STM32 Cross-Building Environment

Confidentiality Level: (Tick the Box ■)		
Top Secret □	Confidential □	Public ■

Document Control Records

Revision History			
Date	Revision	Revision Description	Author
2023-08-15	0.0	Initial	Eddy Mao
2023-10-11	0.1	Update Section 5.3.5 Multiple Host Operation Systems Support	Eddy Mao

Contents

1 Purpose

This document explains the procedures how to generate the source code with ST semiconductor's IDE and how to set up a cross compiling environment, which enabled the consistent integration within a software development team.

2 Scope

This document applies to all developers of STM32 microcontroller ecosystem.

3 Introduction of STM32

STMicroelectronics (ST) is a multinational semiconductor manufacturer headquartered in Geneva, Switzerland.

One of ST's most popular product lines is the STM32 series of microcontrollers. STM32 microcontrollers are based on ARM Cortex-M processors and are widely used in a diverse range of applications, from consumer electronics to industrial automation and beyond. The STM32 family is known for its high-performance, energy efficiency, and extensive peripheral integration.

STM32 microcontroller has a few product families. This document will take entry-level MCU (EVB) STM32L073RZT6 as an example.

STMicroelectronics provides a robust ecosystem for development, including development boards, software libraries, integrated development environments (IDEs), and support resources for engineers and developers working with STM32 microcontrollers.
Following page tells more,
STM32 Microcontrollers (MCUs) - STMicroelectronics

4 Responsibilities

Quectel engineering team develops sample codes for user-friendly module application purpose. The reference code are based on STM32 microconrollers but all the source code generating, compiling and relevant toolchains are not Quectel pripriatory. Quectel may not be responsible for any risks and technical issues in applications based on this document.

5 Source Code Generation

5.1. Toolchain

ST has two graphic tools or IDE: ST CubeMX and CubeIDE. Basically CubeIDE can be viewed as CubeMX + graphic Builder & Debugger. As a result, CubeMX is sufficient enough to generate the source code. However, the CubeIDE would be highly recommended to be installed on developer's PC; so all following

statement and examples will be based on CubeIDE *[document 3]*, instead of a light CubeMX.

5.2. Code Generation

5.2.1. Create a new project

in CubeIDE (Figure 1). Select “STM32 Project”.

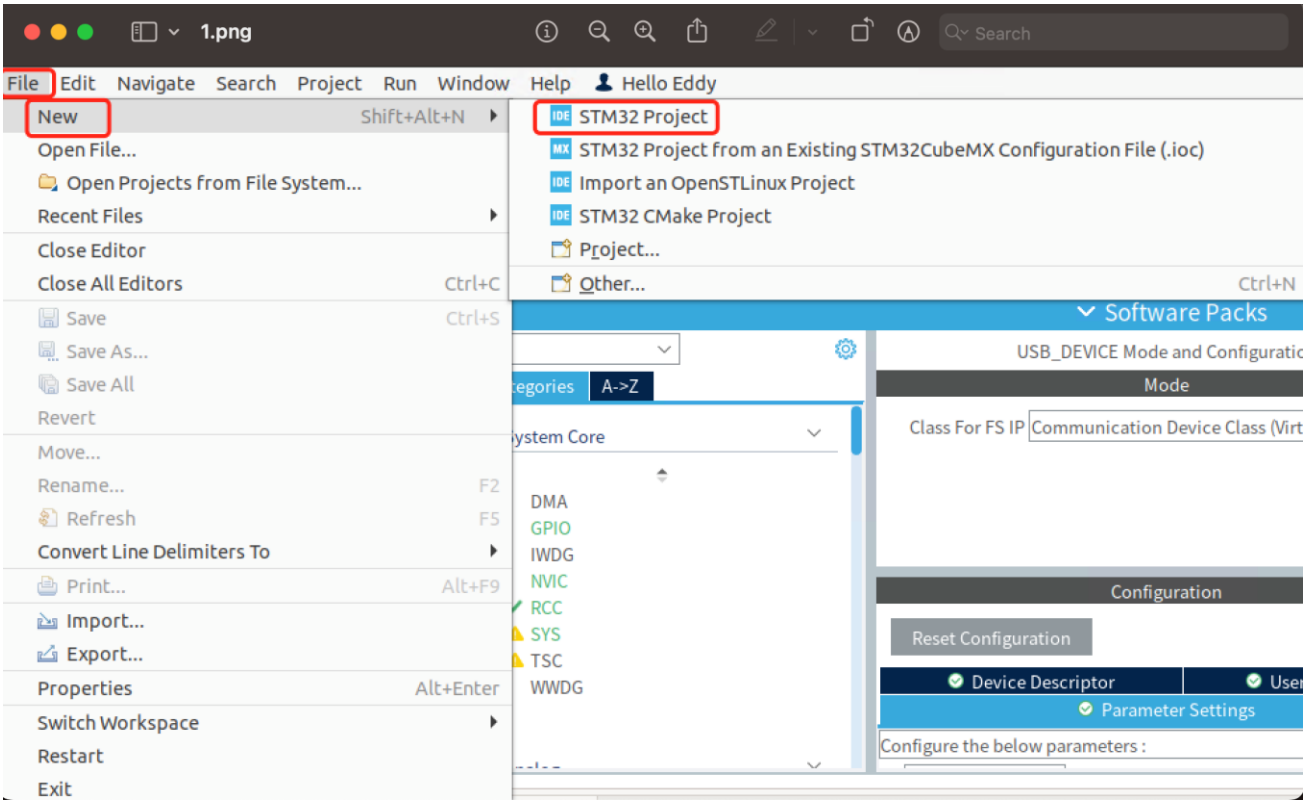


Figure 1 - Create a New Project

5.2.2. MCU selector

Then in the “MCU/MPU Selector” menu, key in the MCU part no. “STM32L073RZ”, the optional part Number will show up. Select the one that matches.

