

ARM Cortex M Programming: 시작하기

Lesson 01

준비

Code Composer Studio 설치

Code Composer Studio 다운로드

<http://www.ti.com/tool/ccstudio>

The screenshot shows the Texas Instruments website with the URL <http://www.ti.com/tool/ccstudio>. The page title is "Code Composer Studio (CCS) Integrated Development Environment (IDE)". The main content area includes sections for "Description", "Features", and "Download". A red arrow points from the text "클릭한다." (Click here) to the "Download" button at the bottom of the page.

Texas INSTRUMENTS

Products Applications & designs Tools & software Support & training Order Now About TI

Recommendations from TI.com: Reference designs selected for you Products of interest for you Top technical documents for you

Worldwide (in English)

Code Composer Studio (CCS) Integrated Development Environment (IDE)

(ACTIVE) CCSTUDIO

Description

Maximize your experience with Code Composer Studio v7

- Single IDE for all TI processors
- Code quality improvement
- Reduced development time

Cloud-based Development Tools

Start working with your MSP LaunchPad or SensorTag immediately with Code Composer Studio Cloud and Resource Explorer. All the software and tools you need are in the cloud.

more

Code Composer Studio™ - Integrated Development Environment

Code Composer Studio is an integrated development environment (IDE) that supports TI's Microcontroller and Embedded Processors portfolio. Code Composer Studio comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features. The intuitive IDE provides a single user interface taking you through each step of the application development flow. Familiar tools and interfaces allow users to get started faster than ever before. Code Composer Studio combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers.

Features

By Platform - Find out more about the features available for a specific processor family:

- MSP Low Power MCUs
- C2000 Real-time MCUs
- SimpleLink Wireless MCUs
- TM4x MCUs
- TMS320 & RM4 Safety MCUs
- Silera (Cortex A & ARM9) Processors
- Multicore DSP and ARM including KeyStone Processors
- F24x/C24x devices
- C3x/C4x DSPs

Code Composer Studio supports TI's broad portfolio of embedded processors. If you do not see a link for the family you are interested in above then select the one that is closest in terms of the processor cores used.

Download

- CCS latest version - Click below to download Code Composer Studio
- Additional downloads - For a complete list of downloads including previous versions please visit the [Code Composer Studio download site](#)
- Cloud Tools - Visit [dev.ti.com](#) to access TI Cloud Tools. Browse through the resources available for a device, run demo applications and even develop code using CCS Cloud

Download

클릭한다.

Code Composer Studio 다운로드

The screenshot shows a web browser window with the URL processors.wiki.ti.com. The search bar at the top contains the query "gcc". The main content area is titled "Download CCS". A sidebar on the left includes links for Main Page, All pages, All categories, Recent changes, Random page, Help, Print/export, and Toolbox.

Cloud Tools

If you are using a LaunchPad or a SensorTag you can begin working with many of these boards without downloading CCS. Visit dev.ti.com to access Cloud-based development tools. Resource Explorer provides instant access to all of the examples, documentation and libraries, CCS Cloud is a cloud-based IDE and PinMux enables you to select your peripherals and generate the pin configuration.

Download the latest CCS

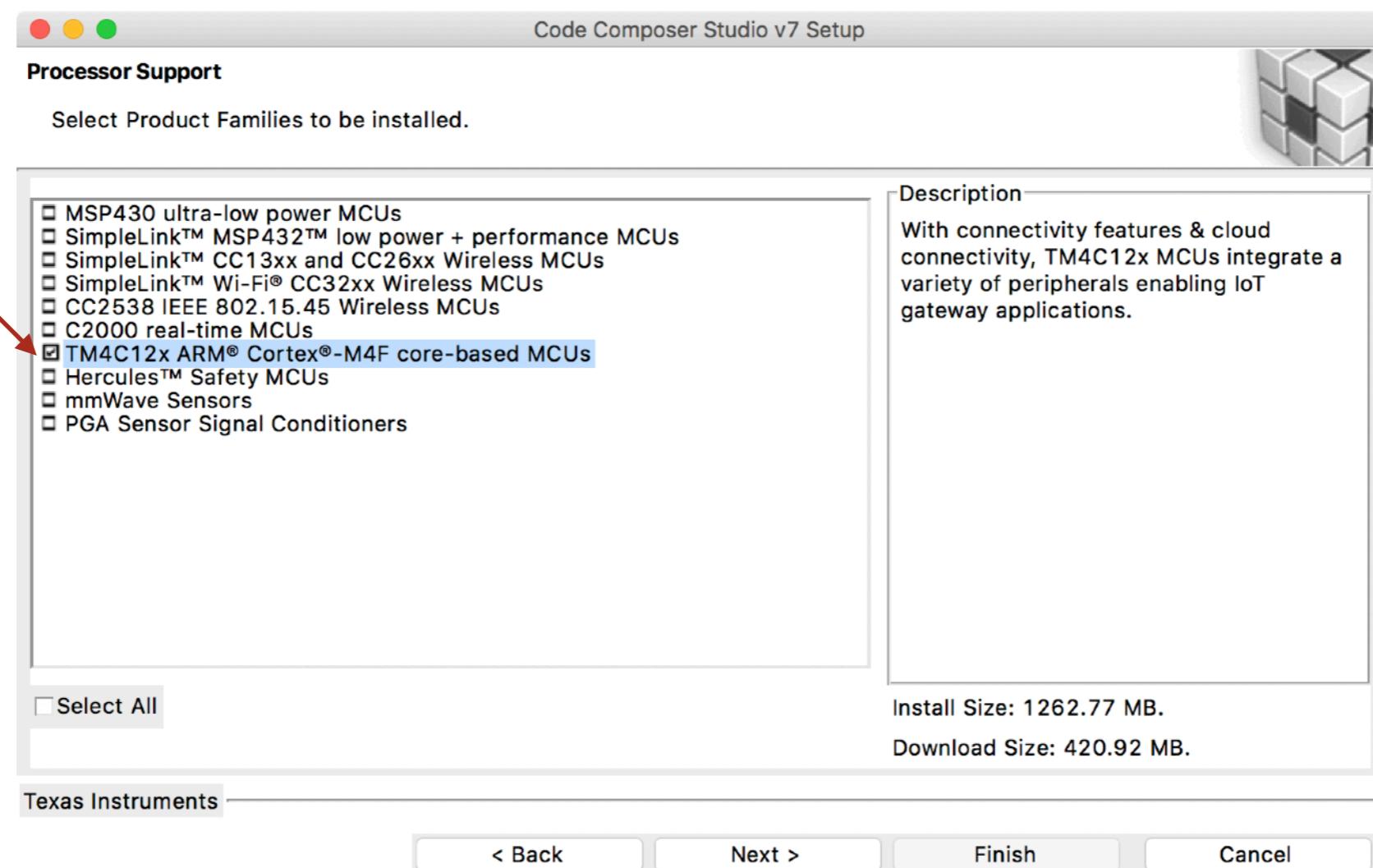
Download 7.3.0.00019	Installers (Offline installer is recommended for slow and unreliable connections)
Windows	Offline Installer Online Installer
Mac OS	Offline Installer Online Installer
Linux 64bit	Offline Installer Online Installer

A red circle highlights the "Installers" column, and a red arrow points from this circle to the text "필요한 버전을 다운로드한다." (Download the required version).

필요한 버전을 다운로드한다.

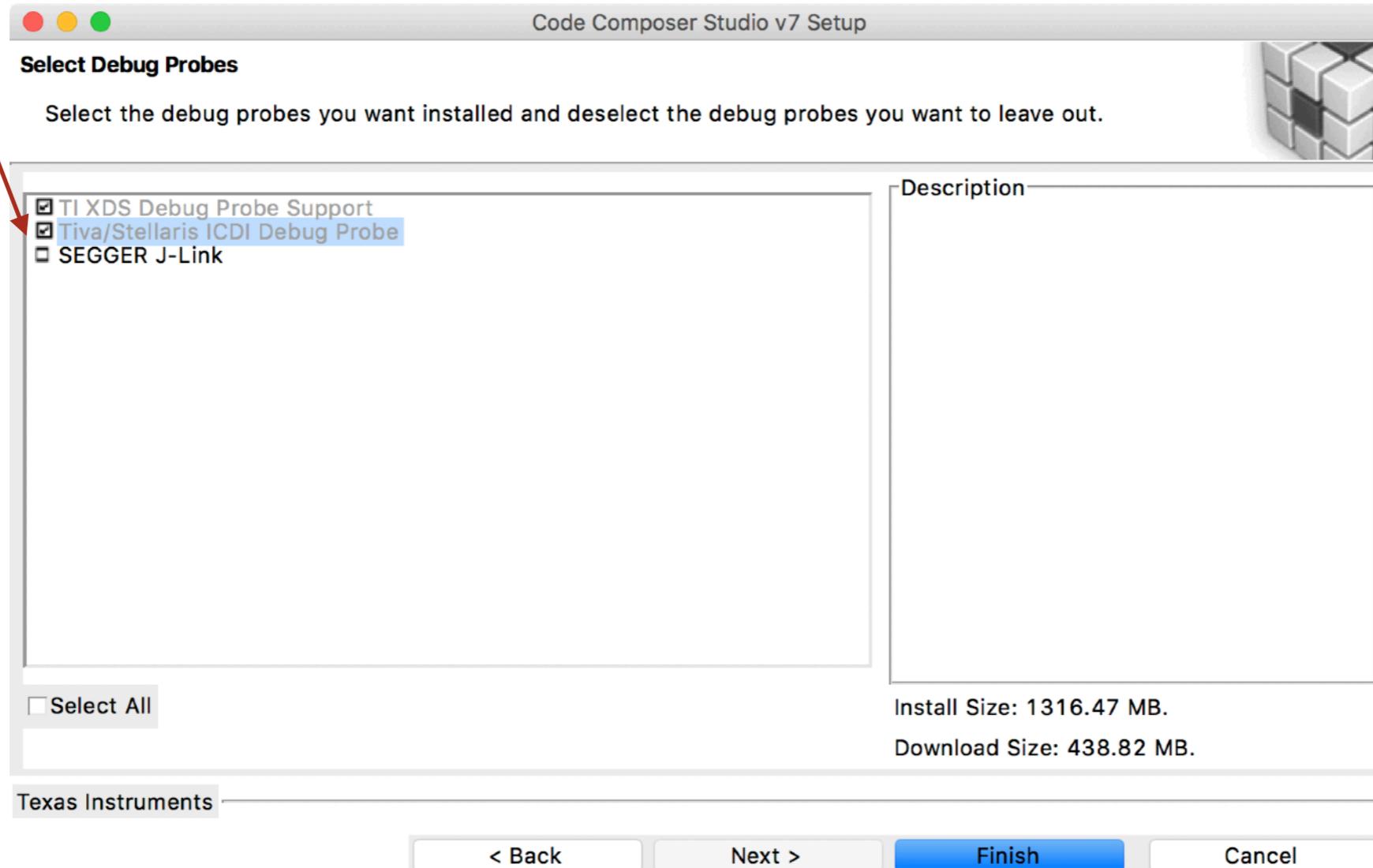
Code Composer Studio 설치

이 곳에
체크한다.



