

EN675

Cmake와 Eclipse를 활용한 개발환경구성



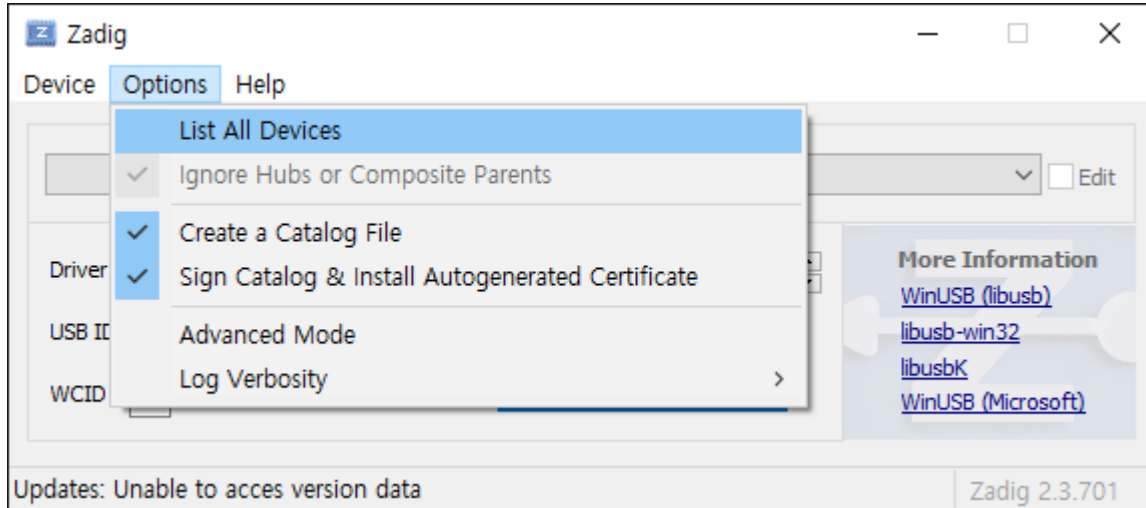
Version 0.6

Mar. 24, 2020

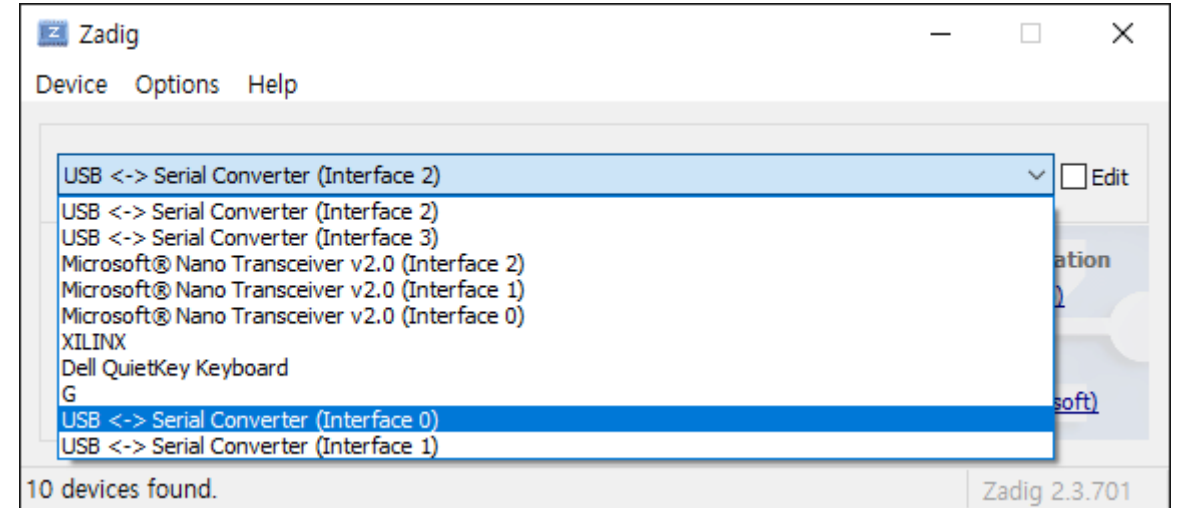
1. openOCD가 ftdi를통해 WinUSB를 사용하게 바꿔주는 역할