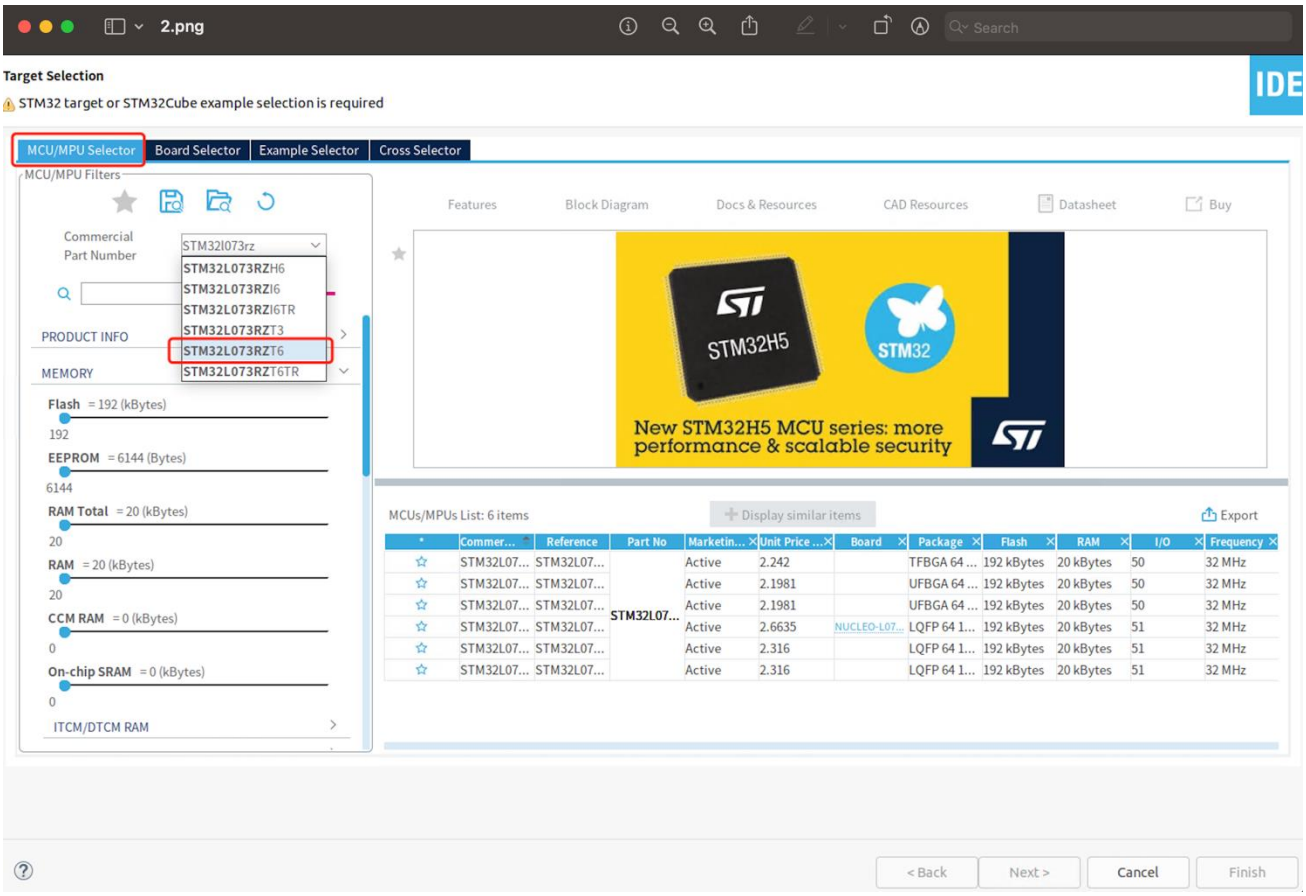


Figure 2 - MCU Selector

Left click the MCU Item from the list until the item is highlighted.