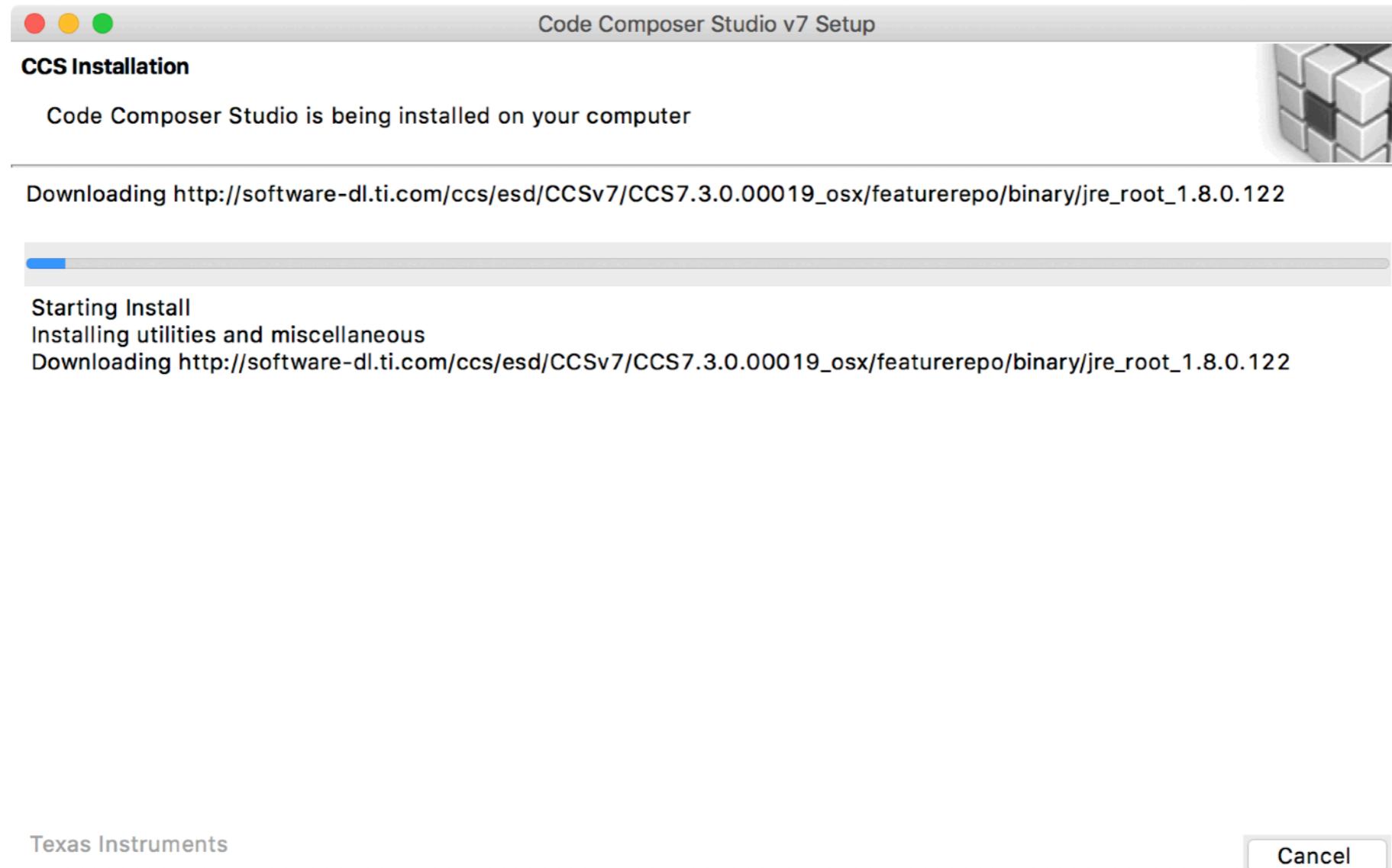
Code Composer Studio 설치

이 곳이
체크되어 있는지 확인한다.



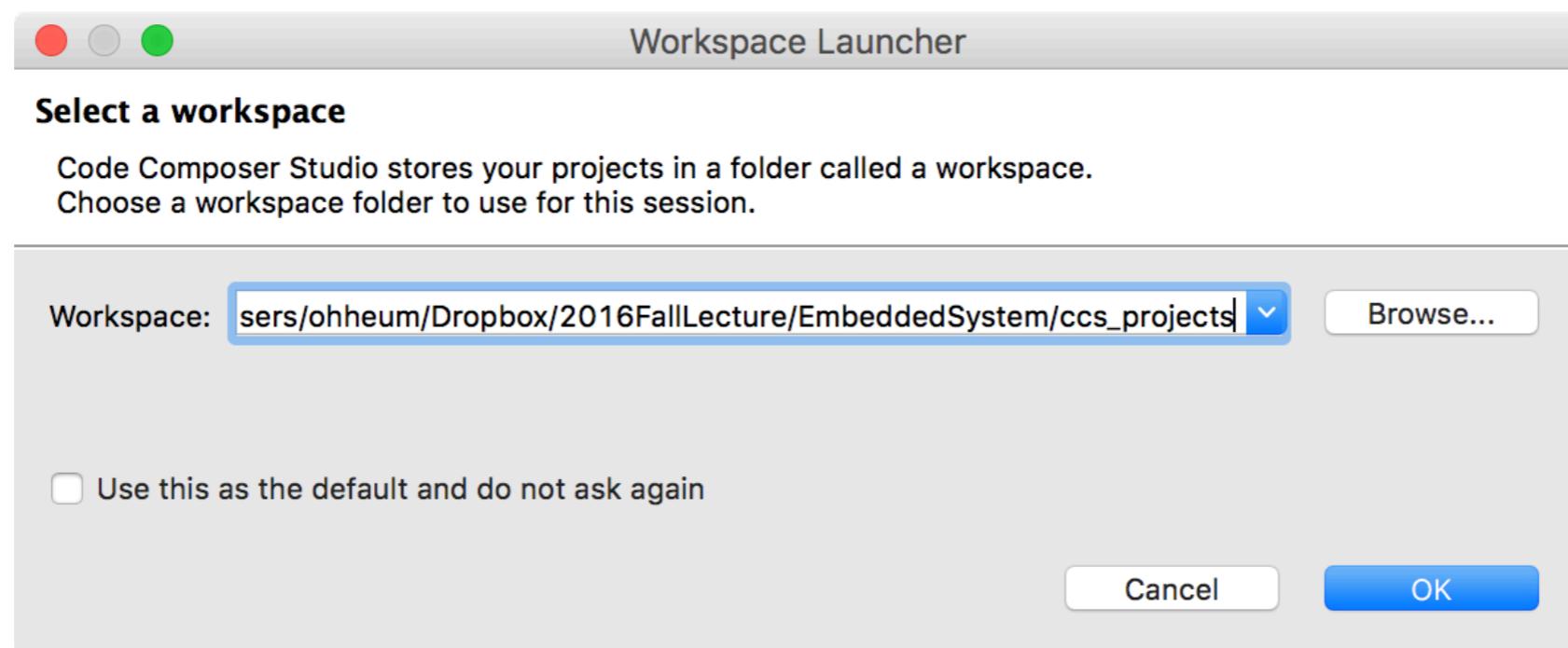
설치 과정에는 나머지는 그냥 디폴트 상태를 따른다.