- zadig 프로그램 <https://zadig.akeo.ie/>

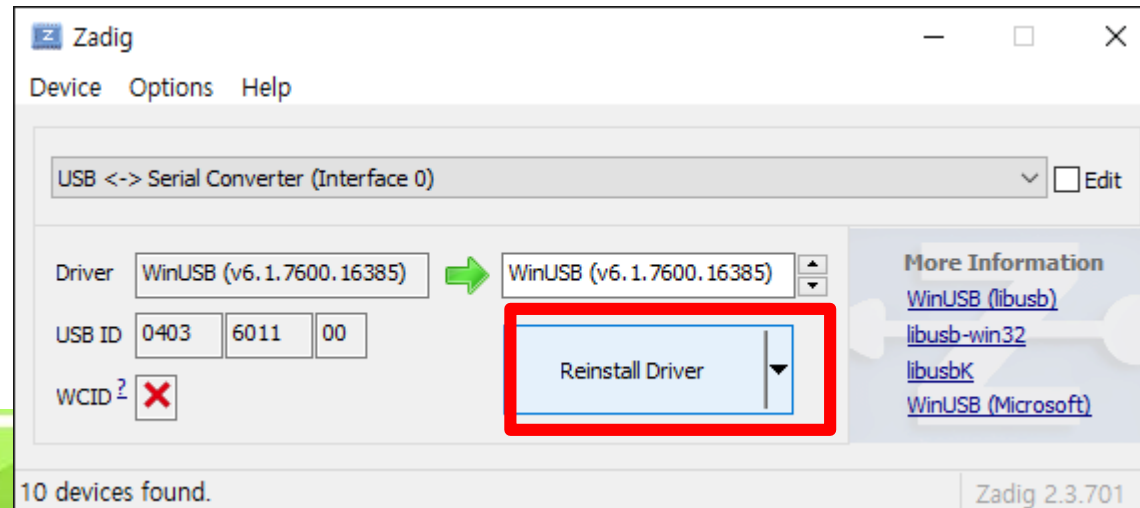
2. Options > List All Devices



3. Select USB<->Serial Converter (Interface 0)

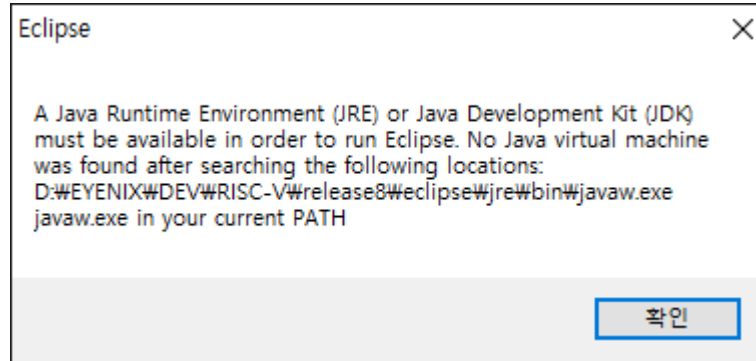


4. Click Reinstall Driver



1. java 유료화 정책으로 openjdk 설치 권장

- <https://github.com/ajdkbuild/ajdkbuild>



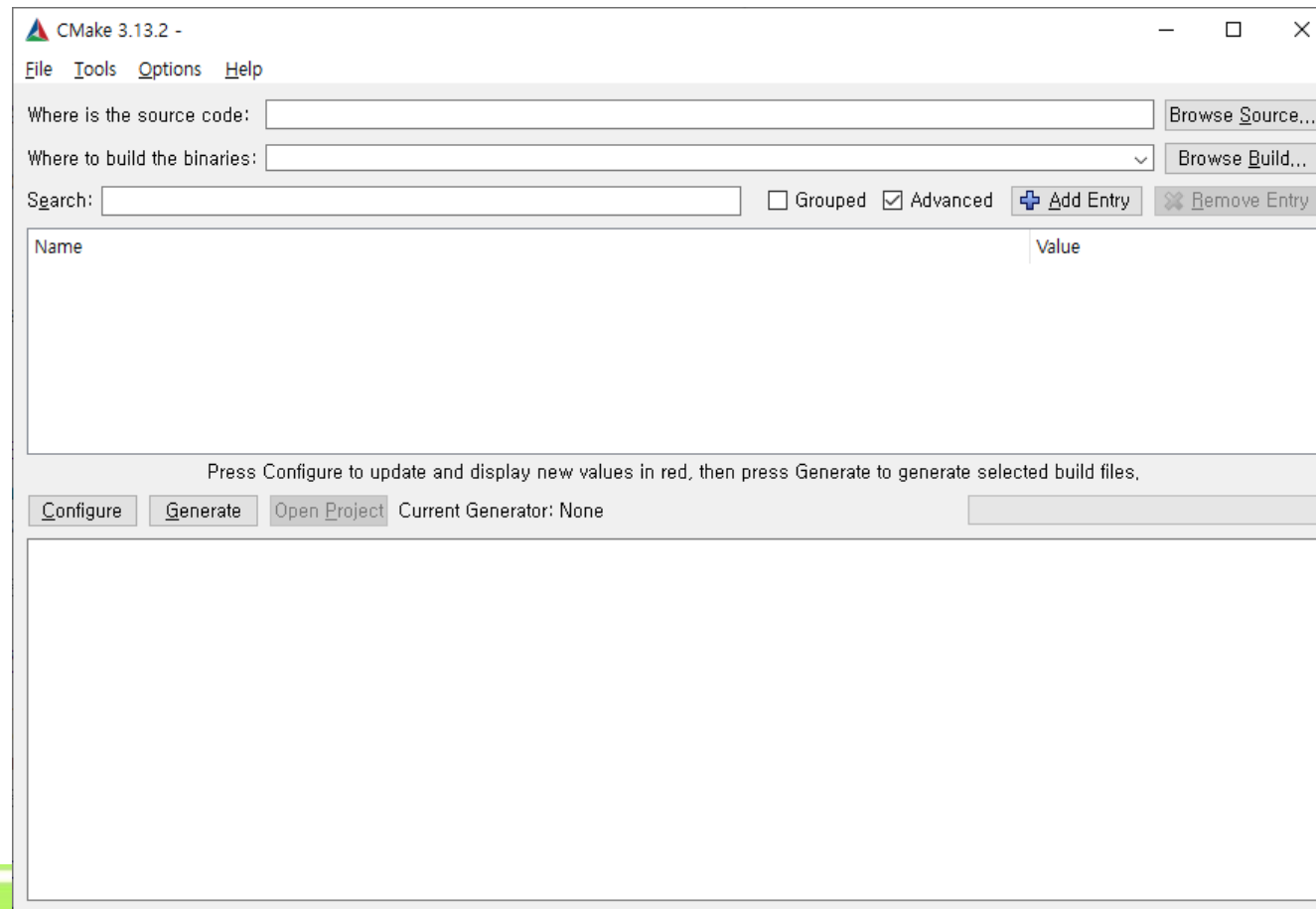
- Download link

https://github.com/ajdkbuild/ajdkbuild/releases/download/1.8.0.201-1/java-1.8.0-openjdk-1.8.0.201-1.b09.ajdkbuild.windows.x86_64.msi

1. makefile 생성을 위해 필요

- 최소 3.13.2 버전 이상
- Download link

<https://cmake.org/download/>



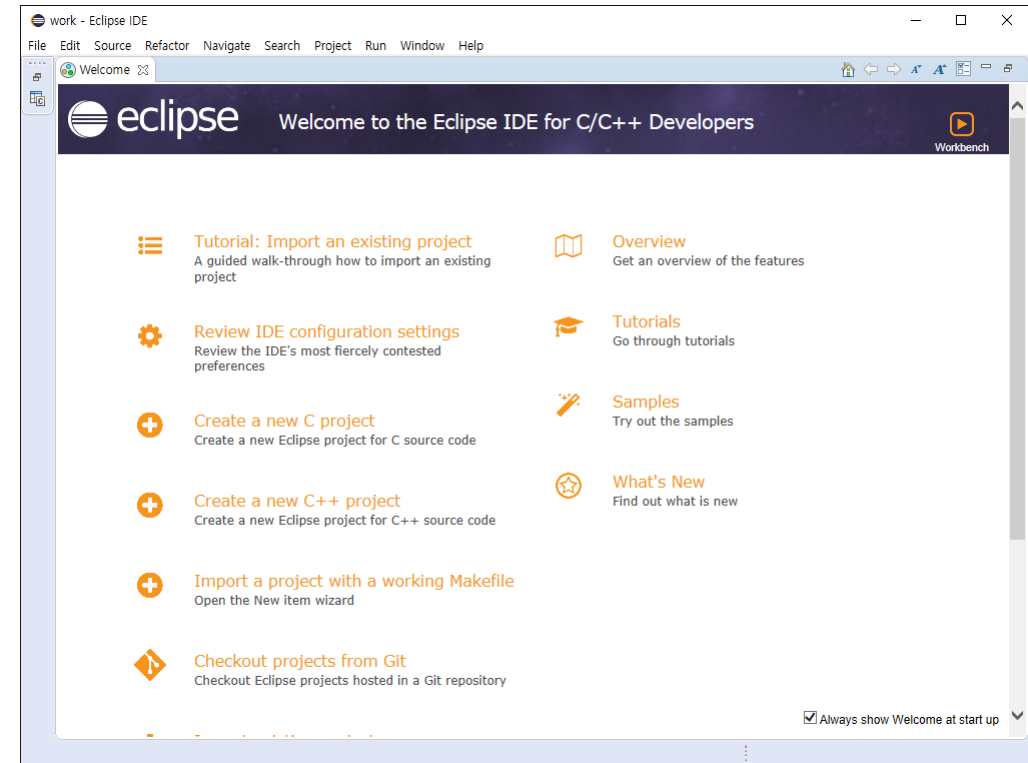
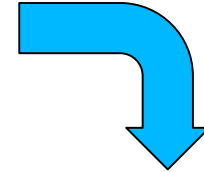
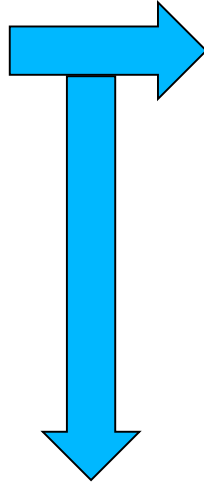
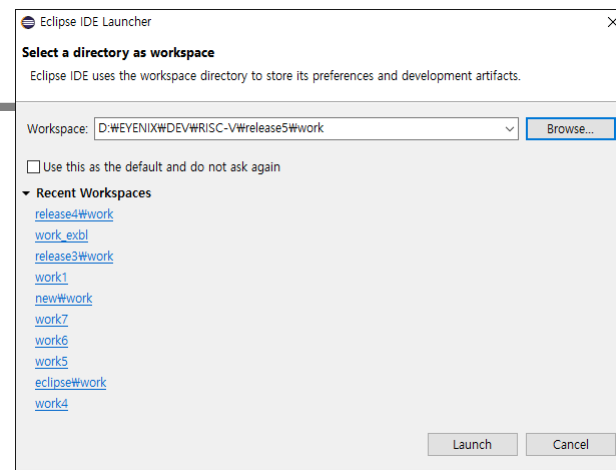
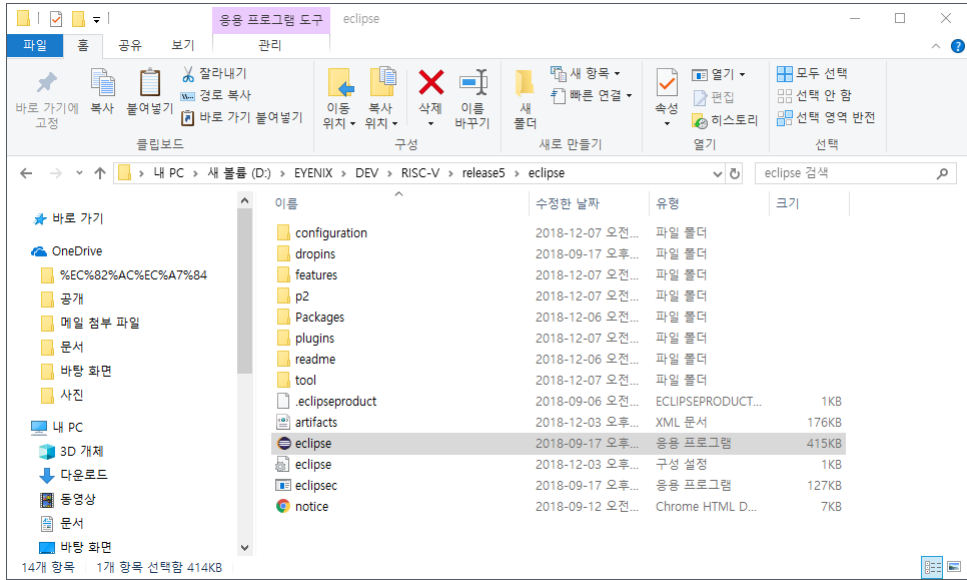
1. 소스코드 Git 주소

- http://192.168.0.217:8000/en675/en675_fpga1.git

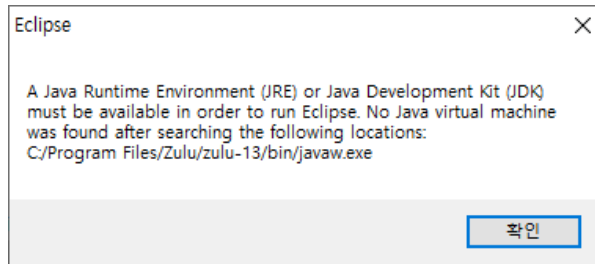
2. Eclipse, gcc 및 openocd의 버전을 확인합니다.

- Eclipse: eclipse-cpp-2019-03-R-win32-x86_64_20191210_update.zip
- gcc: riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-w64-mingw32.zip
- openocd: openocd-0.10.0-eyenix-sba-x86_64-w64.zip
- 버전이 다를 경우, \\192.168.0.10_연구소_Team2_EN675\개발환경구성\toolchain_191210 에서 다운로드 받습니다.

