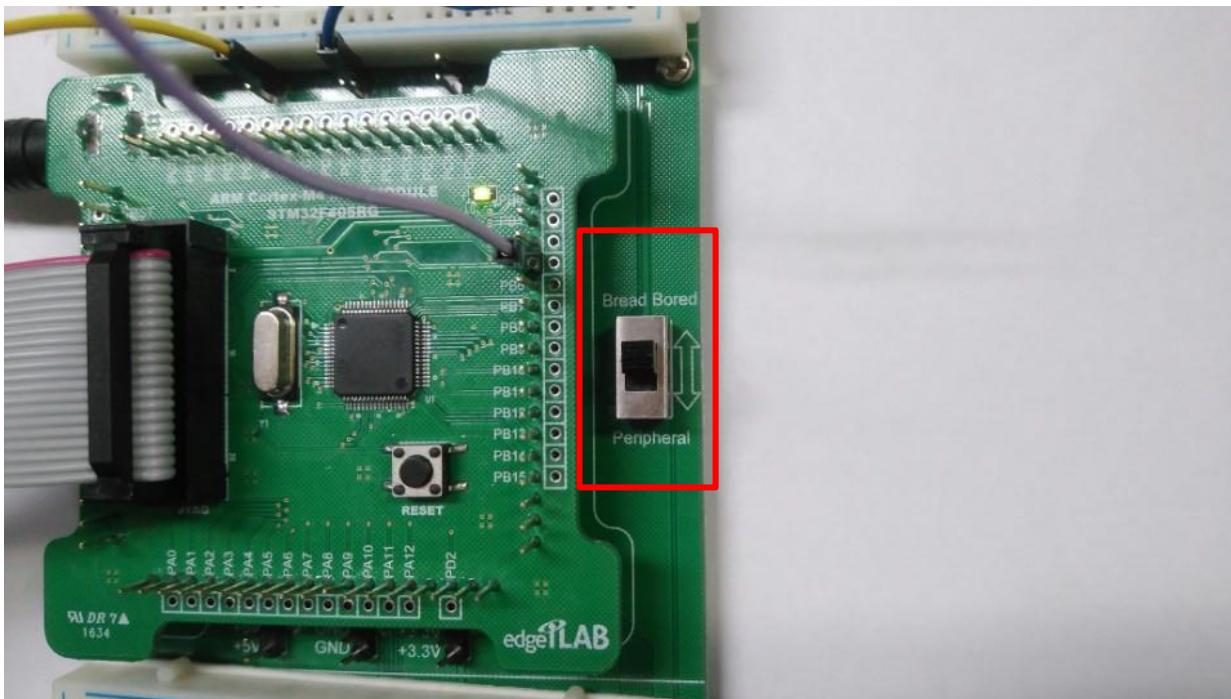
- 사용보드를 다음 그림과 같이 연결
 - Edge-MCU보드의 **+5V** → Bread보드 (+) → Step 모터 보드의 **VCC**
 - Edge-MCU보드의 **PB5** → Bread보드 IC영역 → Edge-Peri 보드의 **PWM**
 - Edge-MCU보드의 GND → Bread보드 (-) → Edge-Peri 보드의 GND



실습 3 : 서보 모터 위치 제어 3

● 실습 준비

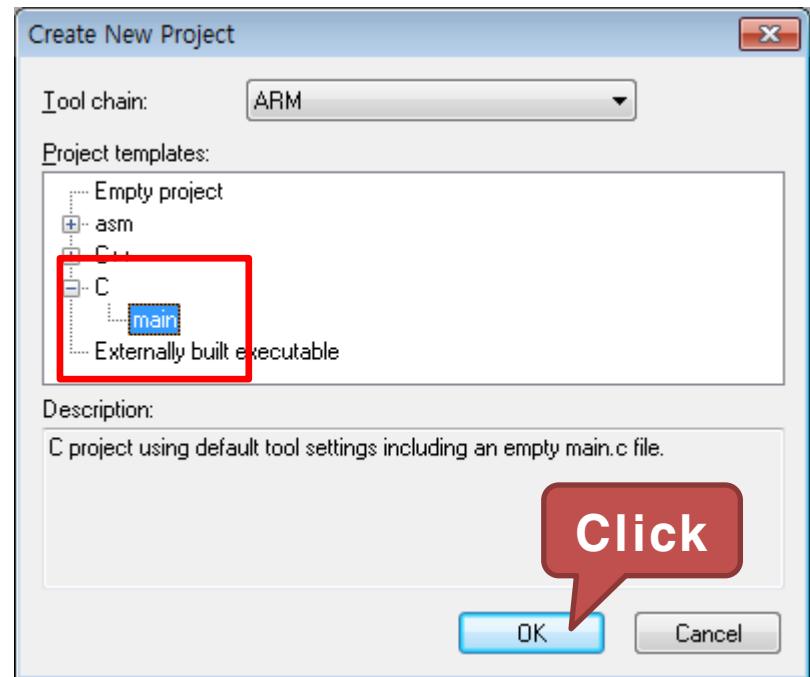
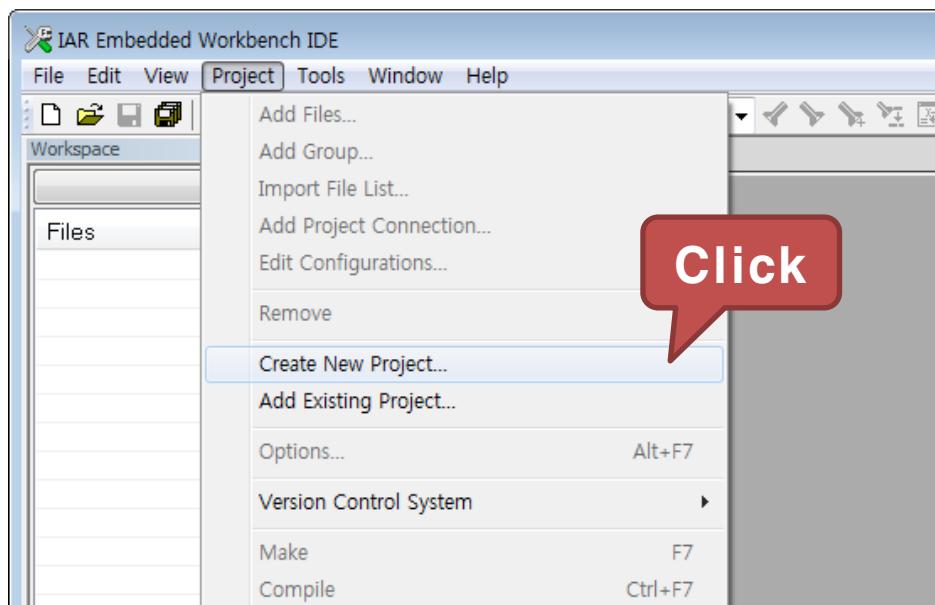
- Edge-MCU보드 오른쪽의 스위치를 Bread Board로 올림