Code Composer Studio 설치



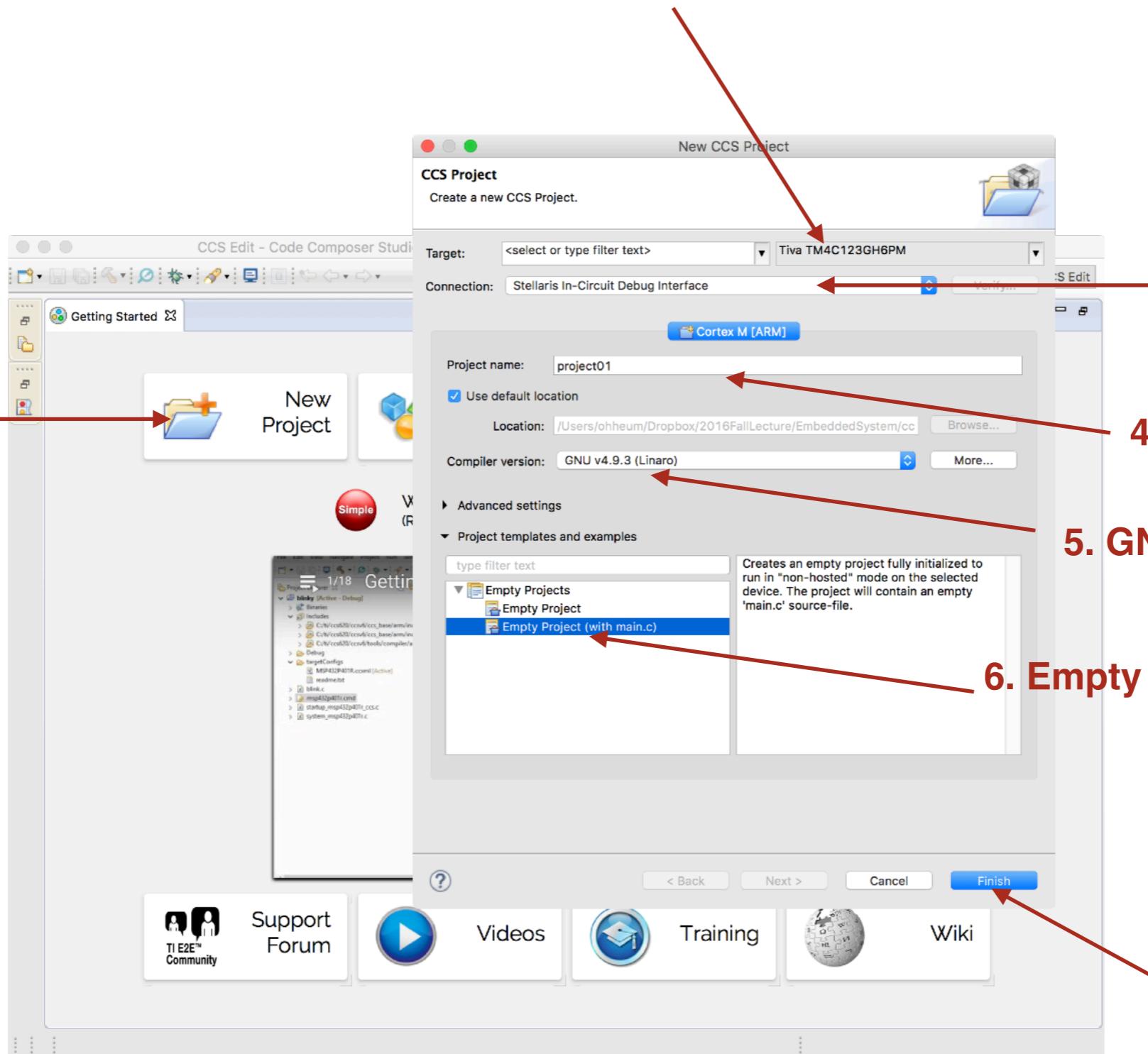
설치가 완료될 때까지 기다린다.

- ☞ CCS는 Eclipse 기반의 IDE이다. 사용법은 Eclipse와 동일하다.
- ☞ 원하는 위치에 앞으로 작성할 프로젝트를 저장할 디렉토리를 만들고 아래의 대화 상자에서 그 디렉토리를 workspace로 지정한다. 경로명에 한글이 포함되지 않는 디렉토리를 선택한다.



프로젝트 만들기

2. Tiva TM4C123GH6PM을 찾아서 선택



1. 클릭한다.

3. Stellaris In-Circuit Debug Interface를 선택

4. 프로젝트 이름을 지정한다.

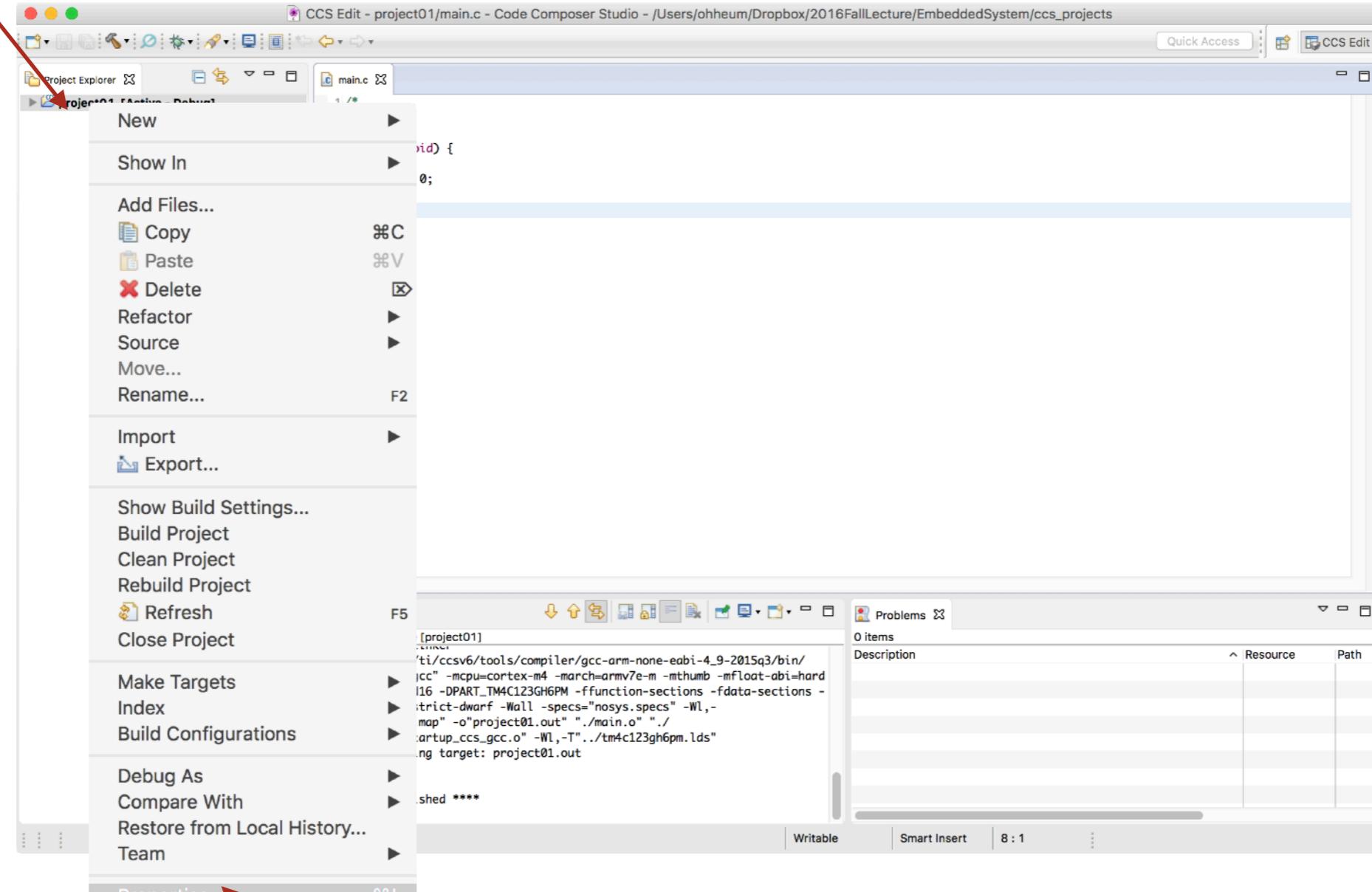
5. GNU v6.3.1 (Linaro)를 선택

6. Empty Project (with main.c)를 선택

7. Finish 버튼을 클릭

프로젝트 설정

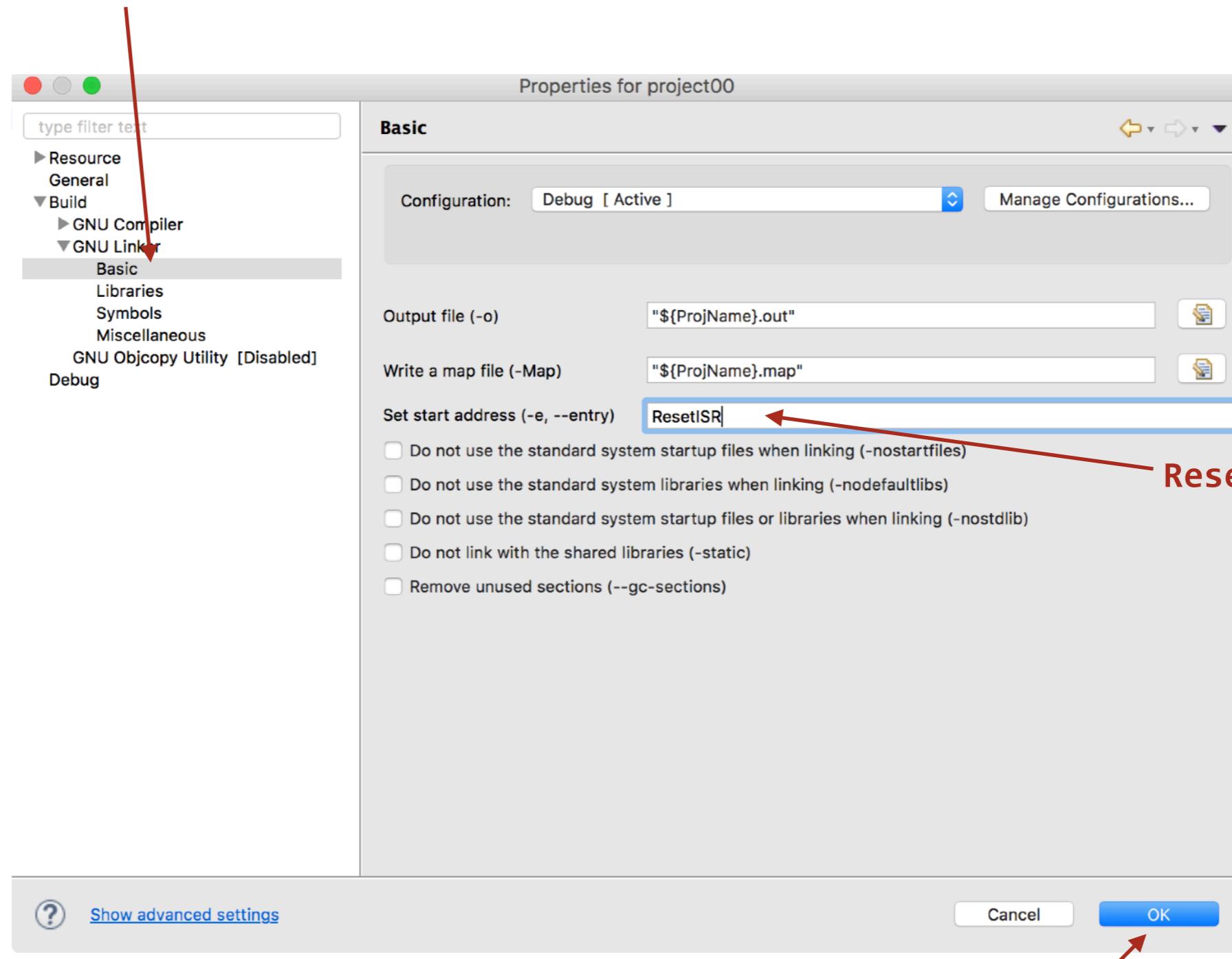
여기에서 마우스 오른쪽 버튼을 클릭한다.