Eclipse 실행 / 시작화면



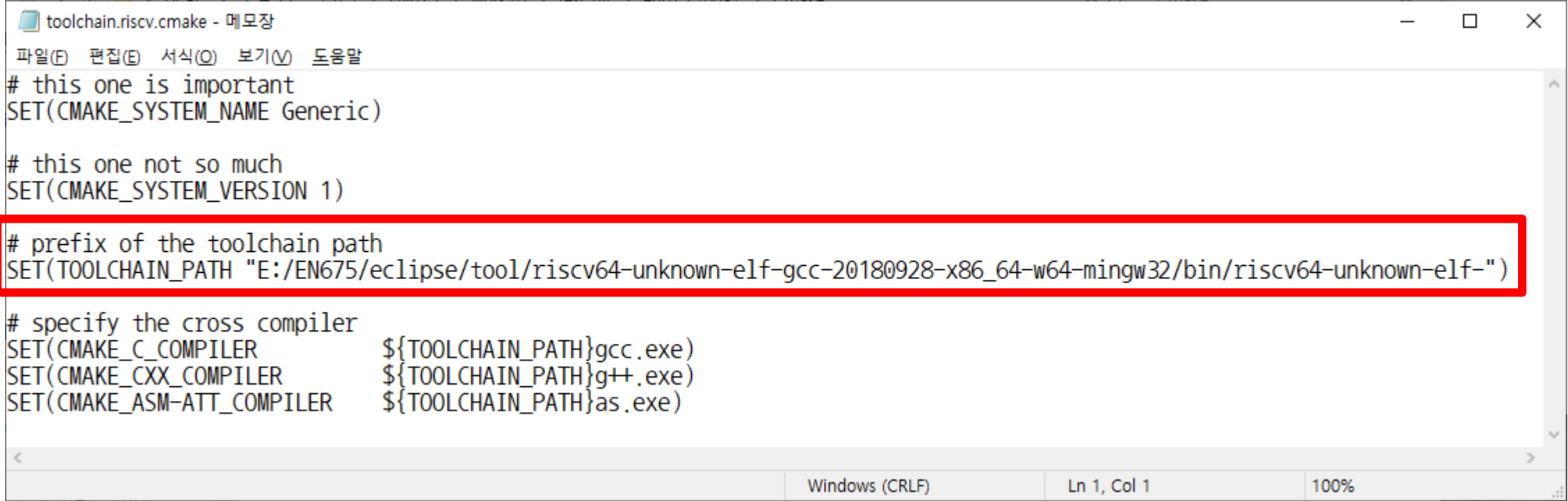
- 오류 발생할 경우
같은 폴더의 eclipse.ini 수정



```
--launcher.defaultAction
openFile
launcher.appendVMargs
-vm
C:/Program Files/ojdkbuild/java-1.8.0-openjdk-1.8.0.201-1/bin
-Dosgi.requiredJavaVersion=1.8
```

JDK설치 경로

1. 다운받은 소스코드 경로를 E:\EN675\work10\test_git\en675_fpga1 으로 **가정**합니다.
2. en675_fpga1\Cmake\toolchain.riscv.cmake 파일을 열어서 Toolchain 경로를 지정하고 저장합니다.



```

toolchain.riscv.cmake - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말
# this one is important
SET(CMAKE_SYSTEM_NAME Generic)

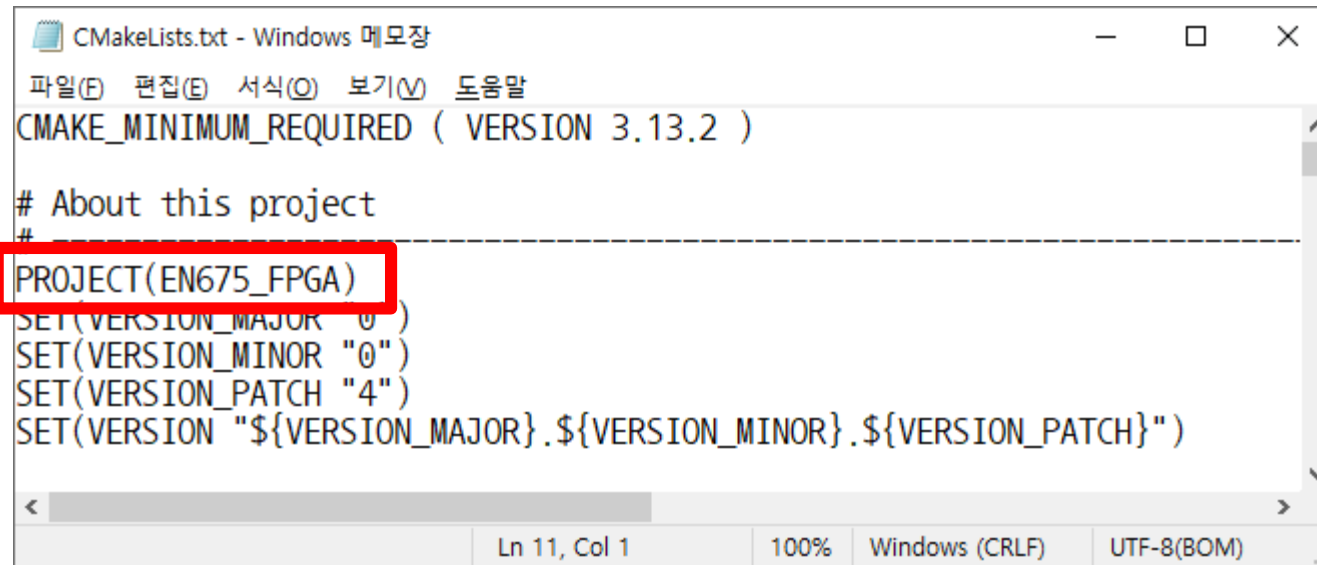
# this one not so much
SET(CMAKE_SYSTEM_VERSION 1)

# prefix of the toolchain path
SET(TOOLCHAIN_PATH "E:/EN675/eclipse/tool/riscv64-unknown-elf-gcc-20180928-x86_64-w64-mingw32/bin/riscv64-unknown-elf-")

# specify the cross compiler
SET(CMAKE_C_COMPILER      ${TOOLCHAIN_PATH}gcc.exe)
SET(CMAKE_CXX_COMPILER    ${TOOLCHAIN_PATH}g++.exe)
SET(CMAKE_ASM-ATT_COMPILER ${TOOLCHAIN_PATH}as.exe)
  
```

Project 이름 변경하기

- 하나의 Eclipse Workspace에서 복수의 Project를 운영하기 위해서는 Project의 이름이 달라합니다. 이를 위해 아래와 같이 Project 이름을 변경합니다.
 - 다운받은 소스코드 경로를 E:\EN675\work10\test_git\en675_fpga1 으로 **가정**합니다.
 - en675_fpga1\CMakeLists.txt 에서 PROJECT 이름을 변경합니다.



```

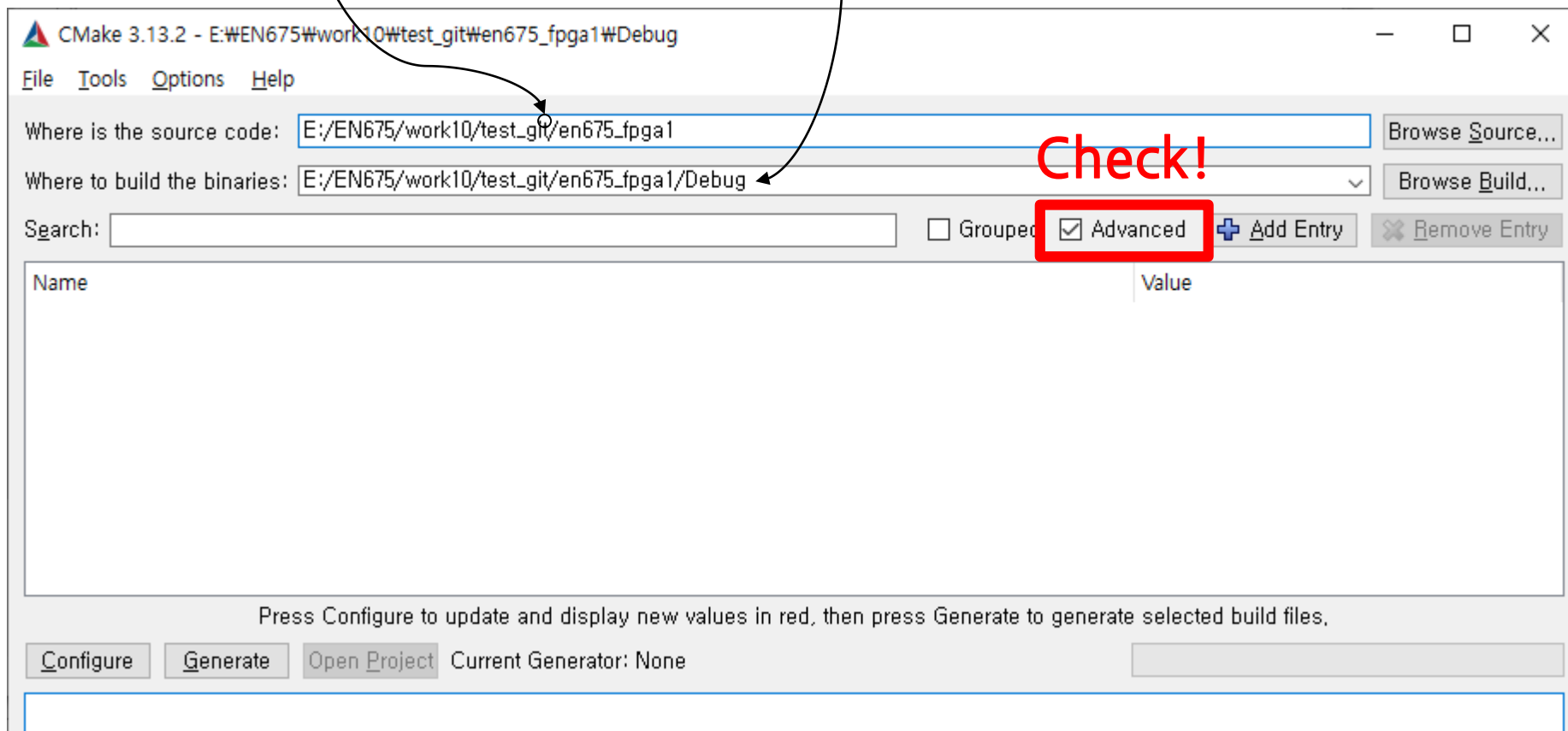
CMakeLists.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말
CMAKE_MINIMUM_REQUIRED ( VERSION 3.13.2 )

# About this project
# -----
PROJECT(EN675_FPGA)
SET(VERSION_MAJOR 0)
SET(VERSION_MINOR "0")
SET(VERSION_PATCH "4")
SET(VERSION "${VERSION_MAJOR}.${VERSION_MINOR}.${VERSION_PATCH}")
    
```

- 다음 페이지를 참조하여 Cmake로 makefile을 생성합니다.