실습 3 : 서보 모터 위치 제어 3

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch13” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “servo3”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 3 : 서보 모터 위치 제어 3

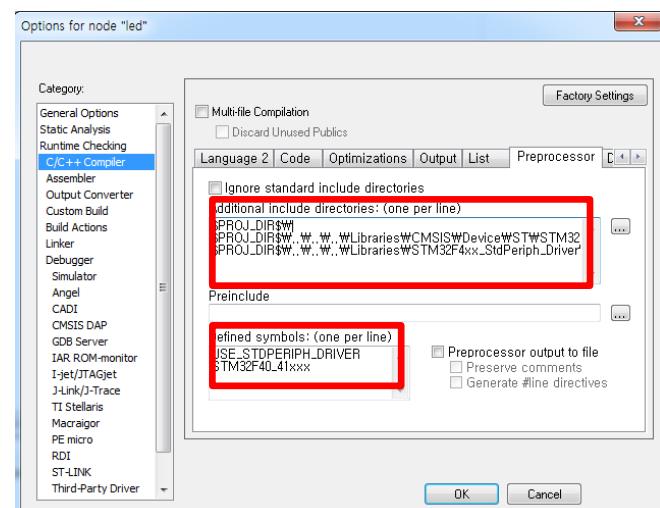
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 "Project" 탭에서 "Add Files..."를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch13\servo3	system_stm32f4xx.c
Cortex_Example\Projects\ch13\servo3	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_misc.c

실습 3 : 서보 모터 위치 제어 3

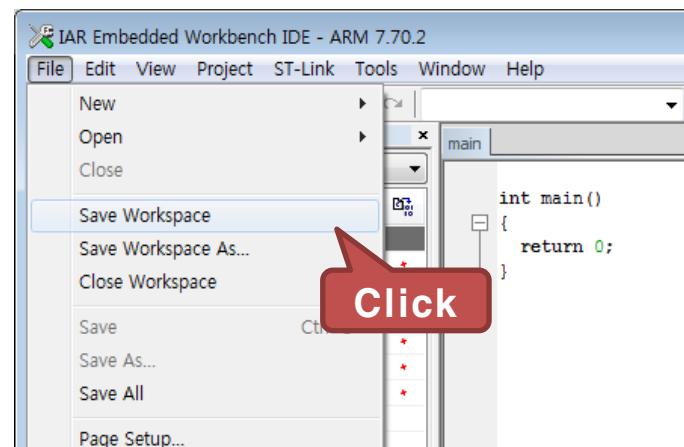
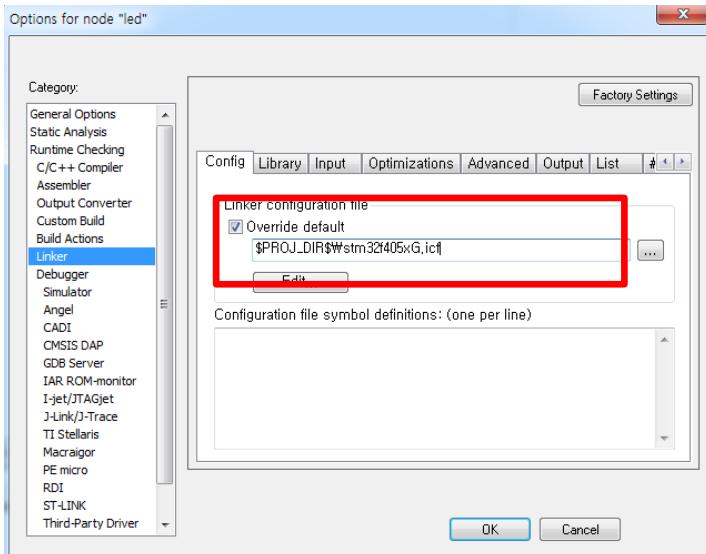
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 3 : 서보 모터 위치 제어 3