Properties 메뉴를 선택한다.

프로젝트 설정

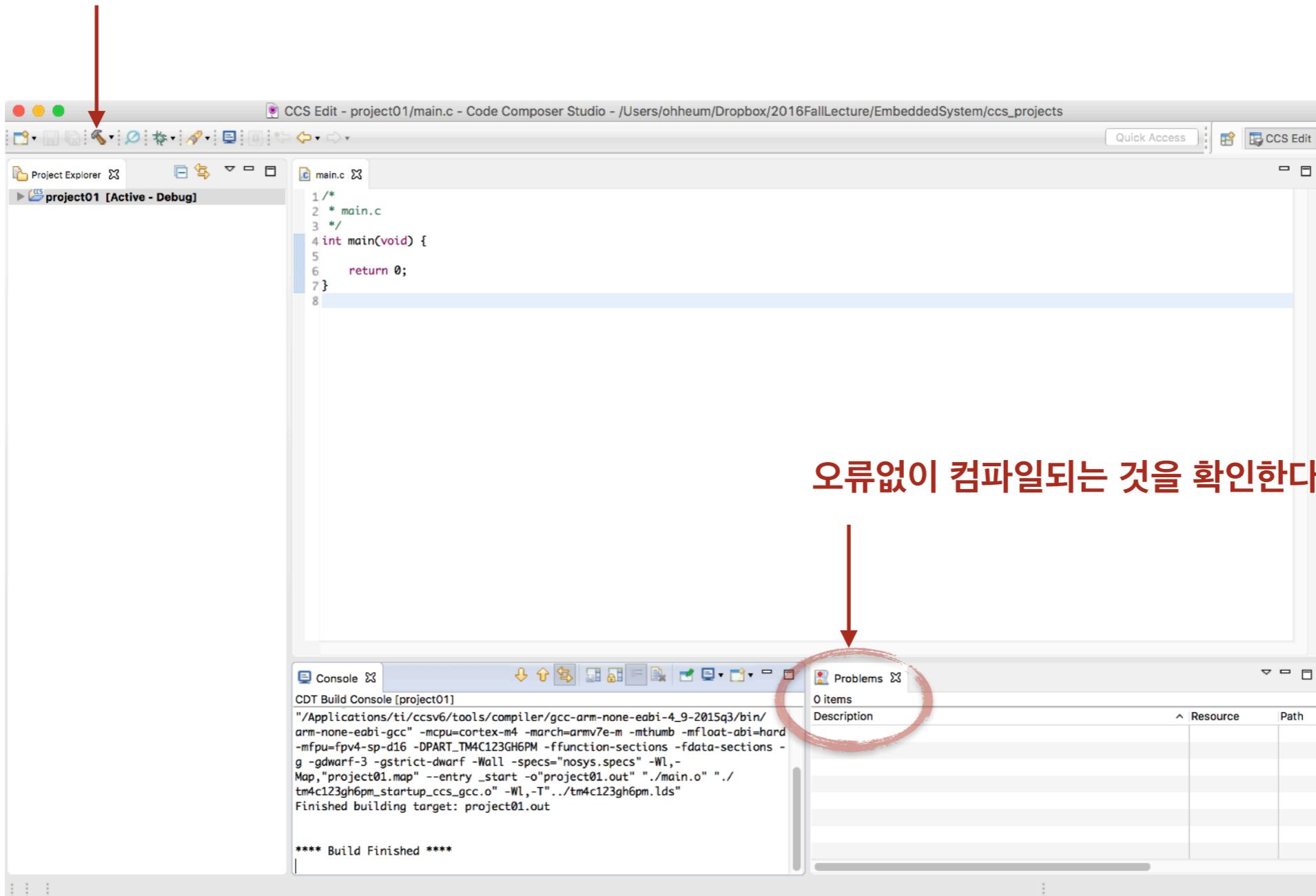
프로젝트 설정 창에서 Build->GNU Linker->Basic을 선택한다.



OK 버튼을 클릭하여 설정창을 종료한다.

프로젝트 컴파일

이 버튼을 누르거나 혹은 Build 메뉴를 실행하여
프로젝트를 빌드한다.

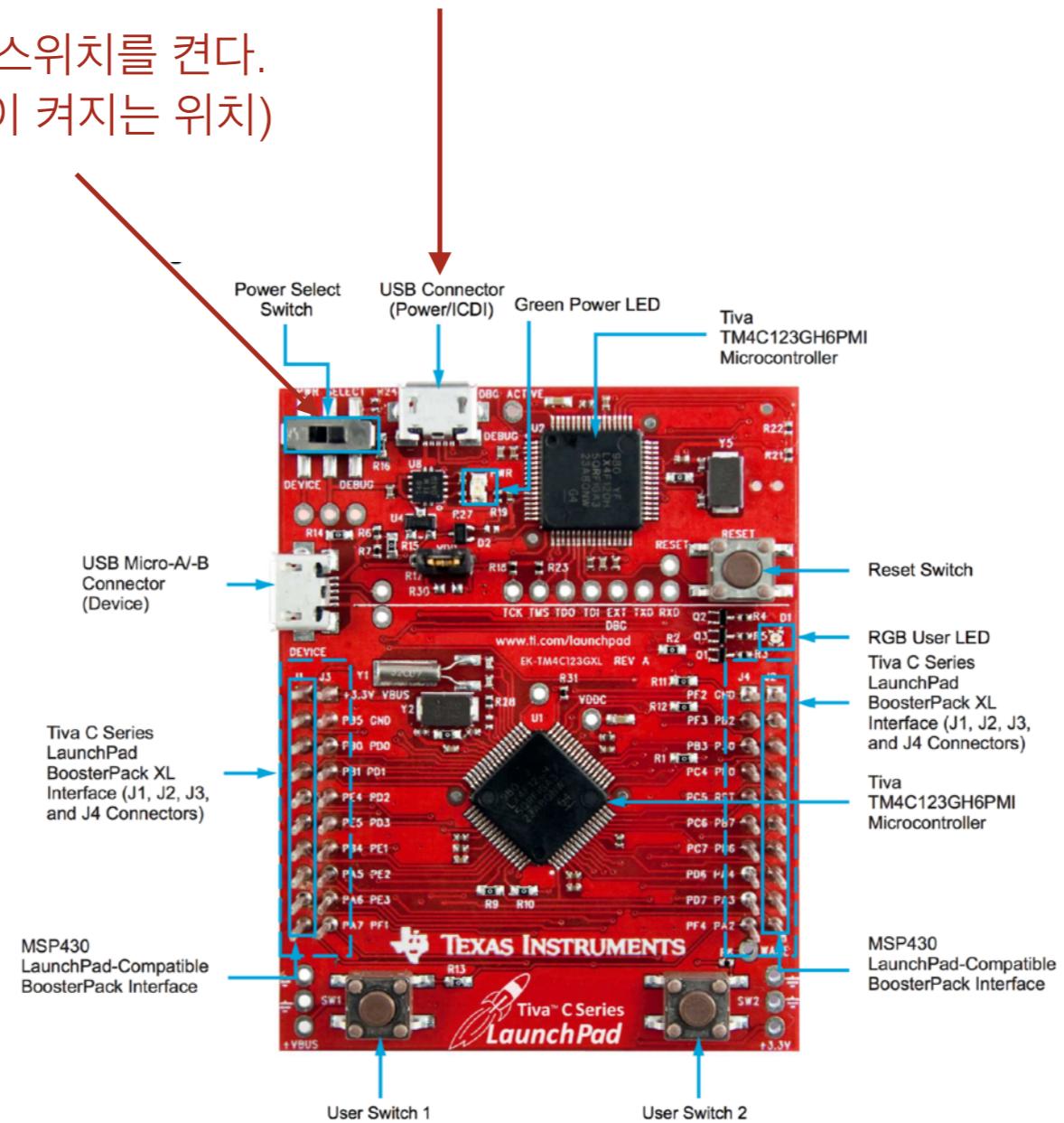


오류없이 컴파일되는 것을 확인한다.