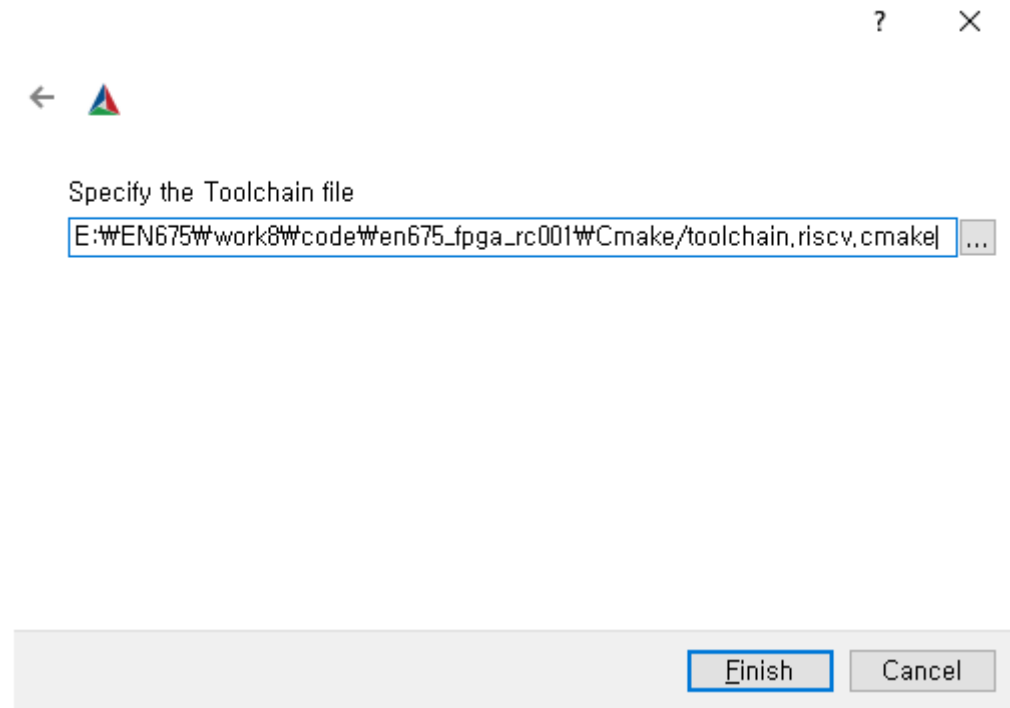
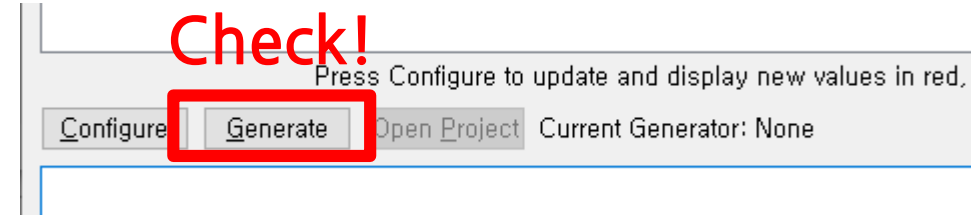
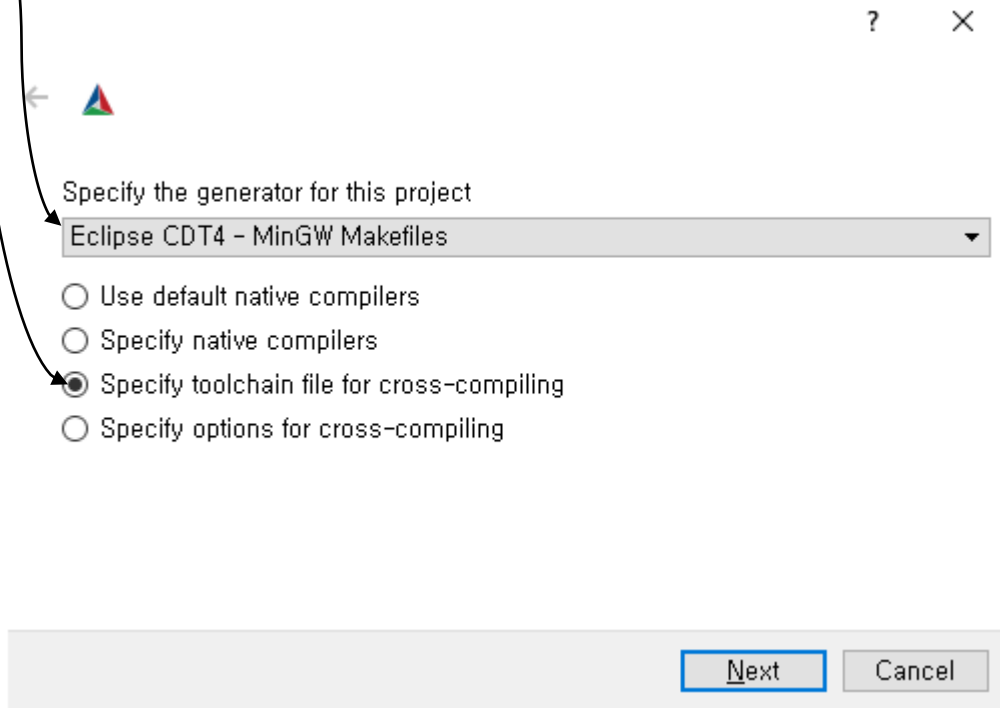
CMake로 makefile 만들기

1. 다운받은 소스코드 경로를 E:\EN675\work10\test_git\en675_fpga1 으로 **가정**합니다.
2. CMake를 실행합니다. Source 경로를 설정합니다.
3. 컴파일 모드에 따라 Build 폴더를 설정합니다. Debug or Release



4. Generate 버튼을 클릭하면, 아래와 같은 창이 나타납니다.

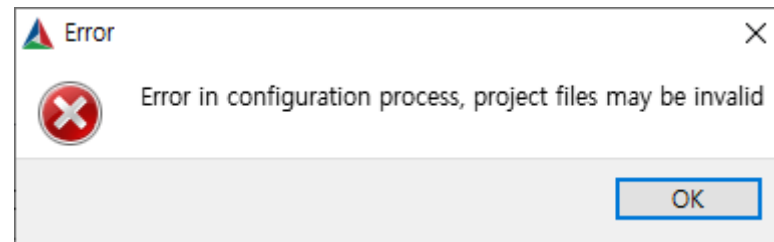
- Eclipse CDT4 – MinGW Makefiles를 선택합니다.
- Specify toolchain file for cross-compiling 를 선택합니다.



5. Toolchain file의 경로를 지정합니다.

- 예시: E:\EN675\work10\test_git\en675_fpga1\Cmake\toolchain.riscv.cmake

6. Finish 버튼을 클릭하면 우측과 같은 오류 메시지가 나타납니다.
OK버튼을 눌러 닫습니다.



7. 우측의 이미지에 있는 붉은 박스에 make.exe 파일의 경로를 설정해야 합니다.

(예시)