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 3 : 서보 모터 위치 제어 3

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

int t_cnt = 0;
int AngleCount = 0;

void Servo(int Angle) {
    if (Angle == -90 )
        AngleCount = 23;      // PWM ON Time 2.3[ms]
    else if (Angle == 0)
        AngleCount = 15;      // PWM ON Time 1.5[ms]
    else if(Angle == 90)
        AngleCount = 7;       // PWM ON Time 0.7[ms]
}
```

실습 3 : 서보 모터 위치 제어 3

- 구동 프로그램
 - main.c 코드 작성

```
void TIM4_IRQHandler(void) {
    if(TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET) {
        TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
        t_cnt++;
        if(t_cnt <= AngleCount)
            GPIO_SetBits(GPIOB, GPIO_Pin_5); // PWM ON Time
        else
            GPIO_ResetBits(GPIOB, GPIO_Pin_5); // PWM OFF Time
        if(t_cnt >= 200) t_cnt = 0; // 20[ms] 주기 PWM
    }
}
```

실습 3 : 서보 모터 위치 제어 3

- 구동 프로그램
 - main.c 코드 작성

```
int main(void) {
    GPIO_InitTypeDef    GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    NVIC_InitTypeDef    NVIC_InitStructure;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5; // SERVO
    GPIO_Init(GPIOB, &GPIO_InitStructure);
```

실습 3 : 서보 모터 위치 제어 3

- 구동 프로그램

- main.c 코드 작성

```
// 인터럽트 enable 및 Priority 설정  
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);  
NVIC_InitStructure.NVIC IRQChannel = TIM4 IRQn;  
NVIC_InitStructure.NVIC IRQChannelPreemptionPriority = 0x01;  
NVIC_InitStructure.NVIC IRQChannelSubPriority = 0x01;  
NVIC_InitStructure.NVIC IRQChannelCmd = ENABLE;  
NVIC_Init(&NVIC_InitStructure);  
  
// (168Mhz/2)/840 = 1MHz(1us)  
TIM_TimeBaseStructure.TIM_Prescaler = 84-1;  
// 1us x 100 = 100us  
TIM_TimeBaseStructure.TIM_Period = 100-1;  
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;  
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  
TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
```

실습 3 : 서보 모터 위치 제어 3

- 구동 프로그램

- main.c 코드 작성

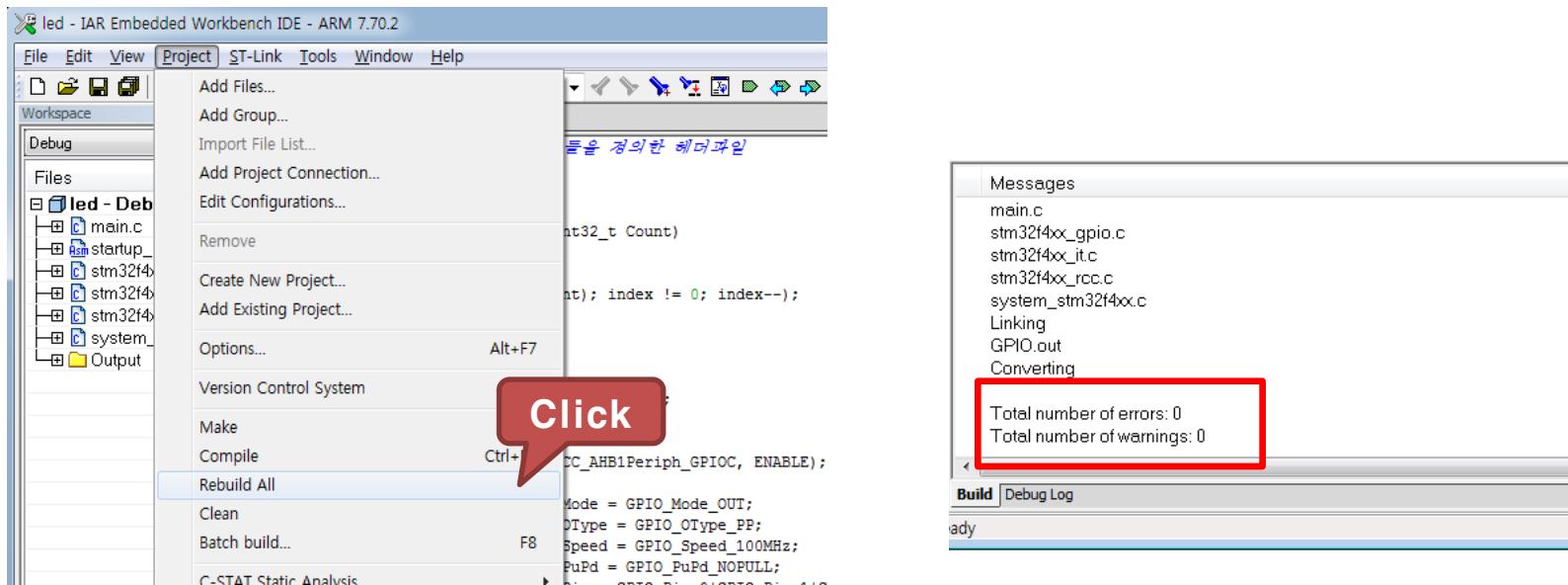
```
//...
// 타이머4를 동작
TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);
TIM_Cmd(TIM4, ENABLE);