Launchpad 연결

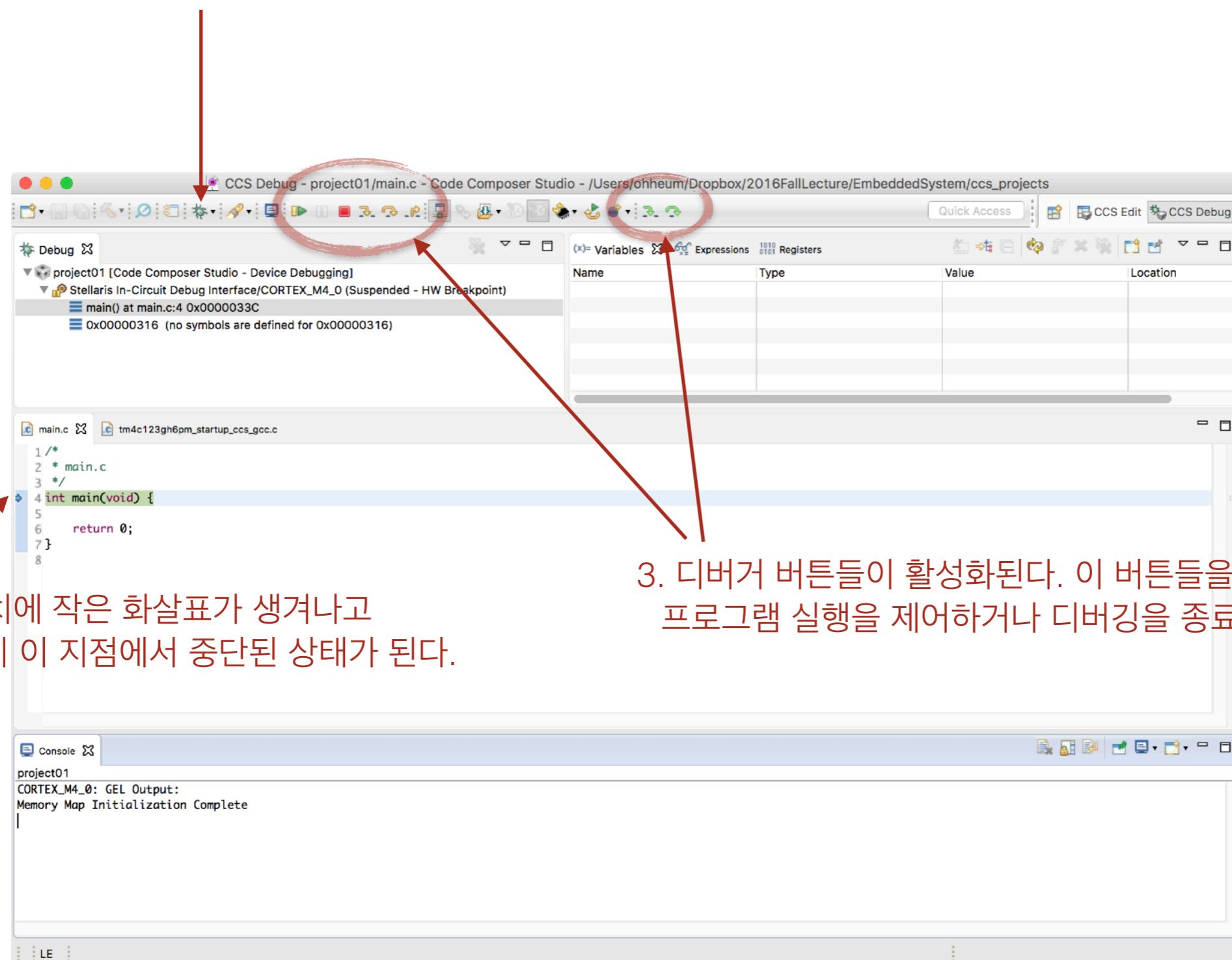
제공된 USB 케이블을 이용하여 여기와 노트북을 연결한다.

Power 스위치를 켠다.
(오른쪽이 켜지는 위치)



1. 이 버튼을 누르거나 Debug 메뉴를 실행하면 Eclipse가 Debug Perspective로 전환된다.

프로그램 다운로드 및 디버그



IDEs for ARM Cortex M4 Programming

- ⦿ **Code Composer Studio (by Texas Instruments)**
 - ⦿ Only for Texas Instruments processors
- ⦿ **IAR Embedded Workbench for ARM (by IAR)**
- ⦿ **Keil MDK-ARM (by Keil)**
- ⦿ **Atmel Studio (by Atmel)**
 - ⦿ Only for Atmel processors.

https://en.wikipedia.org/wiki/List_of_ARM_Cortex-M_development_tools#cite_note-4

ARM

- ⦿ **1990년 설립**

- ⦿ Acorn Computer에서 독립, 12 engineers & CEO
- ⦿ Cambridge, UK

- ⦿ **32-bit RISC IP (Intellectual Property) 제공**

- ⦿ ARM은 직접 칩을 제조 판매 하지는 않는다.
- ⦿ 현재 70개 이상의 semiconductor 파트너 보유
- ⦿ Apple A4, A5,..., Samsung Exynos, Nvdia Tegra,...

- ⦿ **Programmer's Model은 ARM Architecture의 분류 기준**
 - ⦿ 명령어(Instruction Set)
 - ⦿ 데이터 구조
 - ⦿ 동작 모드
 - ⦿ 레지스터의 구조
 - ⦿ Exception 처리 방식 등
- ⦿ **동일한 Architecture에도 여러 개의 프로세서가 있을 수 있다**
- ⦿ **동일한 프로세서 Family라도 Architecture가 서로 다를 수 있다.**

ARM Architecture와 프로세서

Architecture	Core bit width	Cores designed by ARM Holdings	Cores designed by third parties	Profile
ARMv1	32 ^[a 1]	ARM1		
ARMv2	32 ^[a 1]	ARM2, ARM250, ARM3	Amber , STORM Open Soft Core ^[38]	
ARMv3	32 ^[a 2]	ARM6, ARM7		
ARMv4	32 ^[a 2]	ARM8	StrongARM , FA526	
ARMv4T	32 ^[a 2]	ARM7TDMI, ARM9TDMI, SecurCore SC100		
ARMv5TE	32	ARM7EJ, ARM9E, ARM10E	XScale , FA626TE, Feroceon, PJ1/Mohawk	
ARMv6	32	ARM11		
ARMv6-M	32	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1, SecurCore SC000		Microcontroller
ARMv7-M	32	ARM Cortex-M3, SecurCore SC300		Microcontroller
ARMv7E-M	32	ARM Cortex-M4, ARM Cortex-M7		Microcontroller
ARMv7-R	32	ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7, ARM Cortex-R8		Real-time
ARMv7-A	32	ARM Cortex-A5, ARM Cortex-A7 , ARM Cortex-A8, ARM Cortex-A9 , ARM Cortex-A12, ARM Cortex-A15 , ARM Cortex-A17	Qualcomm Krait , Scorpion , PJ4/Sheeva, Apple Swift	Application
ARMv8-A	32	ARM Cortex-A32		Application
ARMv8-A	64	ARM Cortex-A35 ^[39] , ARM Cortex-A53 , ARM Cortex-A57 ^[40] , ARM Cortex-A72 ^[41] , ARM Cortex-A73 ^[42]	X-Gene , Nvidia Project Denver , AMD K12 , Apple Cyclone/Typhoon/Twister , Cavium Thunder X , ^{[43][44][45]} Qualcomm Kryo	Application
ARMv8.1-A	64/32	TBA		Application
ARMv8.2-A	64/32	TBA		Application
ARMv8-R	32	ARM Cortex-R52		Real-time
ARMv8-M	32	TBA		Microcontroller

https://en.wikipedia.org/wiki/ARM_architecture

ARM Processor Numbering

- 초기의 ARM core는 ARM{x}{labels}, 나중에는 ARM{x}{y}{z}{labels} 형태의 이름을 가짐
- 2004년 이후로 모든 ARM core는 Cortex 상표로 통일되어 Cortex-{x}{y} 형태의 이름을 가짐

X	Y	Z	DESCRIPTION	EXAMPLE
7			ARM7 core version	ARM7
9			ARM9 core version	ARM9
10			ARM10 core version	ARM10
11			ARM11 core version	ARM11
	1		Cache, write buffer and MMU	ARM710
	2		Cache, write buffer and MMU, Process ID support	ARM920
	3		Physically mapped cache and MMU	ARM1136
	4		Cache, write buffer and MPU	ARM940
	5		Cache, write buffer and MPU, error correcting memory	ARM1156
	6		No cache, write buffer	ARM966
	7		AXI bus, physically mapped cache and MMU	ARM1176
		0	Standard cache size	ARM920
		2	Reduced cache	ARM1022
		6	Tightly Coupled Memory	ARM1156
		8	As for ARM966	ARM968

ARM Label Attributes

예: ARM926EJ-S

- ARM9 core, Cache, MMU, Tightly Coupled Memory
- Enhanced DSP instruction, Jazelle Java acceleration
- Synthesizable hardware design

ATTRIBUTE	DESCRIPTION
D	Supports debugging via the JTAG interface. Automatic for ARMv5 and above.
E	Supports Enhanced DSP instructions. Automatic for ARMv6 and above.
F	Supports hardware floating point via the VFP coprocessor.
I	Supports hardware breakpoints and watchpoints. Automatic for ARMv5 and above.
J	Supports the Jazelle Java acceleration technology.
M	Supports long multiply instructions. Automatic for ARMv5 and above.
T	Supports Thumb instruction set. Automatic for ARMv5 and above.
-S	This processor uses a synthesizable hardware design.

⦿ Cortex-A series

- ⦿ Application processors series
- ⦿ fully functional computers running OS
- ⦿ Smart phone, tablets, laptops

⦿ Cortex-R series

- ⦿ Real-time processors series
- ⦿ medical devices, car systems, low-level controllers

⦿ Cortex-M series

- ⦿ Microcontroller series
- ⦿ ultra-low-powered, small form-factor
- ⦿ robotics systems, small consumer electronics