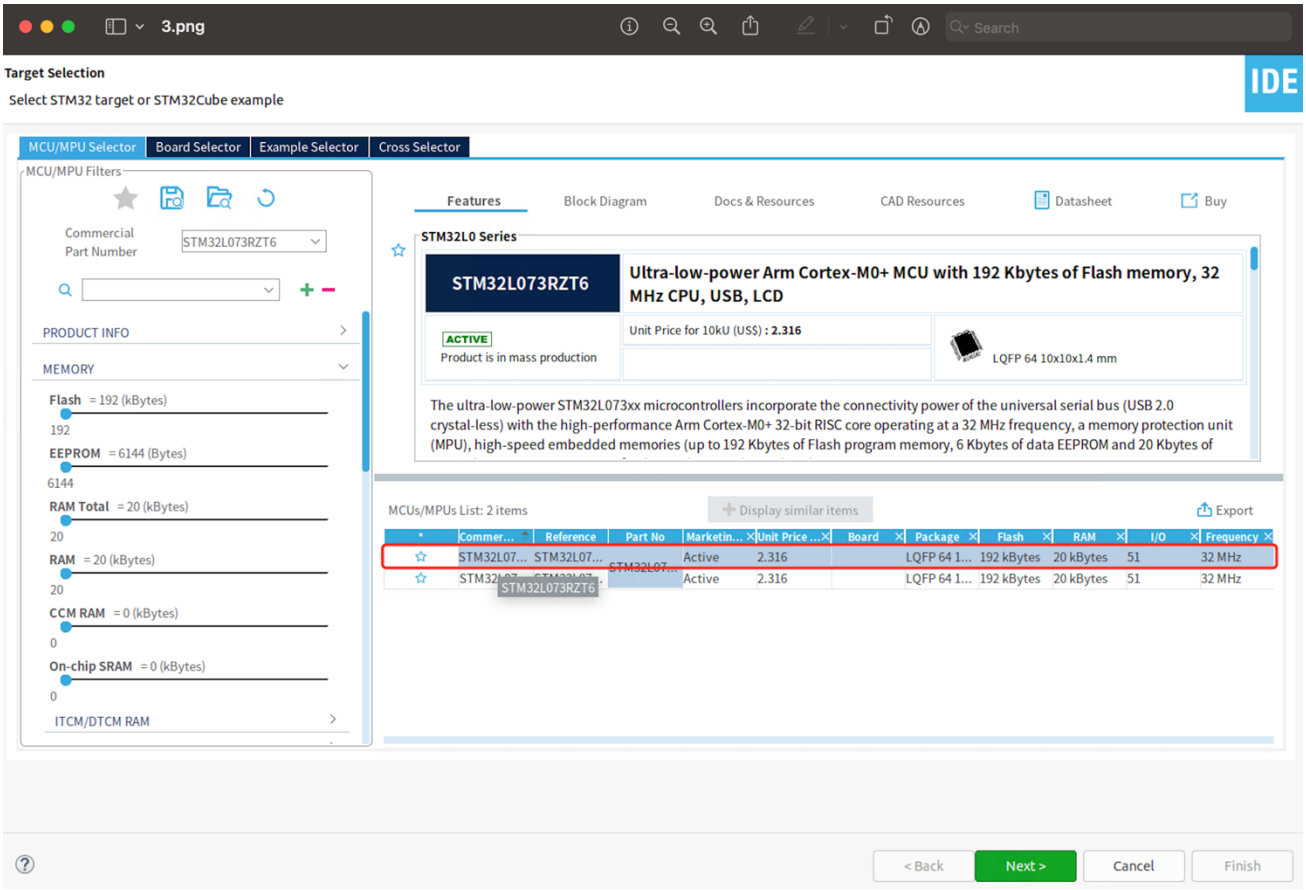


Figure 3 - Select the Item from the list

5.2.3. Board Selector

Select the board (EVB) which needs to match the hardware setup.

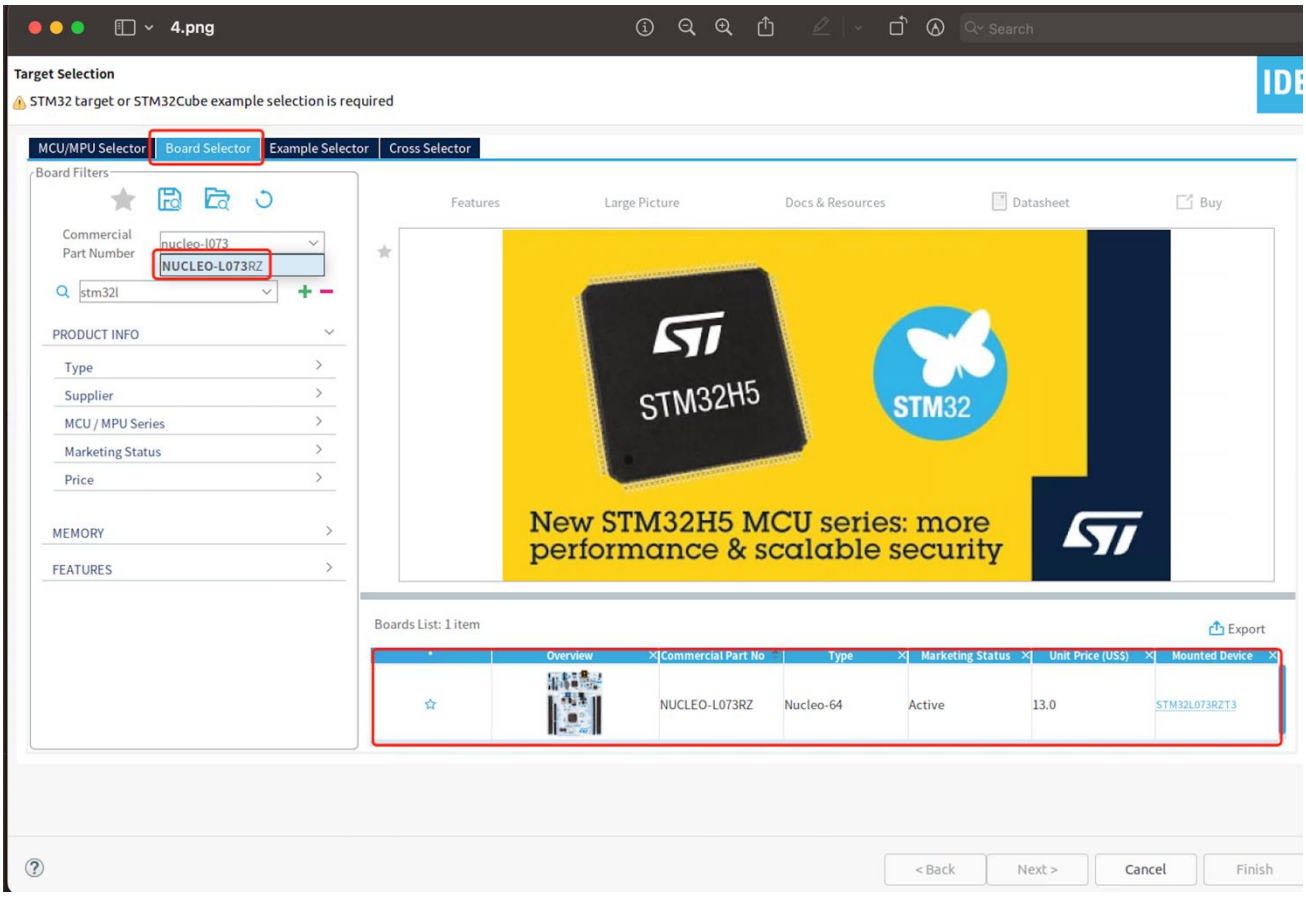


Figure 4 - Board Selector

You may key in the board series name then click the item from the board list. The item has to be highlighted by click.