CMAKE_MAKE_PROGRAM에는

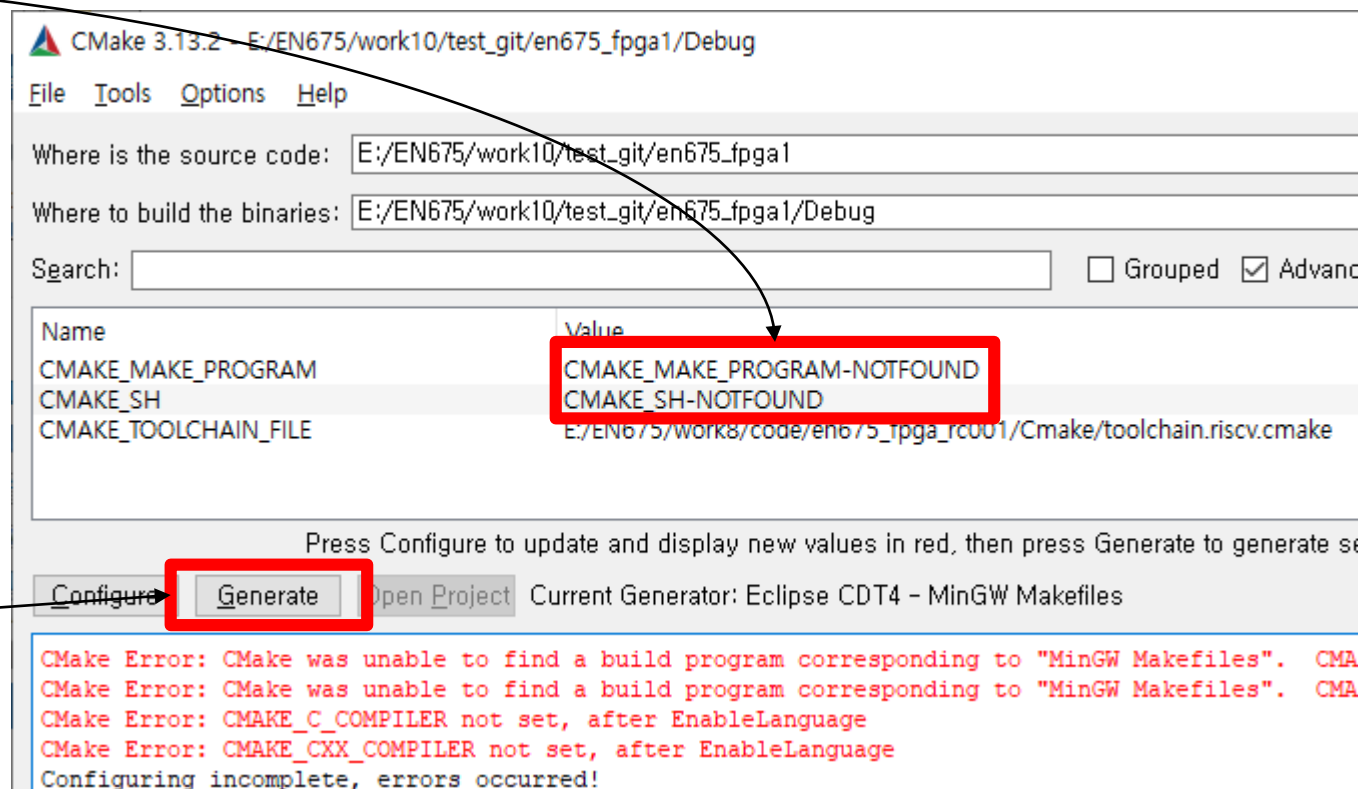
E:\EN675\eclipse\tool\build-tools\bin\make.exe

CMAKE_SH (optional)에는

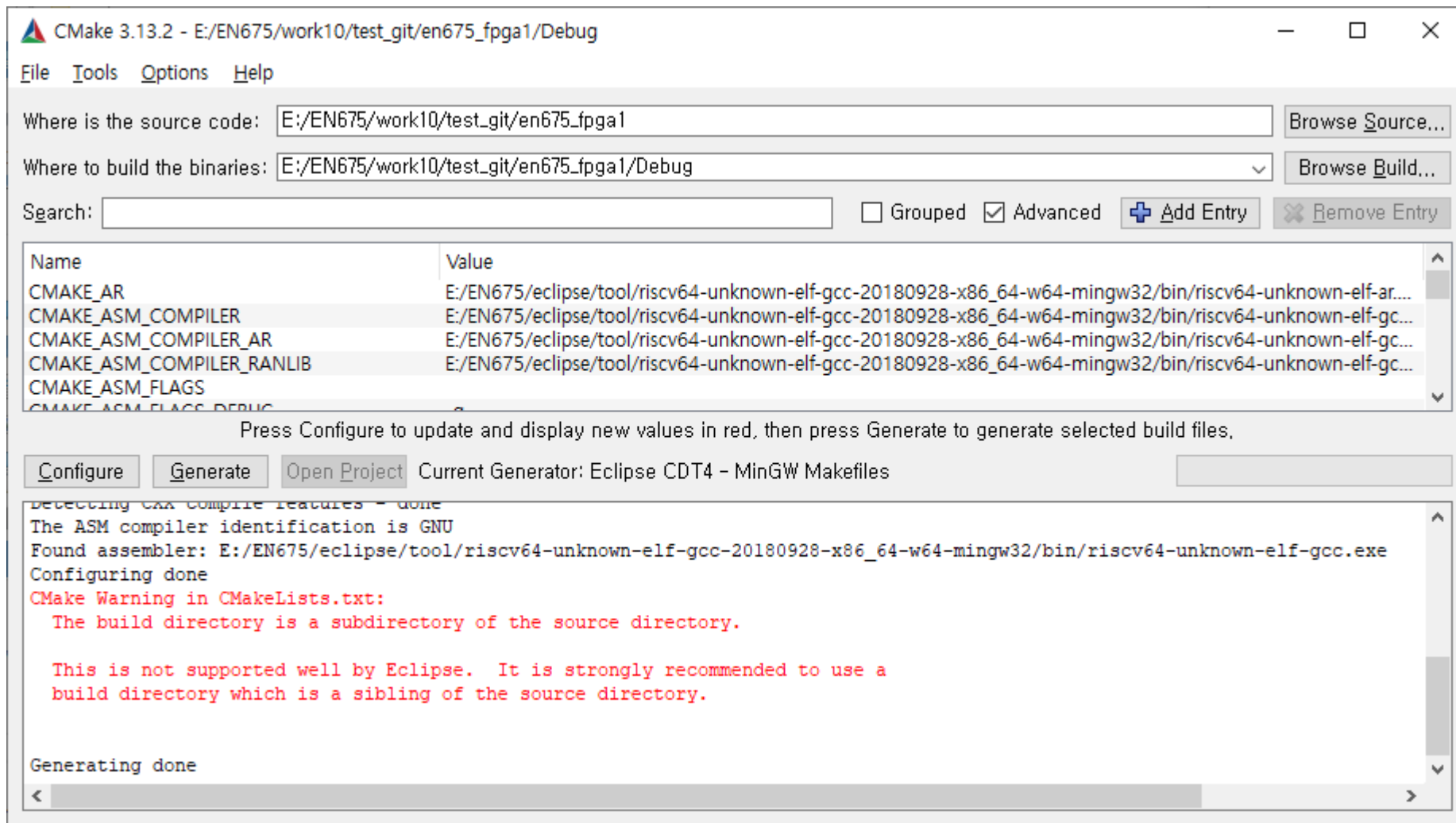
E:\EN675\eclipse\tool\build-tools\bin\sh.exe

를 각각 넣습니다.

8. 다시 한번 Generate를 클릭합니다.



9. 정상적으로 Generate이 되면 아래와 같은 형태로 나타납니다.



1. 소스코드를 압축 해제 합니다.
2. Cmake 폴더의 toolchain.riscv.cmake를 열어서 toolchain의 경로를 수정합니다.

```
toolchain.riscv.cmake - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말
# this one is important
SET(CMAKE_SYSTEM_NAME Generic)

# this one not so much
SET(CMAKE_SYSTEM_VERSION 1)

# prefix of the toolchain path
#SET(TOOLCHAIN_PATH "E:/EN675/eclipse/tool/riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-w64-mingw32/bin/riscv64-unknown-elf-")
SET(TOOLCHAIN_PATH "/home/jhkim/work/EN675/linux/Eyenix_BSP/git_sdk/linux_bsp_EN675/compiler/riscv64-unknown-elf-gcc/bin/riscv64-unknown-elf-")

# specify the cross compiler
#SET(CMAKE_C_COMPILER      ${TOOLCHAIN_PATH}gcc.exe)
#SET(CMAKE_CXX_COMPILER    ${TOOLCHAIN_PATH}g++.exe)
#SET(CMAKE_ASM-ATT_COMPILER ${TOOLCHAIN_PATH}as.exe)
SET(CMAKE_C_COMPILER      ${TOOLCHAIN_PATH}gcc)
SET(CMAKE_CXX_COMPILER    ${TOOLCHAIN_PATH}g++)
SET(CMAKE_ASM-ATT_COMPILER ${TOOLCHAIN_PATH}as)
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8(BOM)

3. Debug 폴더로 이동해서 아래와 같이 명령을 내립니다.

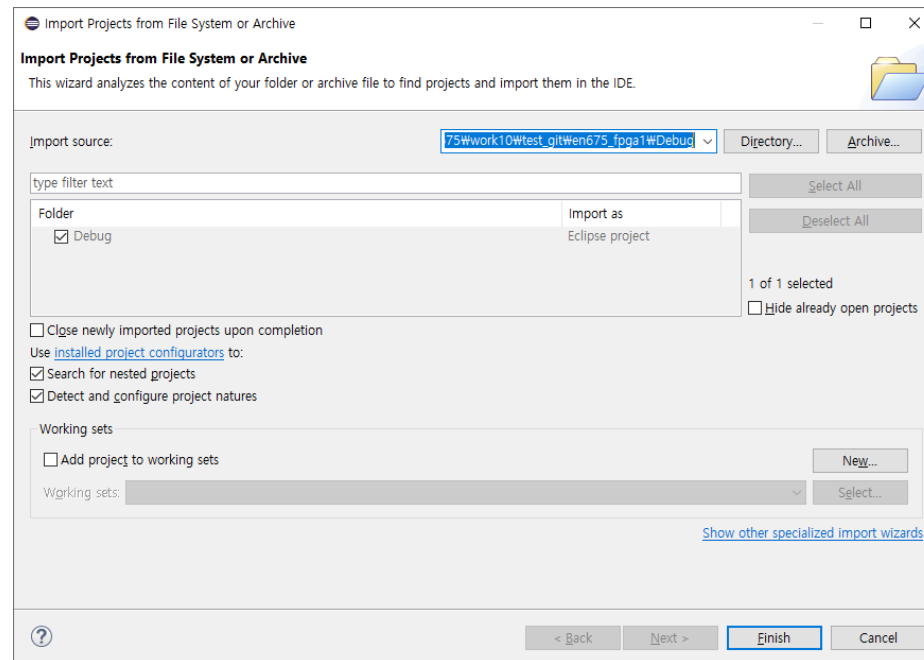
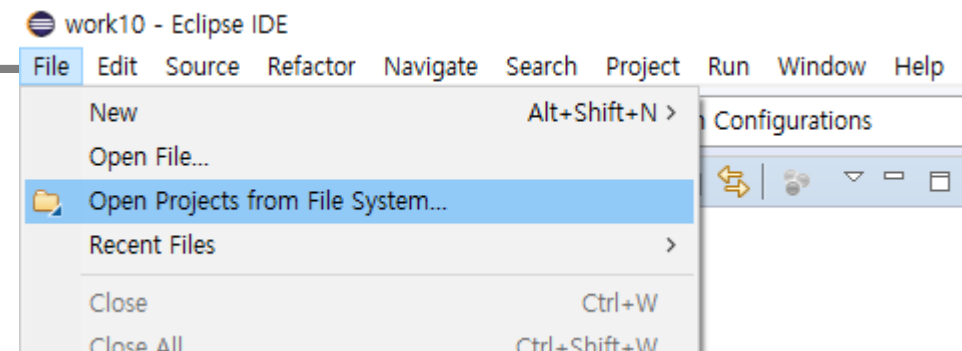
- `cmake .. -DCMAKE_TOOLCHAIN_FILE=../Cmake/toolchain.riscv.cmake`