Programmer's Model

Programmer's 모델

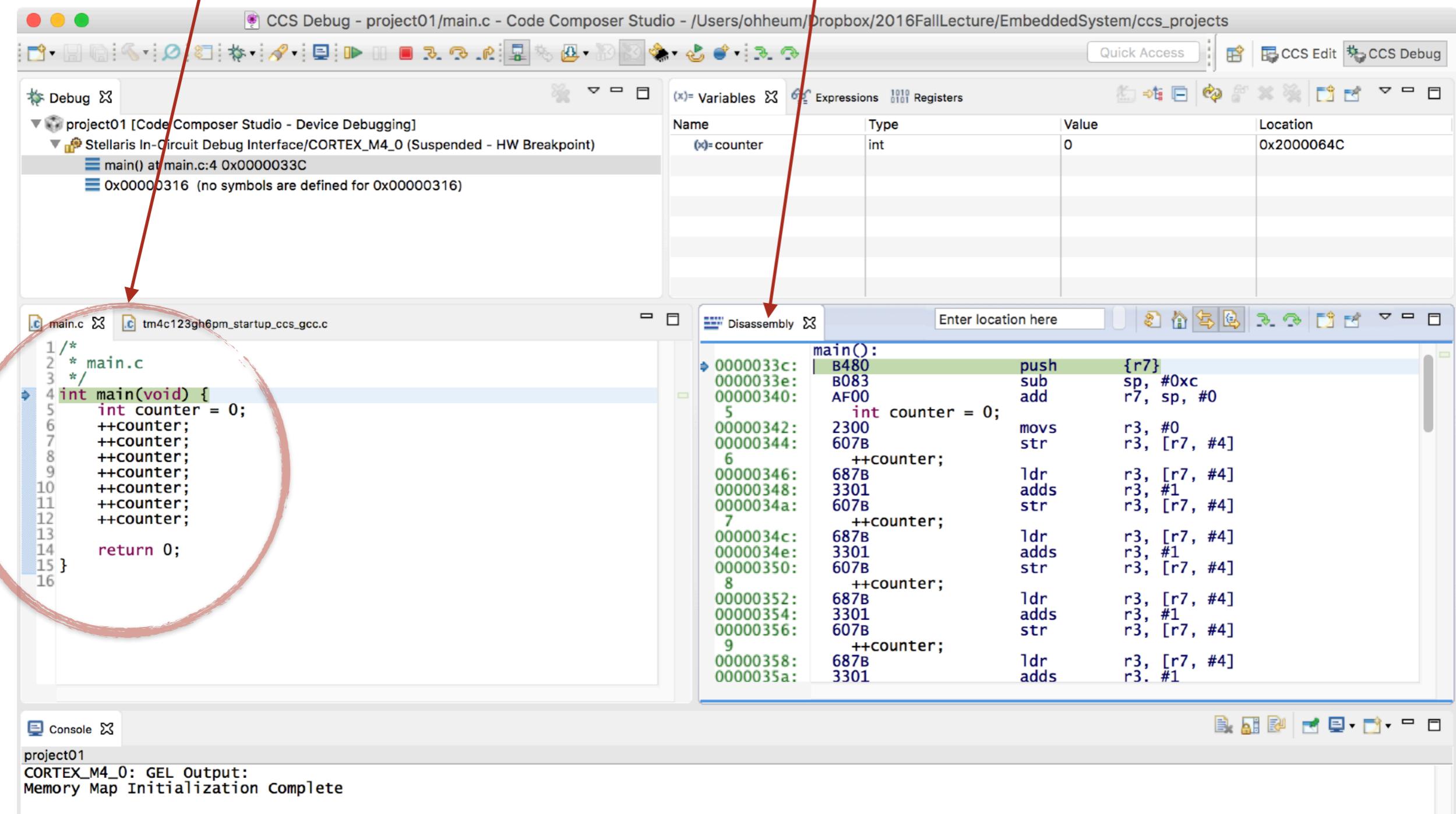
동일한 언어를 사용하는가?

- ⦿ 명령어
- ⦿ 메모리 구조
- ⦿ 데이터 구조 리틀엔디안 빅엔디안
- ⦿ 프로세서의 동작 모드 유저모드, 관리자모드
- ⦿ 프로세서 내부 레지스터의 구성 및 사용법

실습 01

다음과 같이 간단한 코드를 작성한 후 Launchpad를 연결하고 Debug해보자.

View->Disassembly 메뉴를 선택한다.



명령어(Instruction Set)

TI TM4C123의 경우
각 명령어가 2바이트 크기임을
알수 있다.

ARM 프로세서의 명령어

- 32 비트 ARM instruction set
- 16 비트 Thumb instruction set
- 16 비트와 32 비트 혼용의 **Thumb-2 instruction set**

main():		
0000033c:	B480	push
0000033e:	B083	sub
00000340:	AF00	add
5		int counter = 0;
00000342:	2300	movs
00000344:	607B	str
6		++counter;
00000346:	687B	ldr
00000348:	3301	adds
0000034a:	607B	str
7		++counter;
0000034c:	687B	ldr
0000034e:	3301	adds
00000350:	607B	str
8		++counter;
00000352:	687B	ldr
00000354:	3301	adds
00000356:	607B	str
9		++counter;
00000358:	687B	ldr
0000035a:	3301	adds

TI TM4C123
Microcontroller

ARM 명령어 요약

	Instruction Type	Instruction (명령)
1	Branch, Branch with Link	B, BL
2	Data Processing 명령	ADD, ADC, SUB, SBC, RSB, RSC, AND, ORR, BIC, MOV, MVN, CMP, CMN, TST, TEQ
3	Multiply 명령	MUL, MLA, SMULL, SMLAL, SMULL, UMLAL
4	Load/Store 명령	LDR, LDRB, LDRBT, LDRH, LDRSB, LDRSH, LDRT, STR, STRB, STRBT, STRH, STRT
5	Load/Store Multiple 명령	LDM, STM
6	Swap 명령	SWP, SWPB
7	Software Interrupt(SWI) 명령	SWI
8	PSR Transfer 명령	MRS, MSR
9	Coprocessor 명령	MRC, MCR, LDC, STC
10	Branch Exchange 명령	BX
11	Undefined 명령	

Disassembly View

```
...
8     count = 0;
00000342: 2300
00000344: 607B
9     count++;
00000346: 687B
00000348: 3301
0000034a: 607B
10    count++;
0000034c: 687B
0000034e: 3301
00000350: 607B
11    count++;
00000352: 687B
00000354: 3301
00000356: 607B
12    count++;
00000358: 687B
0000035a: 3301
0000035c: 607B
...
```

...
8 count = 0;
00000342: 2300
00000344: 607B
9 count++;
00000346: 687B
00000348: 3301
0000034a: 607B
10 count++;
0000034c: 687B
0000034e: 3301
00000350: 607B
11 count++;
00000352: 687B
00000354: 3301
00000356: 607B
12 count++;
00000358: 687B
0000035a: 3301
0000035c: 607B
...

movs r3, #0 r3 레지스터에 십진수 0을 저장한다.
str store r3, [r7, #4] ← 현재 r7에는 스택 top의 주소가 저장되어 있고,
 load register r7+4번지가 지역변수 count의 주소이다.
 진행방향 <- (스택은 주소가 감소하는 방향으로 자란다.)
 r3, [r7, #4] 즉, 이 명령은 변수 count에 0을 저장한다.

ldr r3, [r7, #4] } count의 값을 1 증가시킨다.
adds r3, #1
str r3, [r7, #4]

ldr r3, [r7, #4]
adds r3, #1
str r3, [r7, #4]

ldr r3, [r7, #4]
adds r3, #1
str r3, [r7, #4]

ldr r3, [r7, #4]
adds r3, #1
str r3, [r7, #4]

- ⦿ ARM에서 모든 명령을 조건에 따라 실행 여부 결정 가능
- ⦿ 분기 명령의 사용을 줄인다.
 - ⦿ 분기 명령이 사용되면 파이프라인이 스툴(stall) 되고 새로운 명령을 읽어오는 사이클의 낭비가 따른다.
- ⦿ 조건 필드(**Condition field**)
 - ⦿ 모든 명령어는 조건 필드를 가질수 있으며 CPU가 명령의 실행 여부를 결정하는데 사용된다.
 - ⦿ Program Status Register(PSR)의 조건 플래그의 값을 사용하여 조건을 검사

조건부 실행 방법

- 조건에 따라 명령어를 실행하도록 하기위해서는 적절한 조건을 접미사로 붙여주면 됨 :
 - 조건 없이 실행

```
ADD r0,r1,r2 ; r0 = r1 + r2
```

- PSR 레지스터의 Zero flag가 세트 되어 있을 때만 실행하고자 하는 경우

직전 명령어가

```
ADDEQ r0,r1,r2 ; If zero flag set then  
; r0 = r1 + r2
```



NE (Not Equal), LT (Less Than), GT (Greater Than), LE (Less Than or Equal) 등의 다양한 조건을 사용할 수 있다.

실습 02

The screenshot shows the Code Composer Studio interface during a debug session. The top bar indicates the project is 'CCS Debug - project01/main.c'.

The left pane shows the 'Debug' window with the project setup:

- project01 [Code Composer Studio - Device Debugging]
- Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Suspended)
- main() at main.c:6 0x0000034E
- 0x00000316 (no symbols are defined for 0x00000316)

The main editor window displays the C source code for 'main.c':1 /*
2 * main.c
3 */
4 int main(void) {
5 int counter = 0;
6 while (counter < 10)
7 ++counter;
8
9 return 0;
10}The line `while (counter < 10)` is highlighted with a red circle.

The bottom right pane shows the assembly disassembly for the same code:main():
0000033c: B480 push {r7}
0000033e: B083 sub sp, #0xc
00000340: AF00 add r7, sp, #0
5 int counter = 0;
00000342: 2300 movs r3, #0
00000344: 607B str r3, [r7, #4]
6 while (counter < 10)
00000346: E002 b #0x34e
7 ++counter;
00000348: 687B ldr r3, [r7, #4]
0000034a: 3301 adds r3, #1
0000034c: 607B str r3, [r7, #4]
6 while (counter < 10)
0000034e: 687B ldr r3, [r7, #4]
00000350: 2B09 cmp r3, #9
00000352: DDF9 ble #0x348
9 return 0;
00000354: 2300 movs r3, #0
10 }
00000356: 4618 mov r0, r3
00000358: 370C adds r7, #0xcThe assembly code for the while loop is circled in red.

실습 02: Disassembly View

```

int main(void) {
main():
0000033c: B480
0000033e: B083
00000340: AF00
5      int counter = 0;
00000342: 2300
00000344: 607B
6      while (counter < 10)
00000346: E002
7      ++counter;
00000348: 687B
0000034a: 3301
0000034c: 607B
6      while (counter < 10)
0000034e: 687B
00000350: 2B09
00000352: DDF9
9      return 0;
00000354: 2300
10 }

```

push	{r7}
sub	sp, #0xc
add	r7, sp, #0
movs	r3, #0
str	r3, [r7, #4]
b	#0x34e
ldr	r3, [r7, #4]
adds	r3, #1
str	r3, [r7, #4]
ldr	r3, [r7, #4]
cmp	r3, #9
ble	#0x348
movs	r3, #0

b는 branch 명령이다. ble는 거기에 less than이라는 조건이 붙은 것이다.

4. counter의 값을 1 증가한다

1. counter값을 r3 레지스터로 load한다.

2. r3 레지스터의 값을 상수 9와 비교한다.

(r3-#9를 계산하여 음수이면 PSR의 N, 0이면 Z flag가 set된다.)

3. 그 결과가 LESS THAN 이면, 즉 N과 Z중 하나가 1이면 0x348번지로 분기한다.