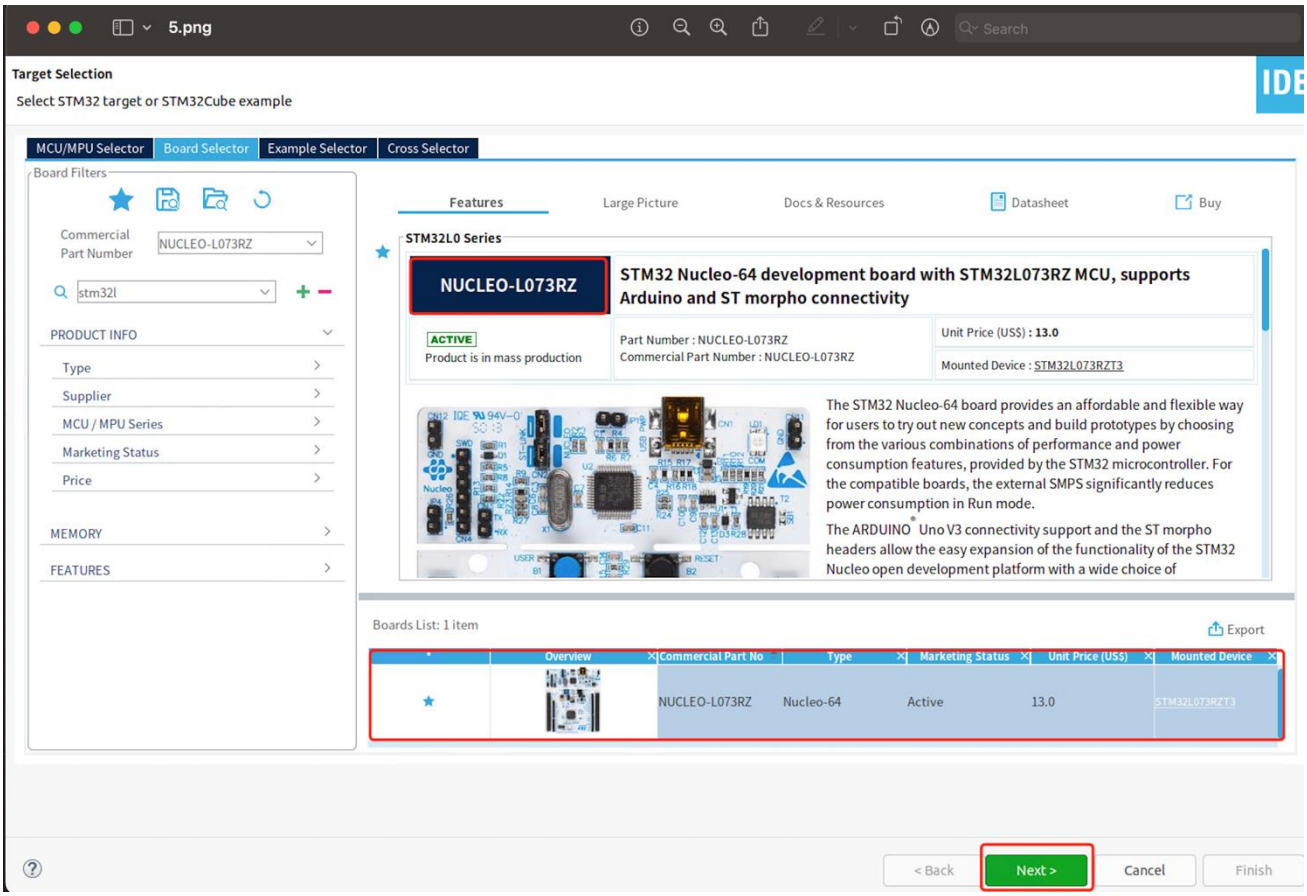


Figure 5 - Highlight by clicking the item

5.2.4. Name the Project

Click the button “Next” then it shows up the Project Setup window. Key in the name of the “project”, following checkboxes can be set by default.