```
hjee@eyenix-desktop:~/disk/rtos/en675_fpga_git/Debug$ cmake .. -DCMAKE_TOOLCHAIN_FILE=../Cmake/toolchain.riscv.cmake
-- The C compiler identification is GNU 8.3.0
-- The CXX compiler identification is GNU 8.3.0
-- Check for working C compiler: /home/jhkim/work/EN675/linux/Eyenix_BSP/git_sdk/linux_bsp_EN675/compiler/riscv64-unknown-elf-gcc/bin/riscv64-unknown-elf-gcc
-- Check for working C compiler: /home/jhkim/work/EN675/linux/Eyenix_BSP/git_sdk/linux_bsp_EN675/compiler/riscv64-unknown-elf-gcc/bin/riscv64-unknown-elf-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /home/jhkim/work/EN675/linux/Eyenix_BSP/git_sdk/linux_bsp_EN675/compiler/riscv64-unknown-elf-gcc/bin/riscv64-unknown-elf-g++
-- Check for working CXX compiler: /home/jhkim/work/EN675/linux/Eyenix_BSP/git_sdk/linux_bsp_EN675/compiler/riscv64-unknown-elf-gcc/bin/riscv64-unknown-elf-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- The ASM compiler identification is GNU
-- Found assembler: /home/jhkim/work/EN675/linux/Eyenix_BSP/git_sdk/linux_bsp_EN675/compiler/riscv64-unknown-elf-gcc/bin/riscv64-unknown-elf-gcc
-- Configuring done
-- Generating done
-- Build files have been written to: /home/hjee/disk/rtos/en675_fpga_git/Debug
hjee@eyenix-desktop:~/disk/rtos/en675_fpga_git/Debug$
```

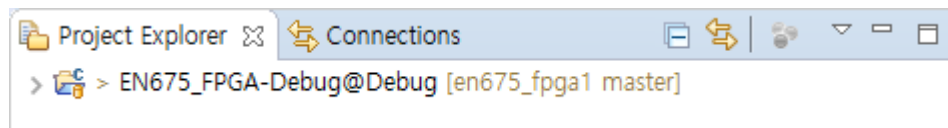

4. make 명령을 내리면 컴파일이 진행됩니다.