while(1) {
    // delay 1초를 주는 이유는 서보가 움직일 충분한 시간을 주기 위함
    Servo(0);
    Delay(1000);
    Servo(90);
    Delay(1000);
    Servo(-90);
    Delay(1000);
}
}
```

실습 3 : 서보 모터 위치 제어 3

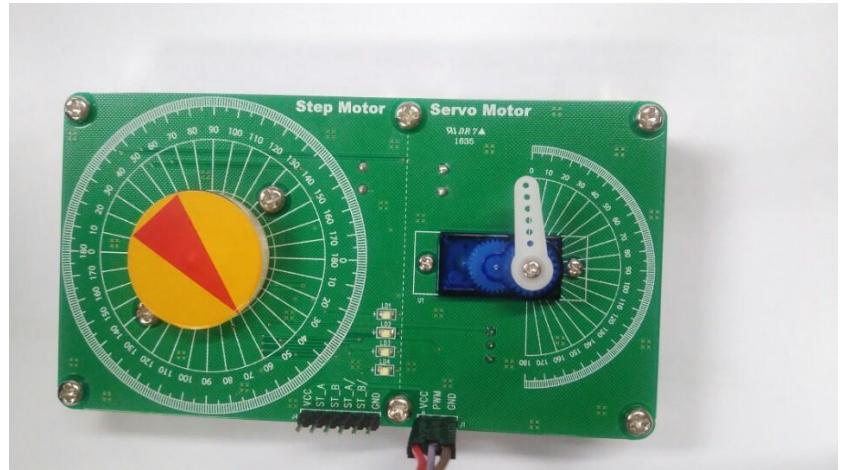
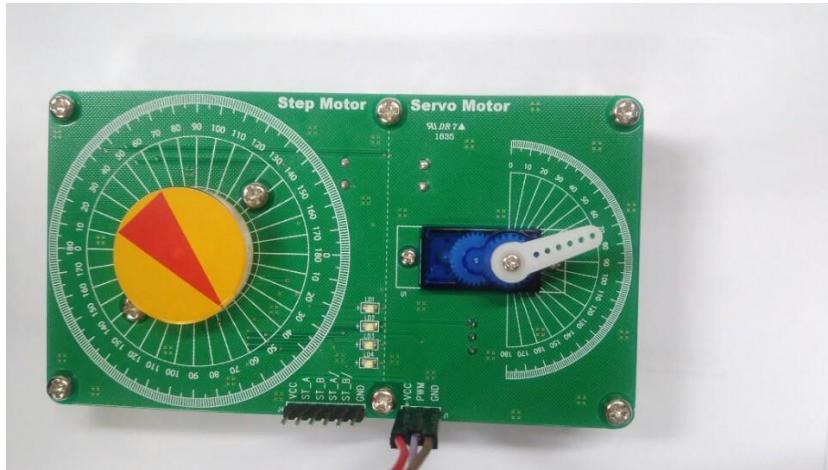
● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 servo3 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 3 : 서보 모터 위치 제어 3

- 실행 결과
 - 서보 모터는 $+90^\circ \rightarrow 0^\circ \rightarrow -90^\circ$ 로 반복해서 이동



실습 4 : 서보 모터 위치 제어 4

- 실습 개요
 - STM32F405의 GPIO핀에 서보 모터의 제어신호를 연결하여, 서보 모터의 각도를 제어
 - 타이머의 PWM 기능을 사용하여 서보 제어 신호를 만들고, 서보의 위치를 $+90^\circ \rightarrow 0^\circ \rightarrow -90^\circ \rightarrow$ 를 왕복하는 프로그램을 구현
- 실습 목표
 - 서보 모터의 동작원리를 이해
 - STM32F405의 GPIO를 이용한 서보 모터 구동 방법 습득
 - STM32F405의 타이머의 PWM 기능 사용법 습득

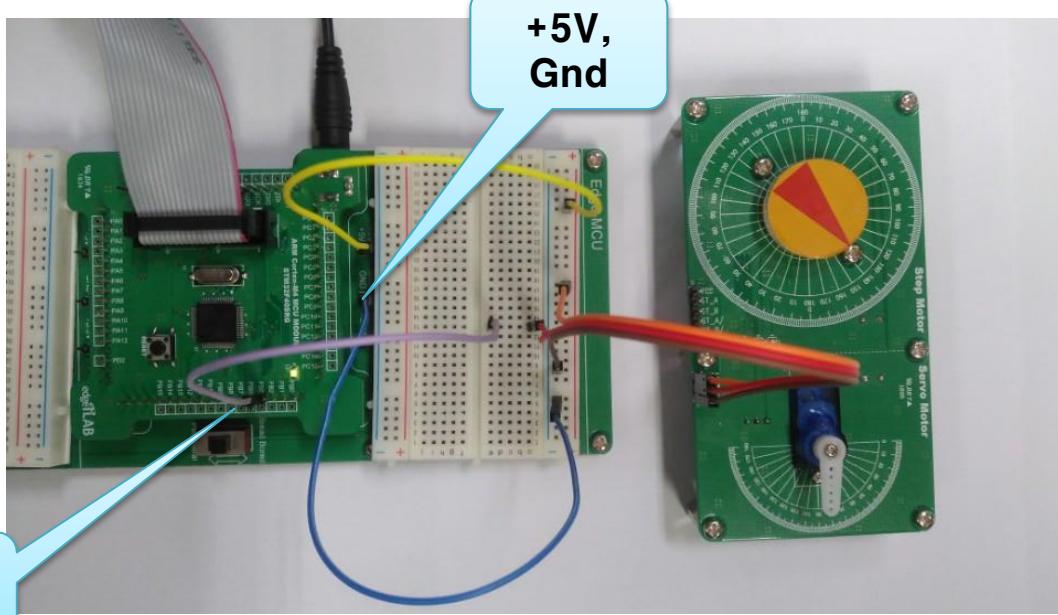
실습 4 : 서보 모터 위치 제어 4

- 구동 프로그램 : PWM 동작 모드 설정
 - 사용할 타이머/카운터 결정