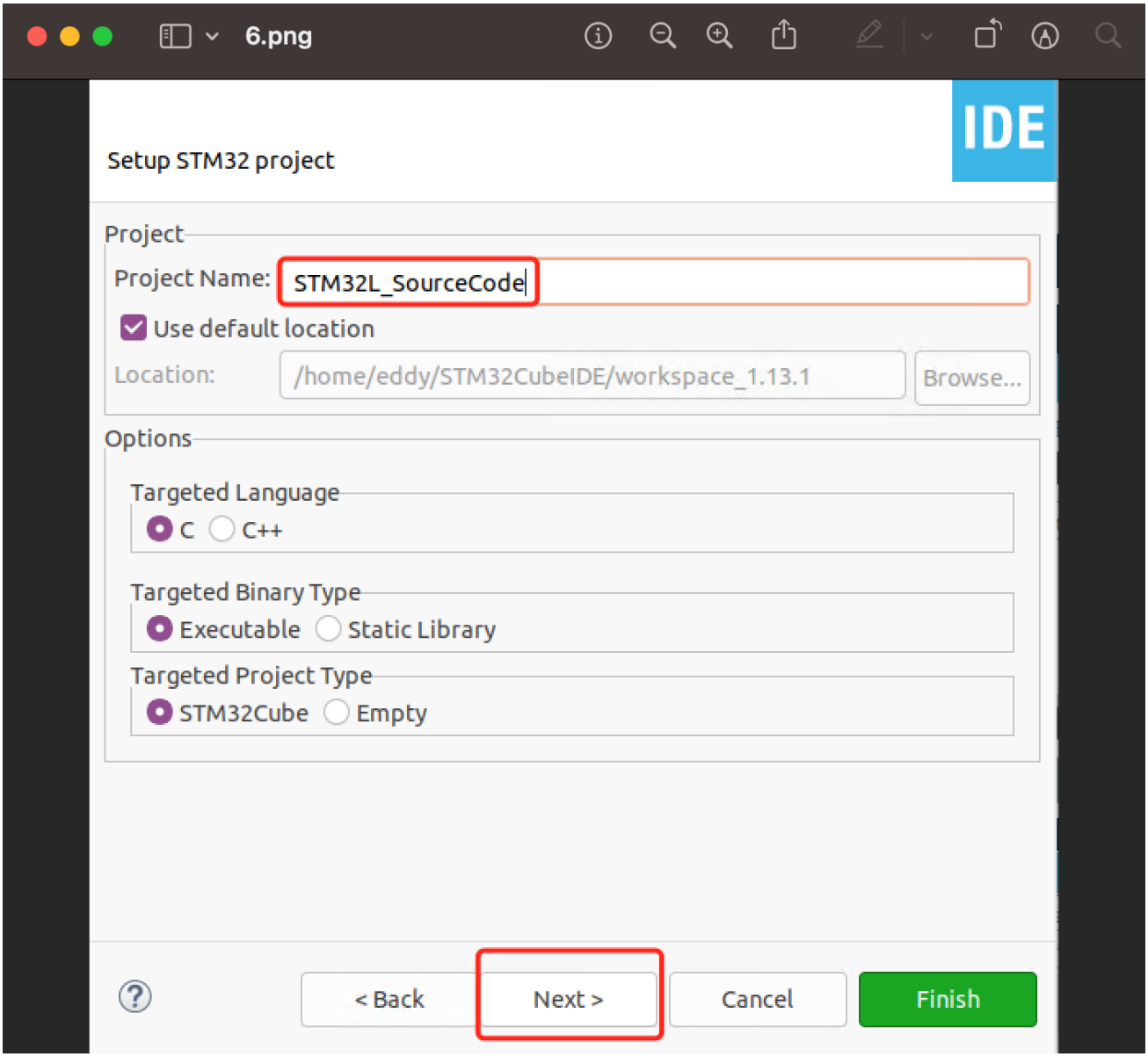


Figure 6 - Setup Project

5.2.5. Configure Code Generation Option

In the “Firmware Library Setup” Window, check “add necessary library files” to avoid the reference of file link in generated source files.

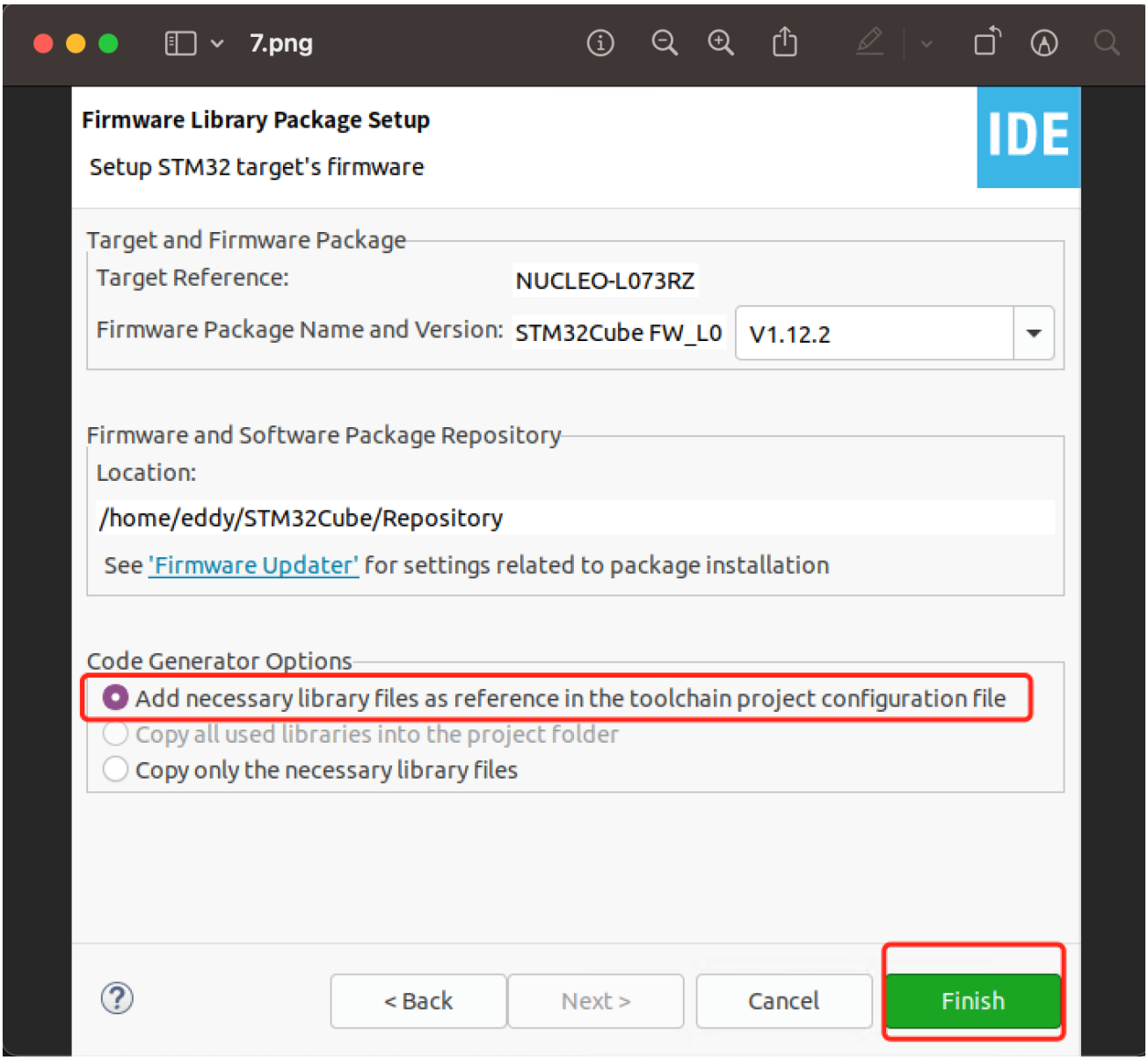
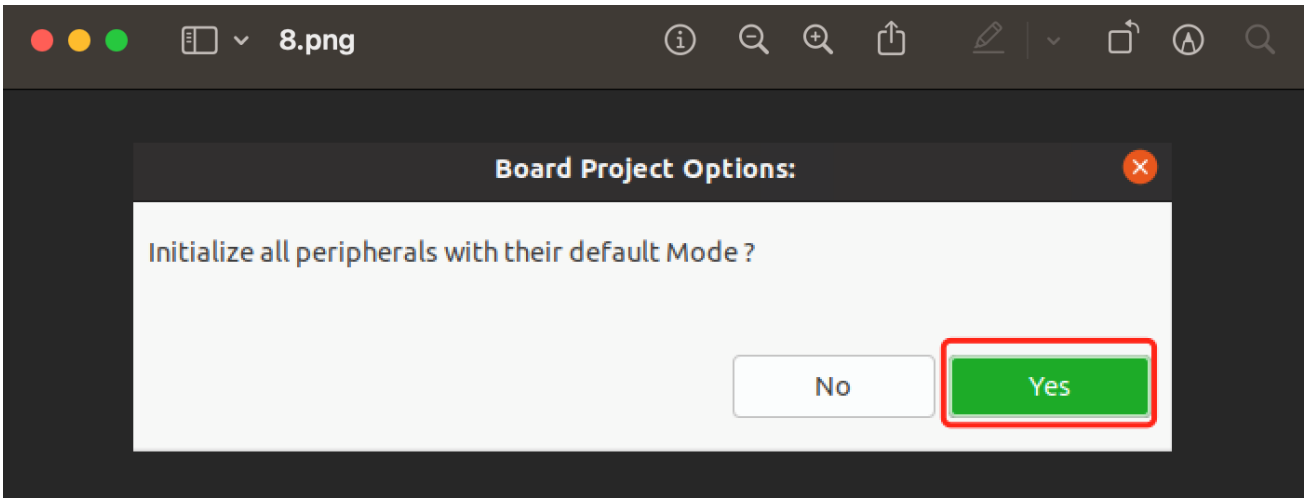