```
[ 94%] Building C object Source/core/core0/CMakeFiles/core0.dir/task/videnc/videnc_info.c.obj
[ 94%] Building C object Source/core/core0/CMakeFiles/core0.dir/task/wifi/wifi.c.obj
[ 95%] Building C object Source/core/core0/CMakeFiles/core0.dir/task/wifi/wifi_task.c.obj
[ 95%] Linking C static library ../../../../lib/libcore0.a
[ 95%] Built target core0
Scanning dependencies of target core1
[ 96%] Building C object Source/core/core1/CMakeFiles/core1.dir/main1.c.obj
[ 96%] Linking C static library ../../../../lib/libcore1.a
[ 96%] Built target core1
Scanning dependencies of target core2
[ 97%] Building C object Source/core/core2/CMakeFiles/core2.dir/main2.c.obj
[ 97%] Linking C static library ../../../../lib/libcore2.a
[ 97%] Built target core2
Scanning dependencies of target core3
[ 97%] Building C object Source/core/core3/CMakeFiles/core3.dir/main3.c.obj
[ 98%] Linking C static library ../../../../lib/libcore3.a
[ 98%] Built target core3
Scanning dependencies of target core.elf
[ 98%] Building C object Source/core/shared/CMakeFiles/core.elf.dir/boot.c.obj
[ 98%] Building C object Source/core/shared/CMakeFiles/core.elf.dir/pmp.c.obj
[ 99%] Building C object Source/core/shared/CMakeFiles/core.elf.dir/exbl.c.obj
[ 99%] Building C object Source/core/shared/CMakeFiles/core.elf.dir/msg.c.obj
[100%] Building C object Source/core/shared/CMakeFiles/core.elf.dir/user.c.obj
[100%] Building ASM object Source/core/shared/CMakeFiles/core.elf.dir/mentry.S.obj
[100%] Linking C executable ../../../../core.elf
[100%] Built target core.elf
hjlee@eyenix-desktop:~/disk/rtos/en675_fpga_git/Debug$
```

1. File > Open Projects from File System... 을 선택합니다.
2. Import source에 CMake에서 설정한 Build 폴더를 선택합니다.
(예시) E:\EN675\work10\test_git\en675_fpga1\Debug
3. Finish 버튼을 클릭합니다.



4. Project Explorer에 추가되었습니다.



1. Project Explorer을 펼치면 우측의 이미지와 같은 형태로 나타나게 됩니다.

2. Build Targets

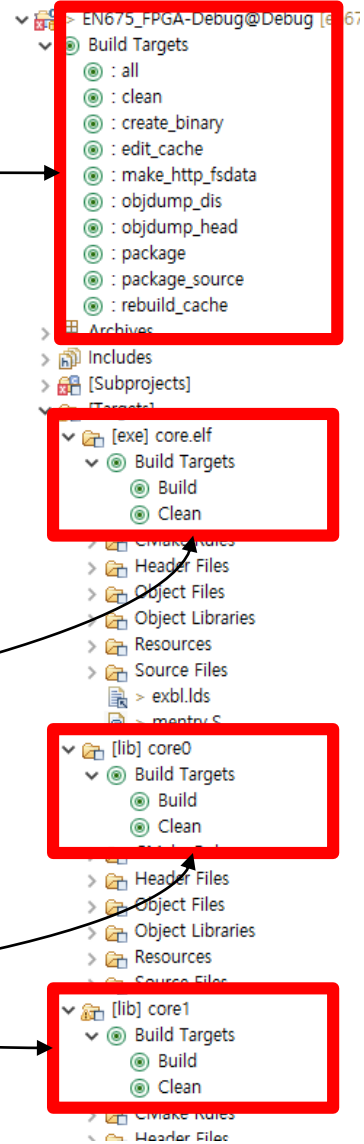
- all: make all을 의미합니다. 모든 target들을 compile 합니다.
- clean: make clean을 의미합니다. 모든 target들을 clean 합니다.
- create_binary: core.elf파일을 binary(core.bin)로 바꾸고 boot.bin + core.bin하여 EN675.bin을 생성.