 - 여기서는 타이머/카운터 3을 사용
 - 동작 모드 결정
 - 여기서는 PWM 모드 사용
 - 카운트 방향을 업카운트(0 부터 TIMx-ARR까지)로 선택
 - TIM3_CR1레지스터의 DIR을 0으로 클리어
 - 타이머 클럭 결정(클럭소스 및 프리스케일러 결정)
 - 여기서는 클럭(84MHz)을 사용
 - 타이머 2의 클럭은 메인 클럭(168Mhz)의 $\frac{1}{2}$ 을 사용
 - 샘플링 클럭의 분주비는 1:1로 설정
 - 타이머 클럭 프리스케일러는 83으로 설정(TIM3_CR1/CKD:00, TIM3_PSC: 83)
 - 84MHz 클럭의 1분주비와 83 프리스케일러로 나누어 지므로 1MHz(1us)로 동작
 - PWM 주파수 결정
 - 20ms를 타이머 주기로 결정
 - TIM3_ARR레지스터의 값을 20000으로 설정
 - 듀티비는 TIM3_CCR1 에 1~20000 까지 입력하여 변화(TIM3_CCR1 최대값은 TIM3_ARR값 까지)

실습 4 : 서보 모터 위치 제어 4

● 실습 준비

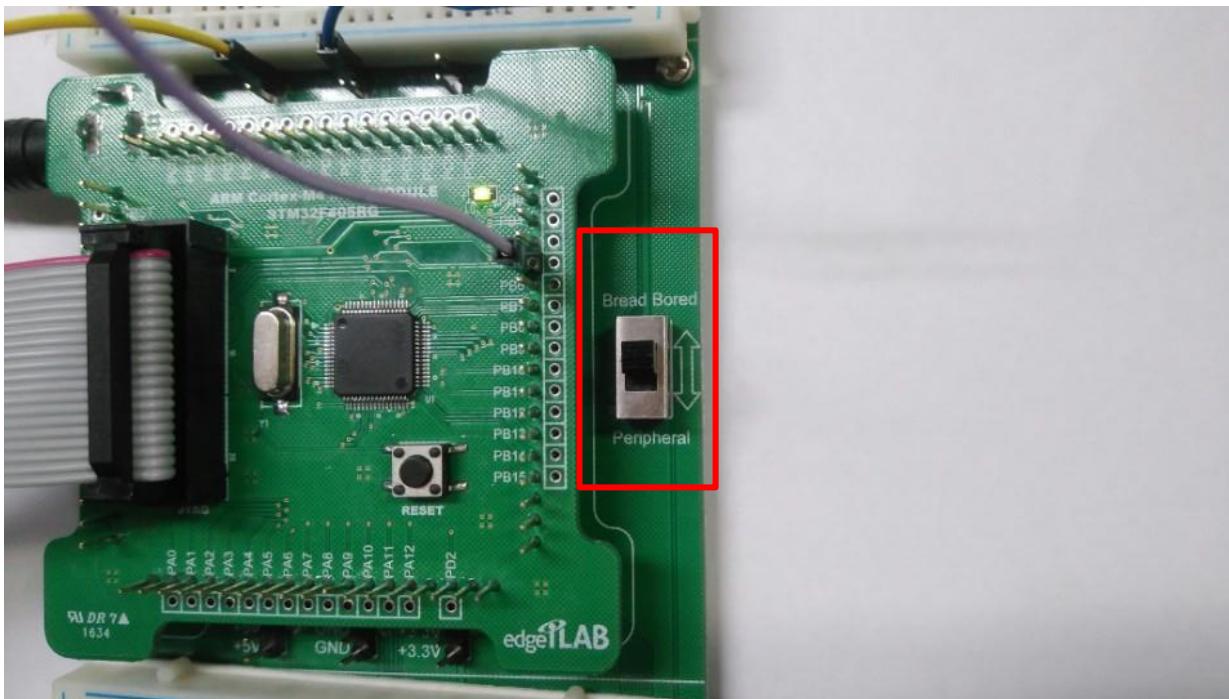
- 사용보드를 다음 그림과 같이 연결
 - Edge-MCU보드의 **+5V** → Bread보드 (+) → Step 모터 보드의 **VCC**
 - Edge-MCU보드의 **PB5** → Bread보드 IC영역 → Edge-Peri 보드의 **PWM**
 - Edge-MCU보드의 GND → Bread보드 (-) → Edge-Peri 보드의 GND



실습 4 : 서보 모터 위치 제어 4

● 실습 준비

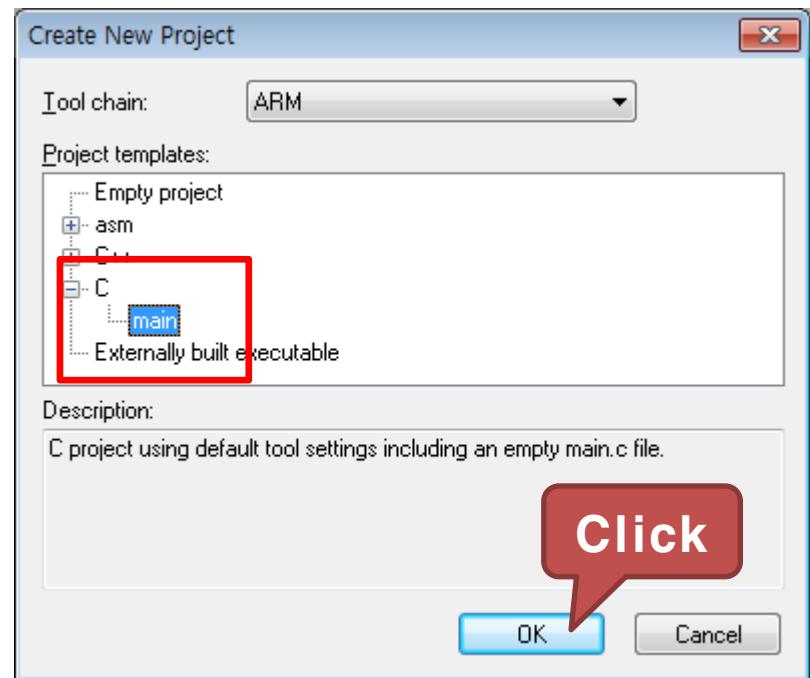
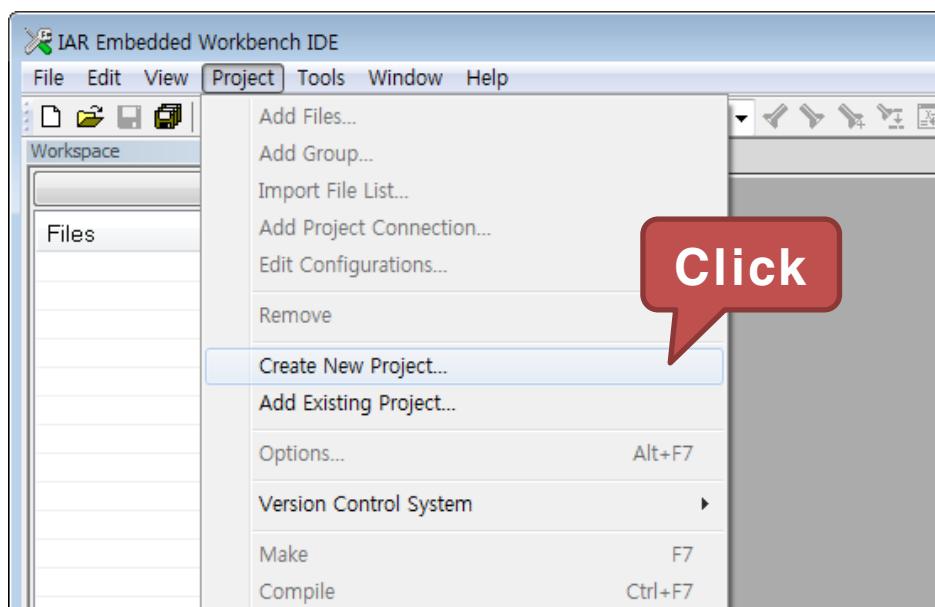
- Edge-MCU보드 오른쪽의 스위치를 Bread Board로 올림