Figure 7 - Code Generator Options

Basically the other peripheral options should be customized for the moment. To make the procedure simplier, these options are skipped for this document. Confirm the default setting by clicking “Yes”.



5.2.6. Project Cinfiguration - more

Now the new project “STM32L_SourceCode” can be observed, with a few configuration options for “Pinout & Configuration”, “Clock Configuration”, “Project Manager” and “Tools”. You may refer to *[Document 2]* for details.

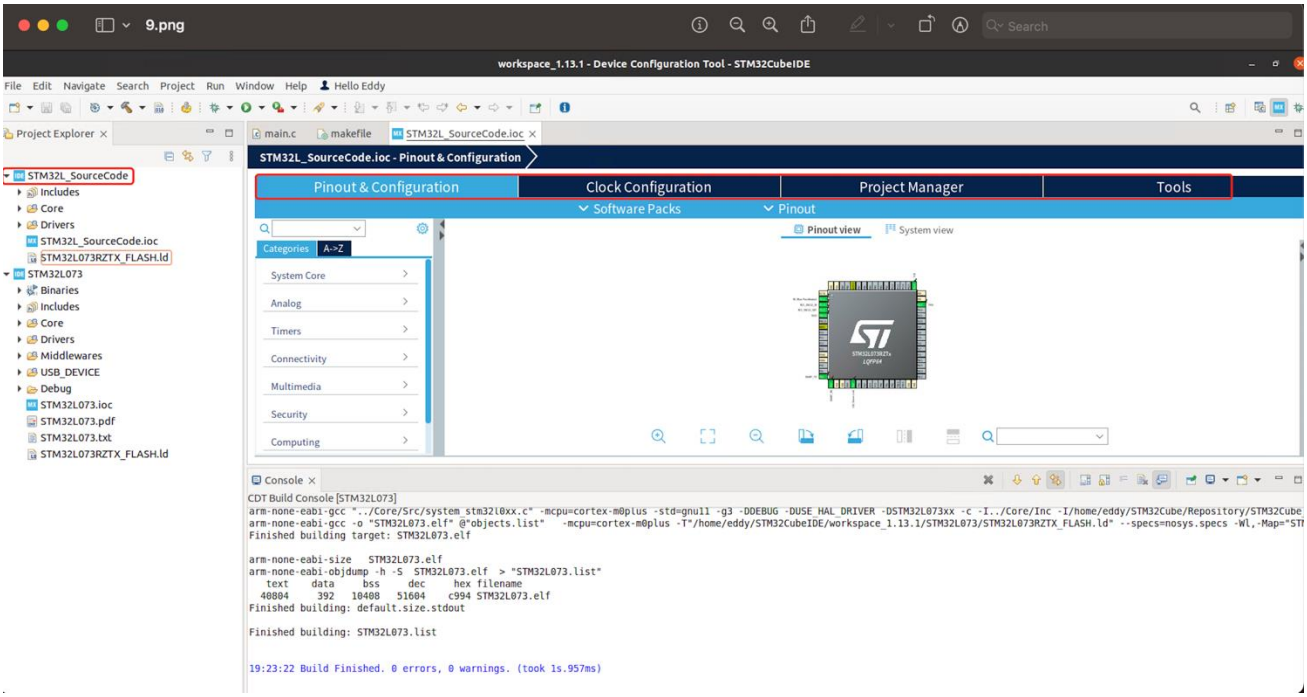


Figure 8 - Project Configuration

5.2.7. Disable “Cyclomatic Complexity” compiling flag

Right click the project Name, e.g. “STM32L_SourceCode”, then select “Properties”.
In sub menus “C/C++ Build” / “Settings” / “Tool Settings” / “MCU/GCC compiler” / “Miscellaneous” / “Other Flags”, uncheck item “Cyclomatic Complexity”, as GNU ARM GCC compiler does not support

this flag. Otherwise, you will run to compiling error in cross-compiling.