BL 명령으로 Subroutine 구현

② BL 명령으로 서브루틴으로 분기

- 다음(next) 명령의 위치를 LR 레지스터에 저장하고 지정된 주소로 분기한다.

```
bl label ; pc = label, lr = address of next instruction  
          ; label is the address of subroutine
```

③ Subroutine에서 Return

- LR에 저장된 주소를 PC에 옮긴다.

```
mov pc, lr ; pc = lr
```

- 혹은 bx명령으로 서브루틴으로부터 리턴된다.

```
bx lr ; return from function call
```

실습 03: Subroutine 호출

명령어는 짹수로 수행되는데.. 이것은 훌수이다.
필요없는 마지막 비트를 사용해서 sub를 표시함

The screenshot shows the CCS Debug interface with several windows open:

- Debug**: Shows the project is suspended at the main() function.
- Variables**: Shows the current register values. A red arrow points from the text "필요없는 마지막 비트를 사용해서 sub를 표시함" to the "Value" column of the R0 register, which has the value 0x00000000. This highlights a specific bit pattern in the assembly output.
- Disassembly**: Shows the assembly code for the main() and increment() functions. The assembly code includes instructions like push, sub, add, movs, str, b, ldr, b1, cmp, ble, mov, adds, mov, pop, and str.
- main.c**: Shows the C source code for the main() and increment() functions.

```
/* main.c
 * 
 * int increment(int x);
 *
 * int main(void) {
 *     int counter = 0;
 *     while (counter < 10)
 *         counter = increment(counter);
 *     return 0;
 * }
 *
 * int increment(int x)
 * {
 *     int y;
 *     y = x + 1;
 *     return y;
 * }
```

```
main():
0000033c: B580      push    {r7, lr}
0000033e: B082      sub    sp, #8
00000340: AF00      add    r7, sp, #0
8          int counter = 0;
00000342: 2300      movs   r3, #0
607B      str    r3, [r7, #4]
9          while (counter < 10)
00000346: E003      b      #0x350
10         counter = increment(counter);
6878      ldr    r0, [r7, #4]
F000F809  b1    #0x360
0000034a: 6078      str    r0, [r7, #4]
9          while (counter < 10)
00000350: 687B      ldr    r3, [r7, #4]
00000352: 2B09      cmp    r3, #9
00000354: DDF8      ble    #0x348
12         return 0;
00000356: 2300      movs   r3, #0
13         }
00000358: 4618      mov    r0, r3
0000035a: 3708      adds   r7, #8
0000035c: 46BD      mov    sp, r7
0000035e: BD80      pop    {r7, pc}
16         {
increment():
00000360: B480      push    {r7}
00000362: B085      sub    sp, #0x14
00000364: AF00      add    r7, sp, #0
00000366: 6078      str    r0, [r7, #4]
18         y = x + 1;
00000368: 687B      ldr    r3, [r7, #4]
0000036a: 3301      adds   r3, #1
0000036c: 60FB      str    r3, [r7, #0xc]
19         return y;
0000036e: 68FB      ldr    r3, [r7, #0xc]
```

ABI – binary 기계어 수준에서 사용되는 규칙들이 저장
함수호출과 관련된 규칙들도 저장
API – program

실습 03: Subroutine 호출

```

main():
0000033c: B580      push    {r7, lr}
0000033e: B082      sub     sp, #8
00000340: AF00      add     r7, sp, #0
8       int counter = 0;
00000342: 2300      movs    r3, #0
00000344: 607B      str     r3, [r7, #4]
9       while (counter < 10)
00000346: E003      b      #0x350
10      counter = increment(counter);
00000348: 6878      ldr     r0, [r7, #4]
0000034a: F000F809   bl      #0x360
0000034e: 6078      str     r0, [r7, #4]
9       while (counter < 10)
00000350: 687B      ldr     r3, [r7, #4]
00000352: 2B09      cmp     r3, #9
00000354: DDF8      ble
12      return 0;
00000356: 2300      movs    r3, #0
13      }
00000358: 4618      mov     r0, r3
0000035a: 3708      adds    r7, #8
0000035c: 46BD      mov     sp, r7
0000035e: BD80      pop
16      {
increment():
00000360: B480      push    {r7}
00000362: B085      sub     sp, #0x14
00000364: AF00      add     r7, sp, #0
00000366: 6078      str     r0, [r7, #4]
18      y = x + 1;
00000368: 687B      ldr     r3, [r7, #4]
0000036a: 3301      adds    r3, #1
0000036c: 60FB      str     r3, [r7, #0xc]
19      return y;
0000036e: 68FB      ldr     r3, [r7, #0xc]
20      }
00000370: 4618      mov     r0, r3
00000372: 3714      adds    r7, #0x14
00000374: 46BD      mov     sp, r7
00000376: F85D7B04   ldr     r7, [sp], #4
0000037a: 4770      bx     lr
255      {

```

1. 변수 counter의 값(메모리 주소 r7+4)을 r0에 저장한다. 즉 r0가 매개변수 전달 역할을 한다.
2. bl 명령으로 return address를 lr 레지스터에 저장하고 서브루틴으로 분기한다.
(LR 레지스터의 값이 0x35F가 됨을 확인해보자.)
3. 이전 스택 프레임의 주소(r7)을 스택에 push 한다. push명령은 sp를 자동으로 변경한다.
4. 매개변수 x와 지역변수 y를 스택에 할당한다.
5. 현재의 스택 top의 주소를 r7에 저장한다.
6. r7+4는 매개변수 x의 주소이다. x에 r0를 통해 전달된 매개변수 값을 저장한다.
^— 여기까지가 매개변수 전달과정이다.
7. r7+0xc는 지역변수 y의 주소이다.
8. bx 명령으로 lr 레지스터에 저장된 주소로 분기 즉 return한다.

Registers

Low registers	R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12						
High registers							
Stack Pointer	SP (R13)						
Link Register	LR (R14)						
Program Counter	PC (R15)						
	<table border="1"><tbody><tr><td>PSR</td><td>Program Status Register flag들의 집합.. 오버플로우가 생겼나 등등</td></tr><tr><td>PRIMASK</td><td>Interrupt mask register</td></tr><tr><td>CONTROL</td><td>Control Register</td></tr></tbody></table>	PSR	Program Status Register flag들의 집합.. 오버플로우가 생겼나 등등	PRIMASK	Interrupt mask register	CONTROL	Control Register
PSR	Program Status Register flag들의 집합.. 오버플로우가 생겼나 등등						
PRIMASK	Interrupt mask register						
CONTROL	Control Register						

- 16개의 범용 레지스터: R0~R15
 - R0에서 R12는 데이터 연산에 사용
 - R13 (**SP** : Stack Pointer)
 - R14 (**LR** : Link Register)
 - R15 (**PC** : Program Counter)
- 1개의 **PSR (Program Status Register)**

Register View in CCS

(x)= Variables	Expressions	Registers	Breakpoints
Name		Value	Description
Core Registers			
PC		0x00000033C	Program Counter [Core]
SP		0x20000658	General Purpose Register 13 - Stack Pointer [Core]
LR		0x000000317	General Purpose Register 14 - Link Register [Core]
xPSR		0x61000000	Stores the status of interrupt enables and critical processor status signals [Core]
N		0	Stores bit 31 of the result of the instruction. In other words stores the sign of the result.
Z		1	Is set to 1 if the result of the operation is zero else stays 0
C		1	Stores the value of the carry bit if it occurred in an addition or the borrow bit in a subtraction
V		0	Set to 1 if an overflow occurred
Q		0	Indicates whether an overflow/saturation occurred in the enhanced DSP instructions
ICL_IT_2		00	ICI/IT - bit26-bit25
T		1	Thumb State
RESV		00000000	Reserved
ICL_IT_1		000000	ICI/IT - bit15-bit10
RESV2		0	Reserved
EXCEPTION		000000000	Exception Number
R0		0x000000000	General Purpose Register 0 [Core]
R1		0x000000000	General Purpose Register 1 [Core]
R2		0x000000000	General Purpose Register 2 [Core]
R3		0x000000000	General Purpose Register 3 [Core]
R4		0x000000000	General Purpose Register 4 [Core]
R5		0x000000000	General Purpose Register 5 [Core]
R6		0x000000000	General Purpose Register 6 [Core]
R7		0x000000000	General Purpose Register 7 [Core]
R8		0x000000000	General Purpose Register 8 [Core]
R9		0x000000000	General Purpose Register 9 [Core]
R10		0x1FFF0658	General Purpose Register 10 [Core]
R11		0x000000000	General Purpose Register 11 [Core]
R12		0x000000000	General Purpose Register 12 [Core]
R13		0x20000658	General Purpose Register 13 [Core]
R14		0x000000317	General Purpose Register 14 [Core]
MSP		0x20000658	MSP Register [Core]
DSD		0x000000000	DSD Register [Core]

• Program Counter (PC 또는 R15)

- 실행할 명령어를 읽어올 메모리 주소를 저장

• Stack Pointer (SP 또는 R13)

- 스택 탑(top)의 주소를 저장

• Link Register (LR 또는 R14)

- 서브루틴에서 되돌아 갈 주소(return address)를 저장
- 서브루틴을 호출하는 BL 명령이 PC 값을 자동으로 LR에 저장하고, 되돌아 갈 때는 MOV 혹은 BX 명령을 이용하여 LR 값을 PC에 저장

• PSR (Program Status Register)

- 프로세서의 현재 동작 상태를 나타냄

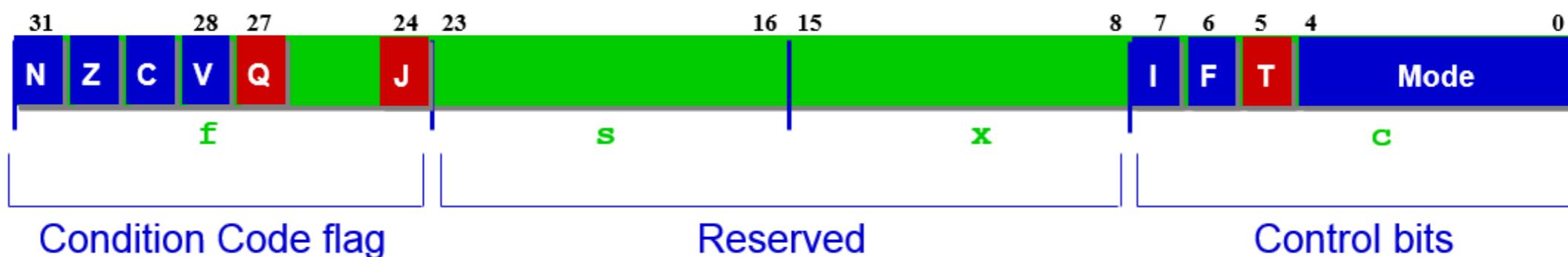
Program Status Register (PSR)

Condition code flag

- ALU의 연산 결과 정보를 가지는 flag들

Control bits

- 프로세서를 제어하기 위한 비트로 구성되어 있다



PSR 레지스터 – flag bits

- Flag bits는 ALU를 통한 명령의 실행 결과를 나타낸다.