실습 4 : 서보 모터 위치 제어 4

● 새 프로젝트 만들기

- “Projects” 폴더 안의 “New Project” 폴더를 복사해서 “ch13” 폴더에 붙여넣기
- “New Project” 폴더가 복사된 뒤, 폴더의 이름을 “servo4”로 변경
- EWARM 실행 → 새 프로젝트 생성 → tool chain ARM, 템플릿 : C – main 선택



실습 4 : 서보 모터 위치 제어 4

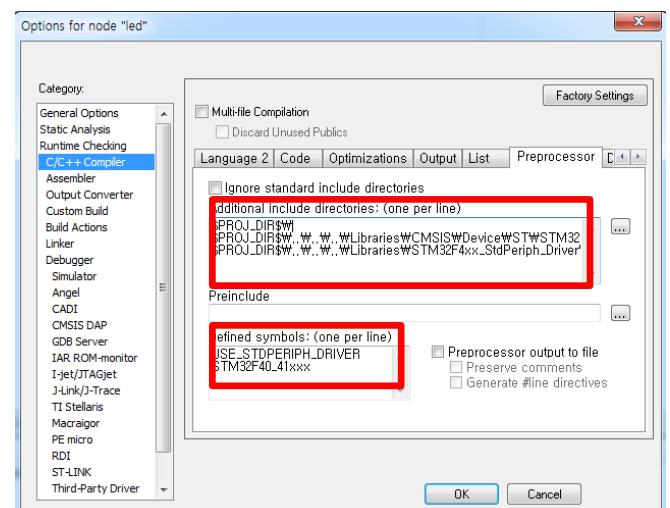
- 라이브러리 파일 추가
 - GPIO를 사용하려면 새로 만든 프로젝트에 다음의 라이브러리 파일 추가
 - 새 프로젝트가 생성이 되면 사용할 라이브러리 파일들을 등록해야 함
 - 다음과 같이 “Project” 탭에서 “Add Files...”를 눌러 라이브러리 파일들을 등록

경로	파일
Cortex_Example\Projects\ch13\servo4	system_stm32f4xx.c
Cortex_Example\Projects\ch13\servo4	stm32f4xx_it.c
Cortex_Example\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\iar	startup_stm32f40_41xxx.s
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_gpio.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_rcc.c
Cortex_Example\Libraries\STM32F4xx_StdPeriph_Driver\src	stm32f4xx_tim.c

실습 4 : 서보 모터 위치 제어 4

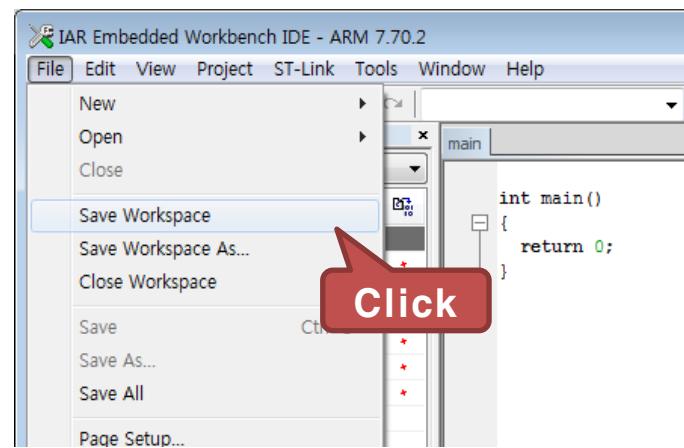