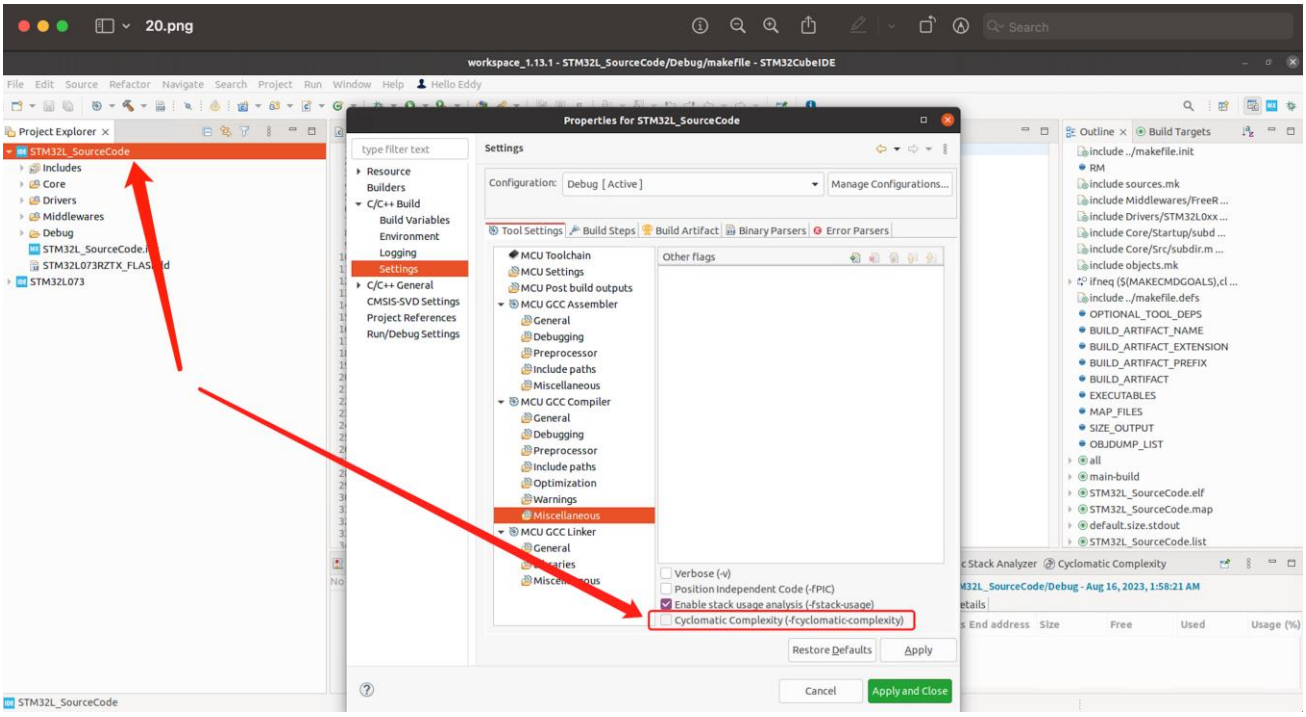


Figure 9 - Uncheck GNU GCC non-supported flag

5.2.8. Features enabler – middleware for FreeRTOS

Back to the “Pinout Configuration” menu. Let us forget the Pin assignment for the moment (This has to be done, according to the hardware connection). What needs to be mentioned is, how to enable the RTOS support? This can be activated in “Configuration” / “Middleware and Software Packs” / FREERTOS. In this project example, a light CMSIS_V1 version is selected – the RTOS infrastructure can be observed in below figure.

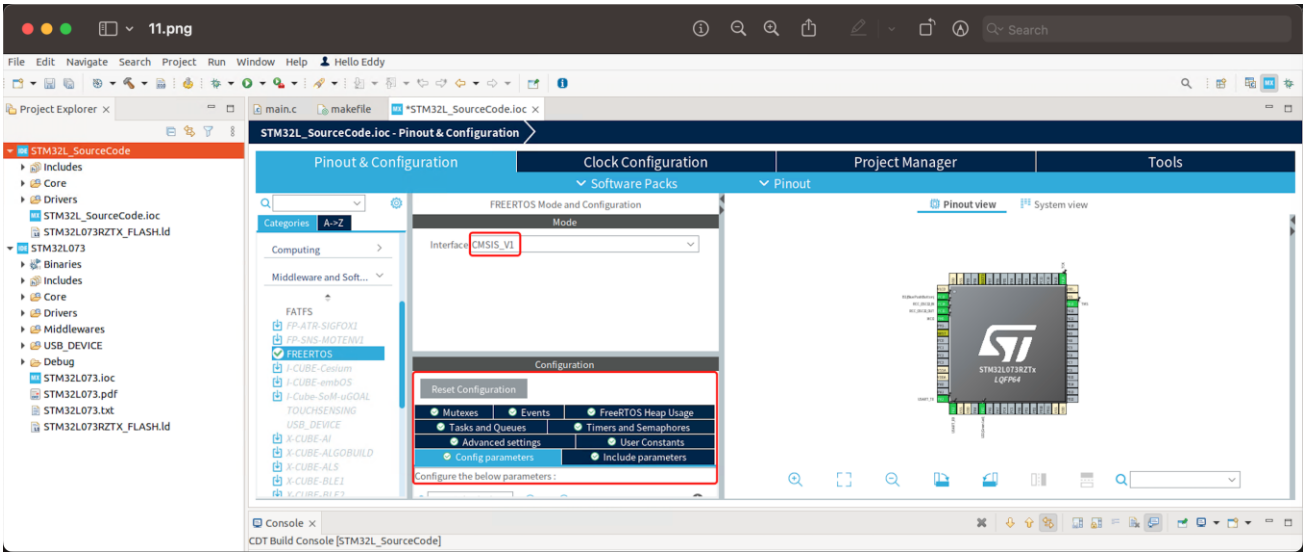


Figure 10 - Enable FreeRTOS

5.2.9. Generate Code

Now the source for the project can be generated (with all peripherals configuration set as default). Browse to “Project” / “Generate Code”.