- edit_cache: 현재 project에 대하여 cmake 를 실행합니다.
- make_http_fsdata: HTTP server에서 사용하는 HTML file을 c file로 만듭니다.
- objdump_dis: elf file의 sections, symbol table, disassembly 정보를 core.dump에 저장합니다.
- objdump_head: elf file의 sections 정보를 화면에 출력합니다.
- Package: 사용X
- Package_source: 소스코드를 압축합니다.
- 결과물 E:/EN675/work10/test_git/en675_fpga1/Debug/en675_fpag_rc001_190207163647.zip 와 같은 형태.

3. (exe) Build Targets – core.bin

- Build: 실행 가능한 binary를 생성합니다. library 수정시 해당 library도 compile 합니다.
- Clean: 해당하는 obj file들을 clean 합니다.

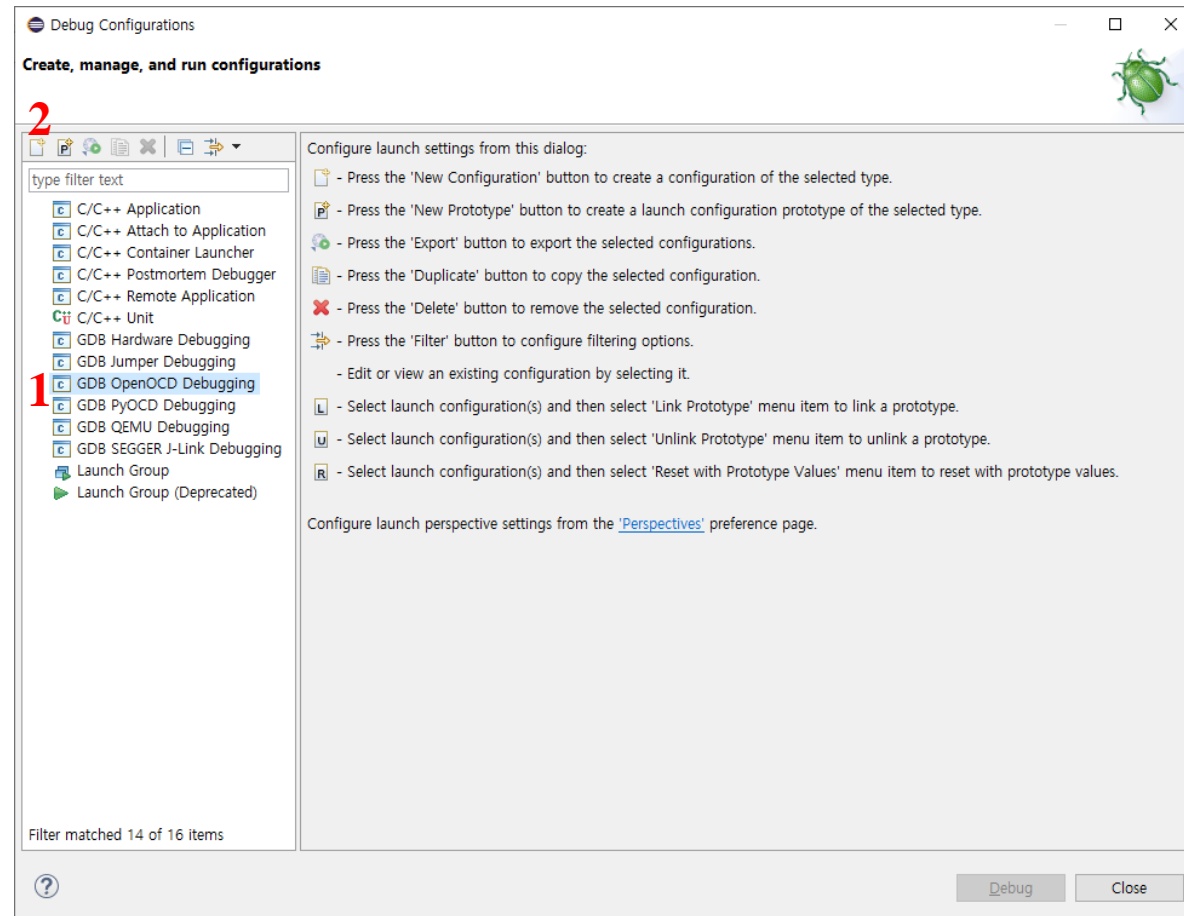
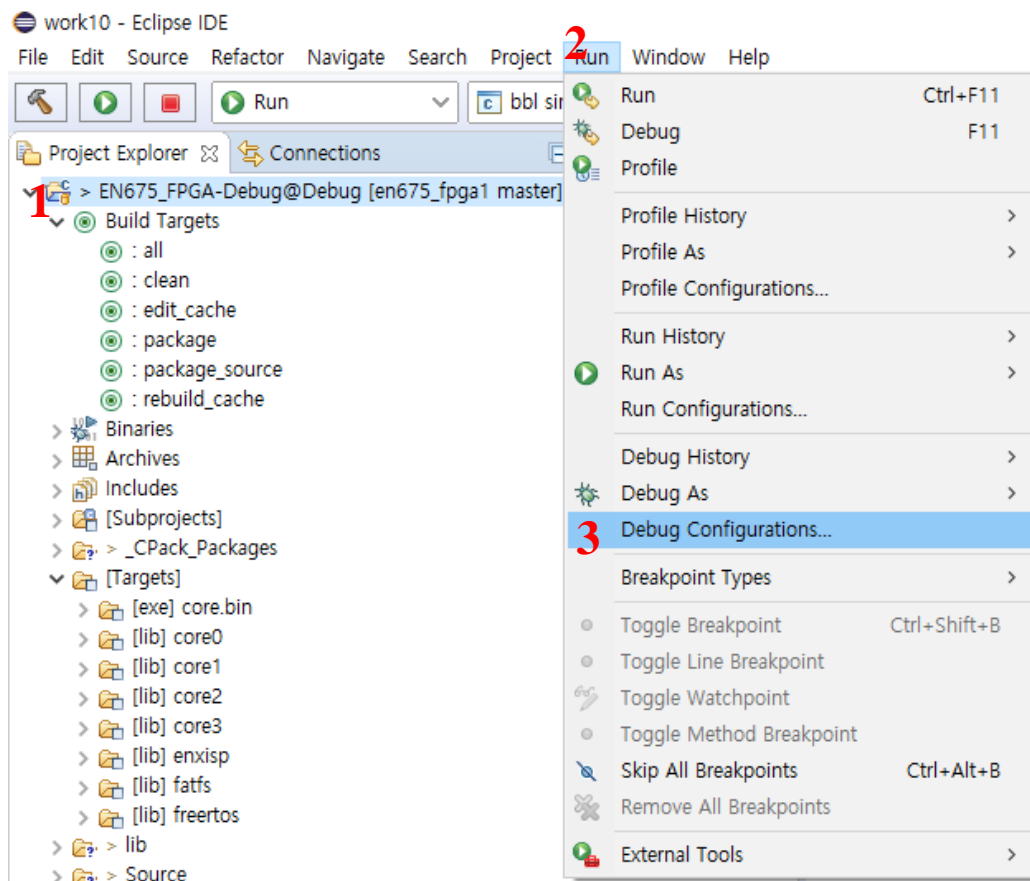
4. (lib) Build Targets – core0, core1, core2, core3, enxisp, fatfs, freertos ...

- Build: 해당하는 library를 compile 합니다.
- Clean: 해당하는 library와 obj file들을 clean 합니다.



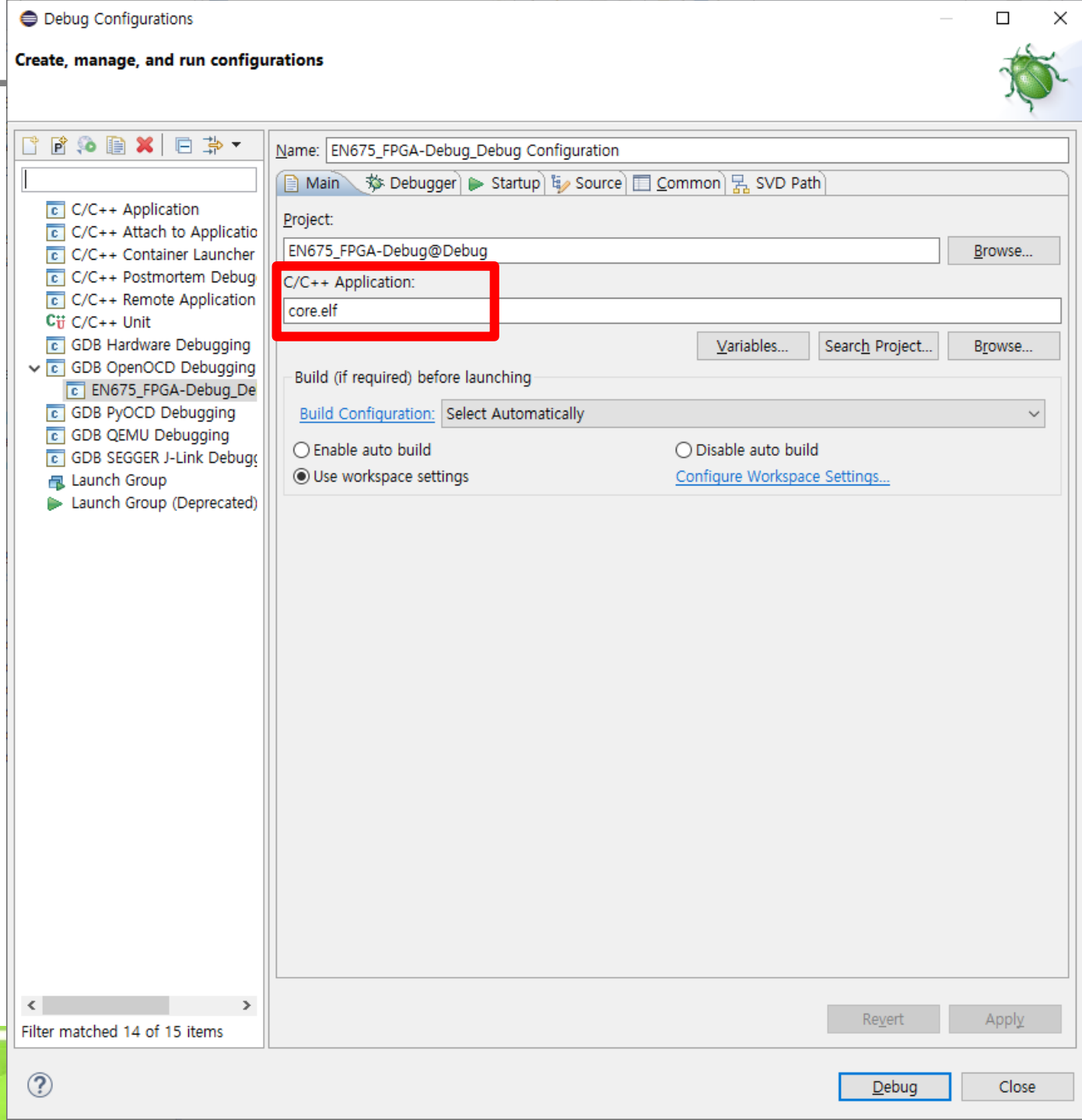
Debug Configurations

1. 해당 프로젝트를 선택하고, Run > Debug Configurations... 을 선택합니다.
2. GDB OpenOCD Debugging를 선택하고, New Configuration를 클릭합니다.



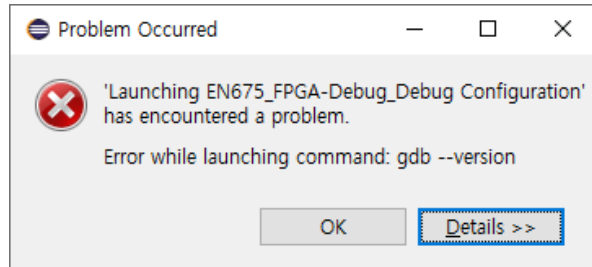
3. Main 메뉴의 상태를 확인합니다.

- C/C++ Application
core.elf



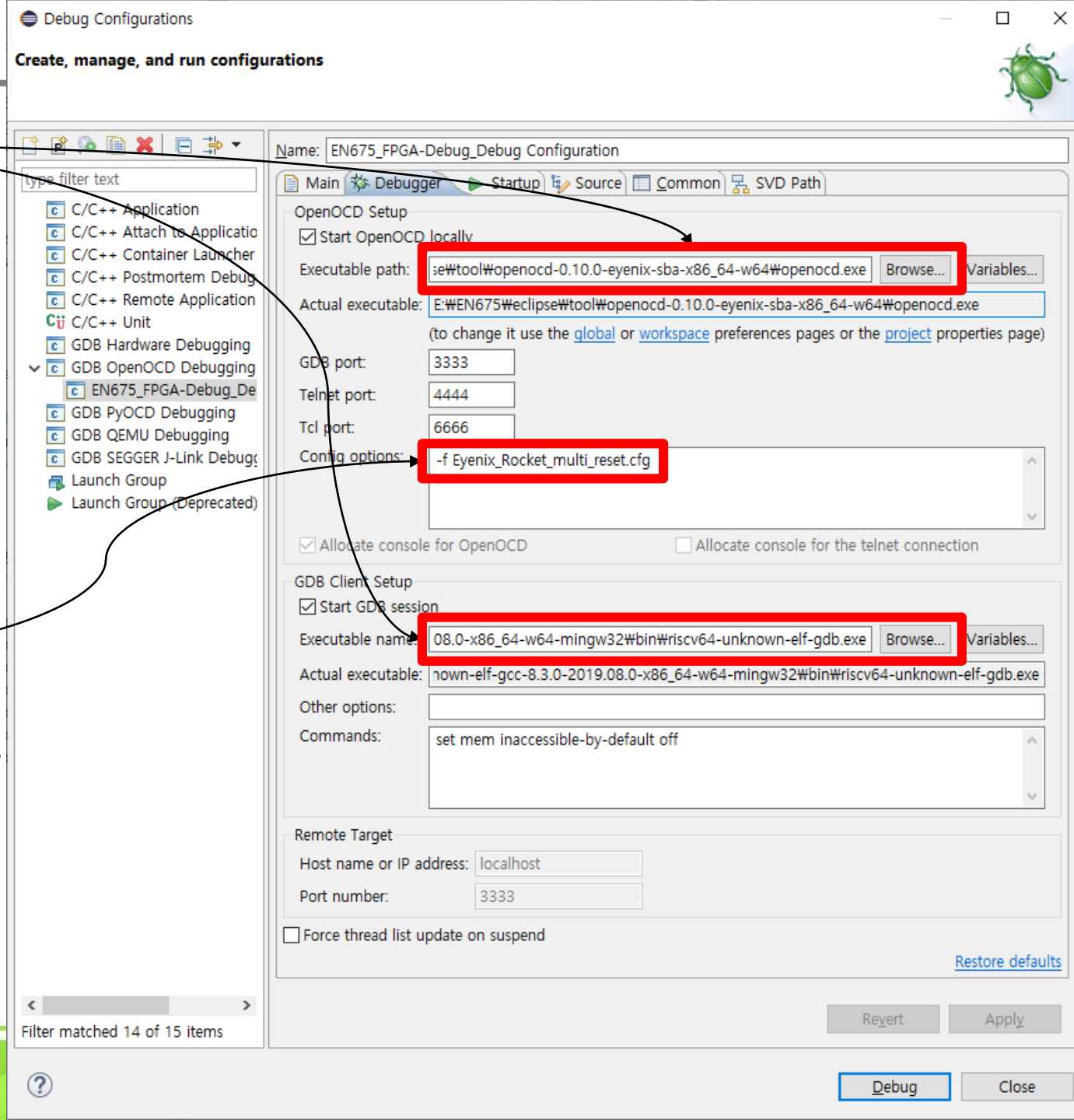
4. Debugger 메뉴의 상태를 확인합니다.

- openocd와 gdb가 올바른 경로인지 확인합니다.
- 잘못된 설정일 경우 아래와 같은 메시지가 나타납니다.



5. Config options에 디버깅 할 cfg 파일을 선택합니다.

- Eyenix_Rocket_multi.cfg: 모든 코어 사용
- Eyenix_Rocket_multi_reset.cfg: 모든 코어 사용
- Eyenix_Rocket_single_0.cfg: CPU0 사용
- Eyenix_Rocket_single_1.cfg: CPU1 사용
- Eyenix_Rocket_single_2.cfg: CPU2 사용
- Eyenix_Rocket_single_3.cfg: CPU3 사용

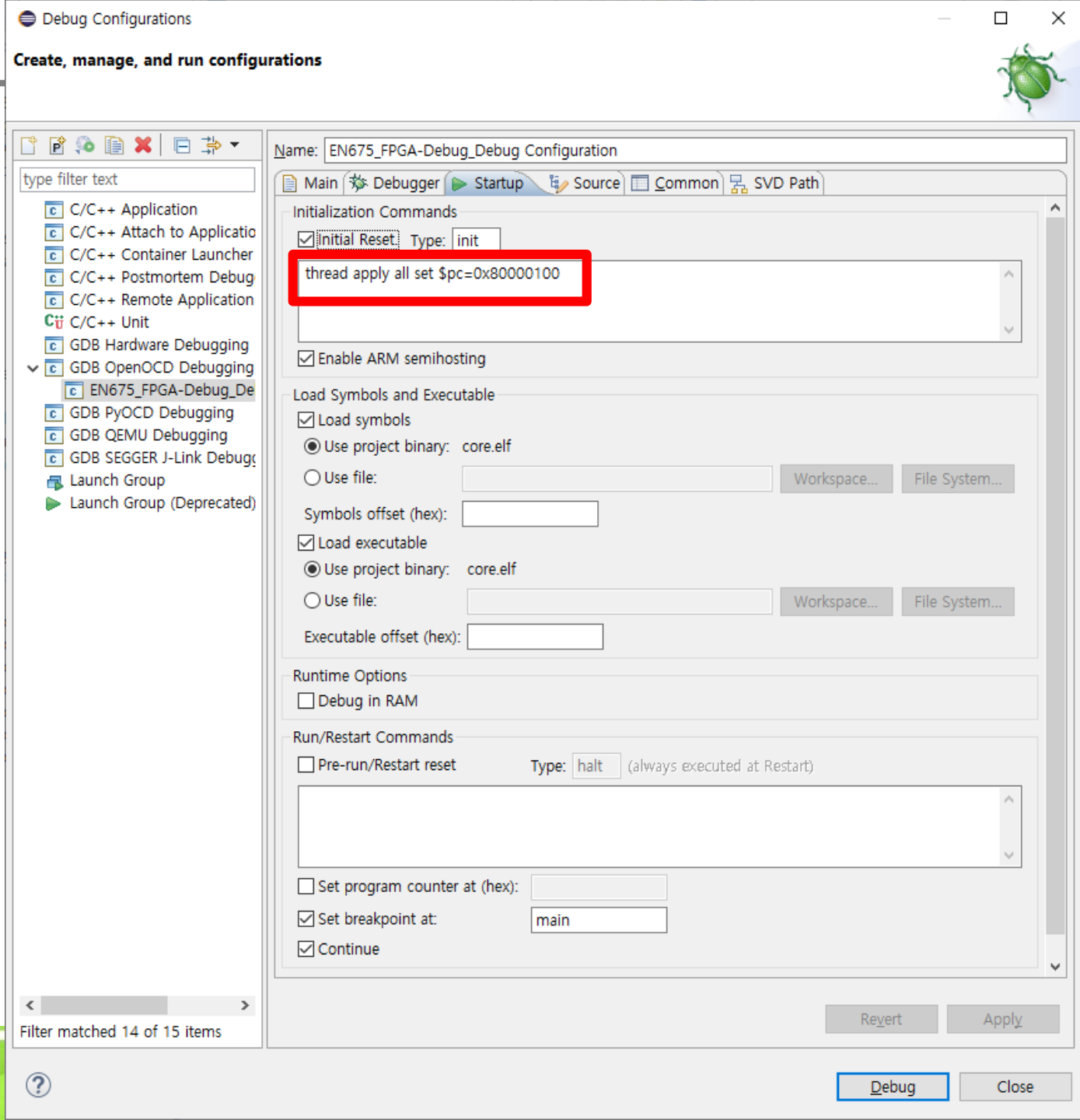


6. Startup 메뉴의 상태를 확인합니다.

- Initialization Commands
thread apply all set \$pc=0x80000100

7. Debug를 클릭하면 디버깅을 시작합니다.

- Debug/Run을 하면 main_0 함수에서 break 됩니다.
- F8키를 누르면 run 합니다.



Version	Date	Description	Modified by
0.1	2019.02.11.	Initial release	HJ Lee
0.2	2019.05.27.	Document review	HJ Lee
0.3	2019.11.05.	DDR 초기화 관련 Debug Configurations 수정	HJ Lee
0.4	2019.11.07.	Build Targets, Debug Configurations 수정, 문서 통합	HJ Lee
0.5	2020.01.15.	Debug Configurations 수정	HJ Lee
0.6	2020.03.19.	복수의 Procjet 등록 방법 추가, Eclipse 실행 / 시작화면 수정	HJ Lee