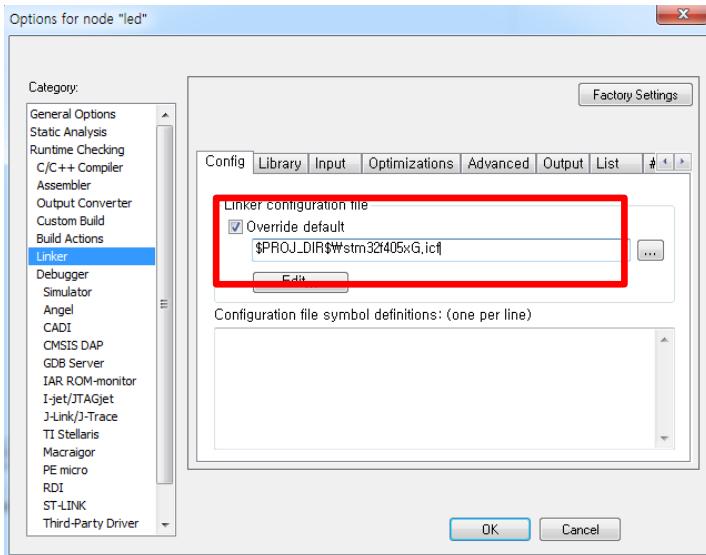
- 프로젝트 옵션 설정

- “Projects” 탭에서 “Options...” 선택(Alt + F7)
- General Options
 - “Target” 탭 → Device : ST STM32F405RG 선택
 - “Library Configuration” 탭 → Library : Full 선택, Use CMSIS 체크
- C/C++ Compiler
 - “Preprocessor” 탭에서 다음과 같이 입력
 - Additional include directories : (one per line)
 - \$PROJ_DIR\$\Libraries\CMSIS\Device\STM32F4xx\Include
 - \$PROJ_DIR\$\Libraries\STM32F4xx_StdPeriph_Driver\inc
 - Defined symbols : (one per line)
 - USE_STDPERIPH_DRIVER
 - STM32F40_41xxx



실습 4 : 서보 모터 위치 제어 4

- 프로젝트 옵션 설정
 - Output Converter
 - “Output” 탭 → Generate additional output 체크, Intel extended 선택
 - Linker
 - “Config” 탭 → Override default 체크, \$PROJ_DIR\$\stm32f405xG.icf 입력
 - Debugger
 - “Setup” 탭 → Driver : ST-LINK 선택
 - “Download” 탭 → Verify download, Use flash loader(s) 체크
- 프로젝트 저장
 - “File” 탭 → “Save Workspace” 클릭해서 저장



실습 4 : 서보 모터 위치 제어 4

- 구동 프로그램

- main.c 코드 작성