이름		설명
N	Negative flag	ALU 연산 결과 마이너스가 발생한 경우 세트
Z	Zero flag	ALU 연산 결과 0가 발생한 경우 세트
C	Carry flag	ALU 연산 결과 자리올림이나 내림이 발생한 경우 세트 shift 연산에서 carry가 발생한 경우 세트
V	oVerflow flag	ALU 연산 결과 overflow가 발생하면 세트
Q	Q flag	ARM Architecture v5TE/J에서 새롭게 추가된 saturation 연산 명령의 수행 결과 saturation이 발생하면 세트된다.
J		ARM Architecture v5TEJ에서 새롭게 추가된 Java 바이트 코드를 수행하는 Jazelle state임을 나타낸다.

Name	Value	Description
LR	0x0000034F	General Purpose Register 14 - Link Register
xPSR	0x01000000	Stores the status of interrupt enables and the result of the operation.
N	0	Stores bit 31 of the result of the instruction.
Z	0	Is set to 1 if the result of the operation is zero.
C	0	Stores the value of the carry bit if it occurred.
V	0	Set to 1 if an overflow occurred.
Q	0	Indicates whether an overflow/saturation has occurred.
ICI_IT_2	00	ICI/IT - bit26-bit25
T	1	Thumb State
RESV	00000000	Reserved
ICI_IT_1	000000	ICI/IT - bit15-bit10
RESV2	0	Reserved
EXCEPTION	00000000	Exception Number

⦿ Big/Little Endian 지원

- ⦿ ARM core는 Big-Endian과 Little-Endian을 모두 지원 (Default: Little)
- ⦿ 외부의 핀에 의해서 하드웨어적으로 결정된다.

⦿ ARM에서 사용 가능한 데이터 Type

- ⦿ byte : 8비트
- ⦿ halfword : 16 비트, 2 바이트
- ⦿ word : 32 비트, 4 바이트

⦿ Unaligned access

- ⦿ 기존의 ARM Architecture v4T, v5T, v5TE 등은 지원되지 않는다.
 - ⦿ Data Abort 발생
- ⦿ ARM Architecture v6에서는 지원한다.

Aligned 과 Unaligned 엑세스

- 메모리 엑세스는 **Byte**, **halfword(2바이트)** 또는 **word(4바이트)** 단위로만 가능

모든 메모리 주소는 짹수여야한다.
모든 메모리 주소는 4의 배수여야한다.

Aligned Access

31	24	23	16	15	8	7	0
11	22	33	44				0x0C
11	22	33	44				0x08
11	22	33	44				0x04
11	22	33	44				0x00

Word 단위 Access :
0x00, 0x04, 0x08, ...

Half-Word Access :
0x00, 0x02, 0x04, 0x06, 0x08, ...

Un-aligned Access

31	24	23	16	15	8	7	0
11	22	33	44				0x0C
11	22	33	44				0x08
11	22	33	44				0x04
11	22	33	44				0x00

Word 단위 Access :  Abort
0x01, 0x06, 0x07, ...

Half-Word Access :  Abort
0x01, 0x03, 0x05 ...

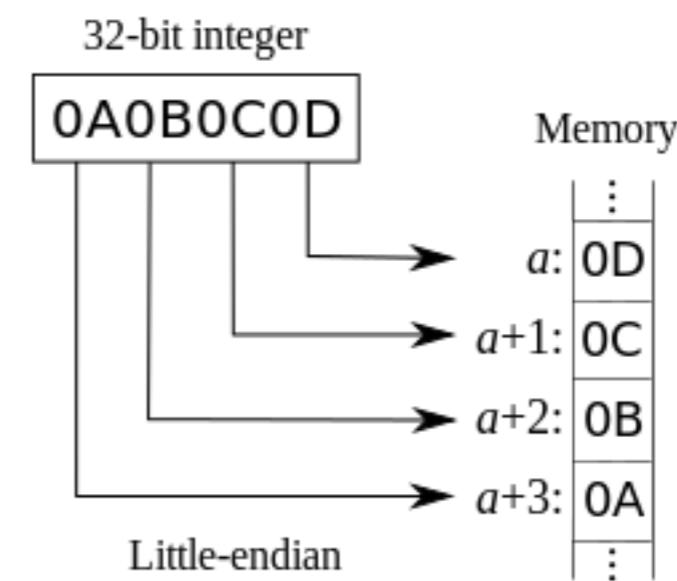
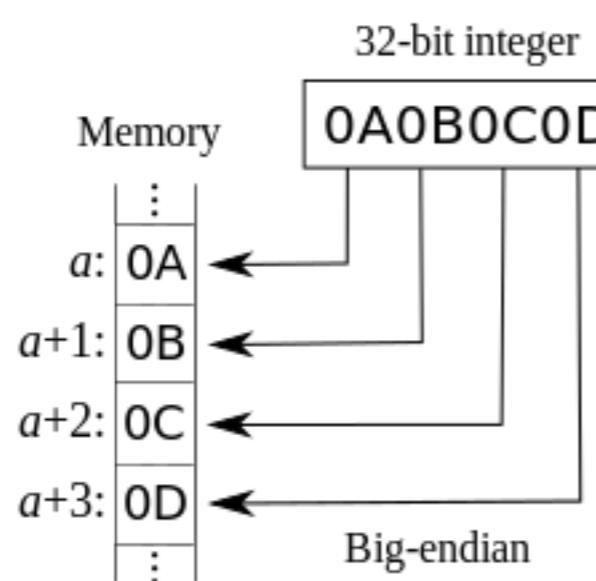
Little-Endian 과 Big-Endian

Big-Endian

- 메모리의 하위 어드레스(Byte 어드레스 “0”)에 MSB(Most Significant Byte)가 위치하고 있는 메모리 구조
- LSB는 메모리 데이터의 가장 상위 비트 24에서 31 까지

Little-Endian

- 메모리의 하위 어드레스(Byte 어드레스 “0”)에 LSB(Least Significant Byte)가 위치하고 있는 메모리 구조
- LSB는 메모리 데이터의 가장 하위 비트 0에서 7 까지



TI Tiva EK-TM4C123GXL Launchpad

TI Tiva EK-TM4C123GXL Launchpad

• **TM4C123GH6PM microcontroller**

- 32-bit ARM Cortex-M4F architecture optimized for small-footprint embedded applications
- Thumb-2 mixed 16-/32-bit instruction set

• **Documents**

- [TM4C123GH6PM Microcontroller Data sheet](#)
- [Launchpad Evaluation Board User Guide](#)

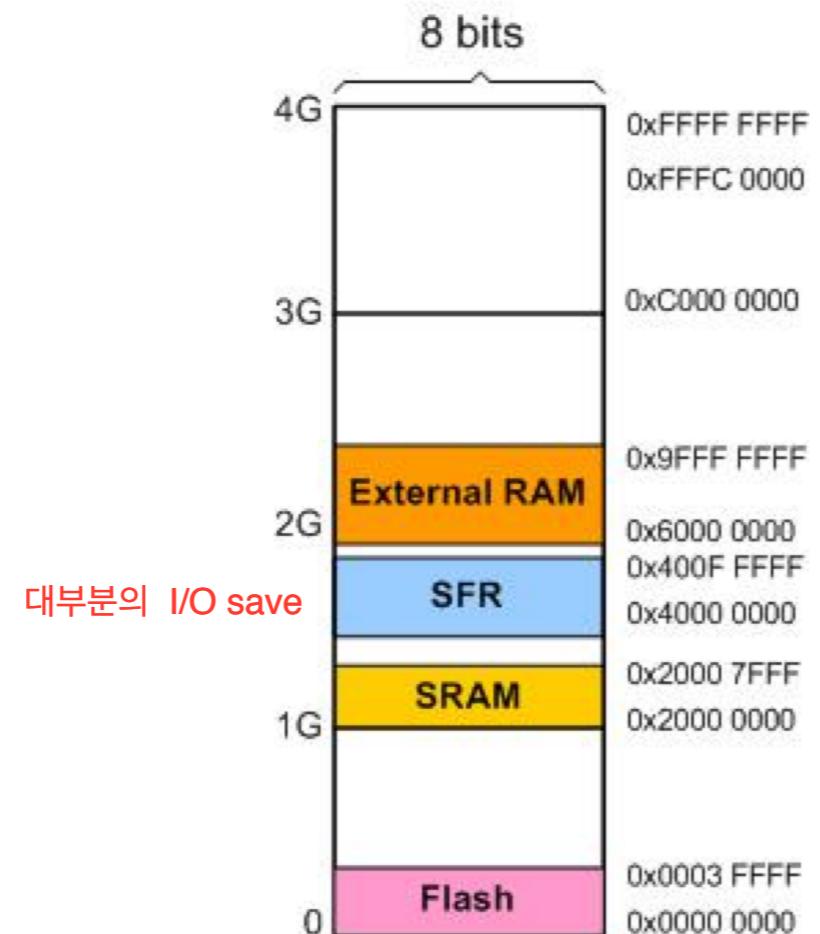
다운받을것.

TI Tiva Launchpad Memory Map

	Allocated size	Allocated address
Flash (for code)	256KB	0x00000000 to 0x0003FFFF
SRAM (for data)	32KB	0x20000000 to 0x2007FFF
I/O	All the peripherals	0x40000000 to 0x400FFFFF

Table 2-1: Memory Map in TM4C123GH6PM

Memory-Mapped IO



ARM has 4GB
of memory space

Figure 2-3: Memory Map

실습: CCS 디버거 익히기