```
#include "stm32f4xx.h"

static void Delay(const uint32_t Count) {           // delay 함수
    __IO uint32_t index = 0;
    for(index = (16800 * Count); index != 0; index--);
}

void Move_ServoMotor(signed char degree){
    unsigned int _duty = 0;
    // +90 = 2400 , 0 = 1500 , -90 = 600
    _duty = 1500+degree*(-10);
    // TIM_TimeBaseStructure.TIM_Period 1 당 every 1us
    TIM_Cmd(TIM3, DISABLE);
    TIM_SetCompare2(TIM3, _duty);
    TIM_Cmd(TIM3, ENABLE);
    Delay(1000);
}
```

실습 4 : 서보 모터 위치 제어 4

- 구동 프로그램

- main.c 코드 작성

```
int main(void) {  
    GPIO_InitTypeDef      GPIO_InitStructure;  
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;  
    TIM_OCInitTypeDef    TIM_OCInitStructure;  
  
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);  
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);  
  
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;  
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;  
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;  
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;  
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;      // SERVO  
    GPIO_Init(GPIOB, &GPIO_InitStructure);  
  
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource5, GPIO_AF_TIM3);
```

실습 4 : 서보 모터 위치 제어 4

- 구동 프로그램
 - main.c 코드 작성

```
//...  
TIM_TimeBaseStructure.TIM_Prescaler = 84-1;  
// (168Mhz/2)/840 = 1MHz(1us)  
TIM_TimeBaseStructure.TIM_Period = 20000-1;  
//1us x 20000 = 20ms  
  
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;  
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);  
  
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;  
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;  
TIM_OCInitStructure.TIM_Pulse = 0;  
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;  
TIM_OC2Init(TIM3, &TIM_OCInitStructure);
```

실습 4 : 서보 모터 위치 제어 4

- 구동 프로그램
 - main.c 코드 작성

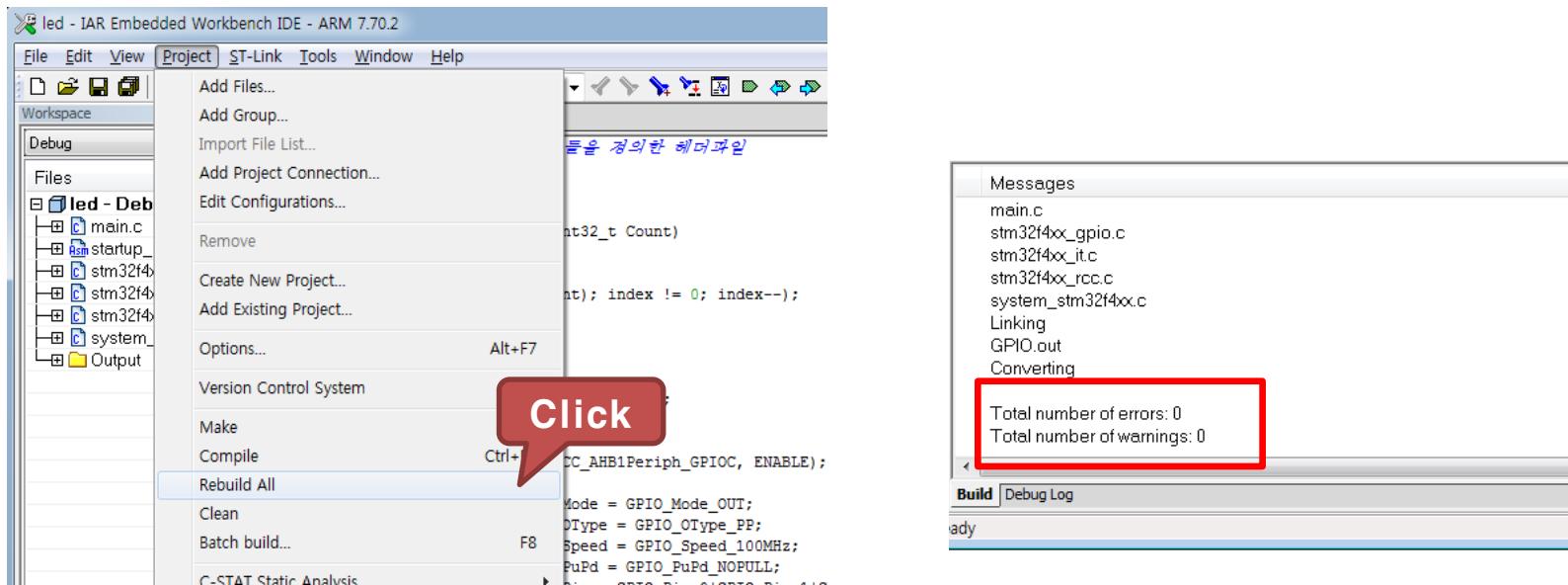
```
// 타이머3을 동작
TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable);
TIM_ARRPreloadConfig(TIM3, ENABLE);
TIM_Cmd(TIM3, ENABLE);

while(1){
    Move_ServoMotor(90);
    Move_ServoMotor(0);
    Move_ServoMotor(-90);
}
}
```

실습 4 : 서보 모터 위치 제어 4

● 프로젝트 빌드 / 다운로드

- 코드 작성이 완료되면 servo4 프로젝트 빌드
- "Project" 탭에서 "Rebuild All" 클릭
- 오류가 발생하지 않았다면, 프로젝트 다운로드
- "Project" 탭에서 "Download"의 "Download active application"을 클릭하여 빌드 한 프로젝트를 다운로드 함
- 다운로드 완료하면 MCU 모듈의 "Reset" 버튼 클릭



실습 4 : 서보 모터 위치 제어 4

● 실행 결과

- 서보 모터는 초기 상태인 0도로 움직이고 -90 -> 0 -> 90 -> 0 -> -90로 1초마다 반복적으로 동작