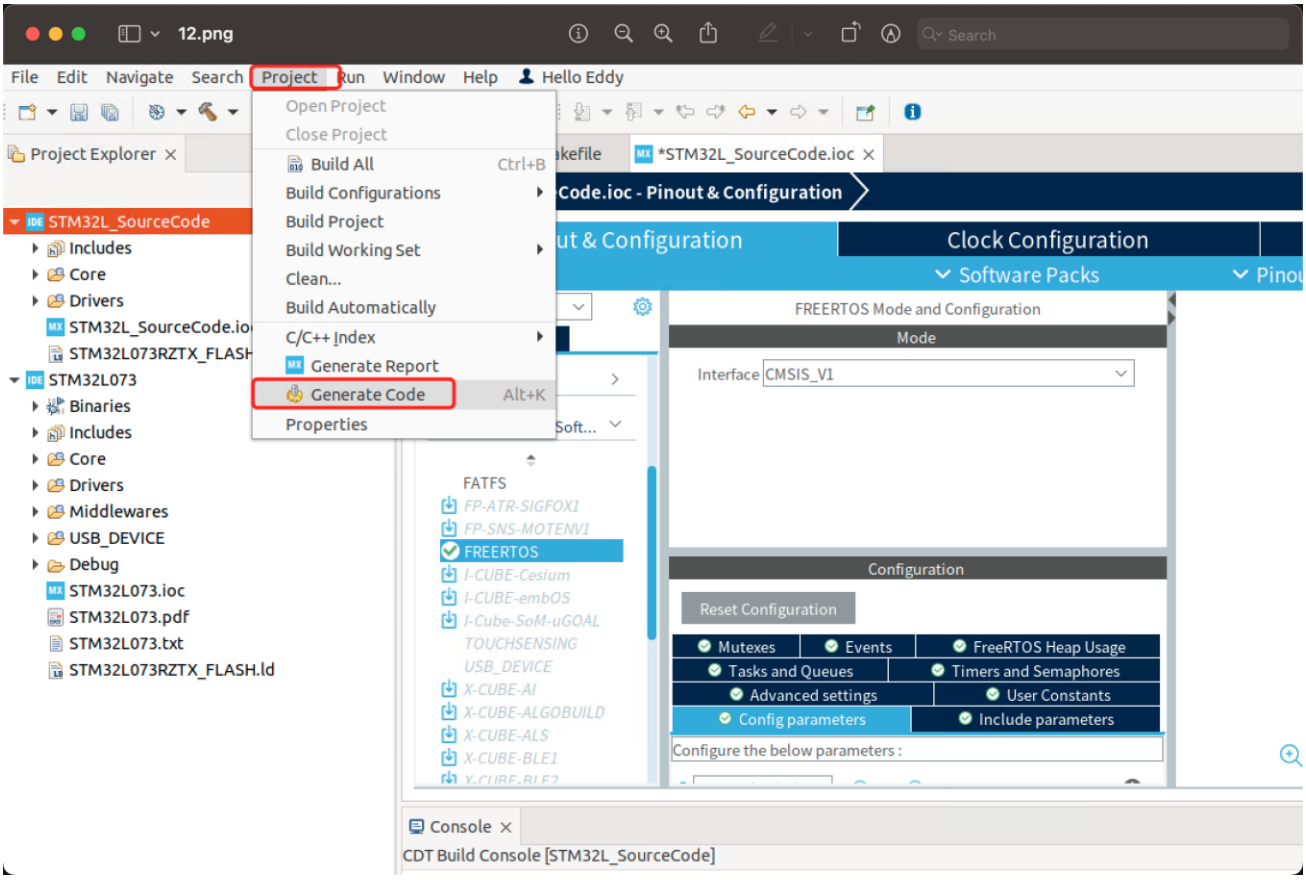
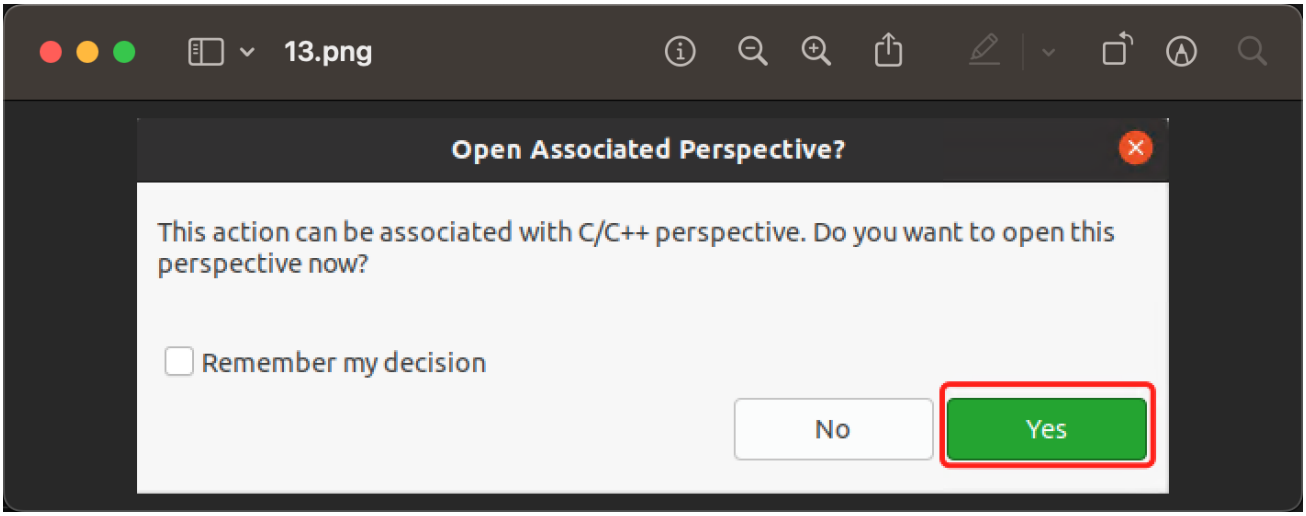
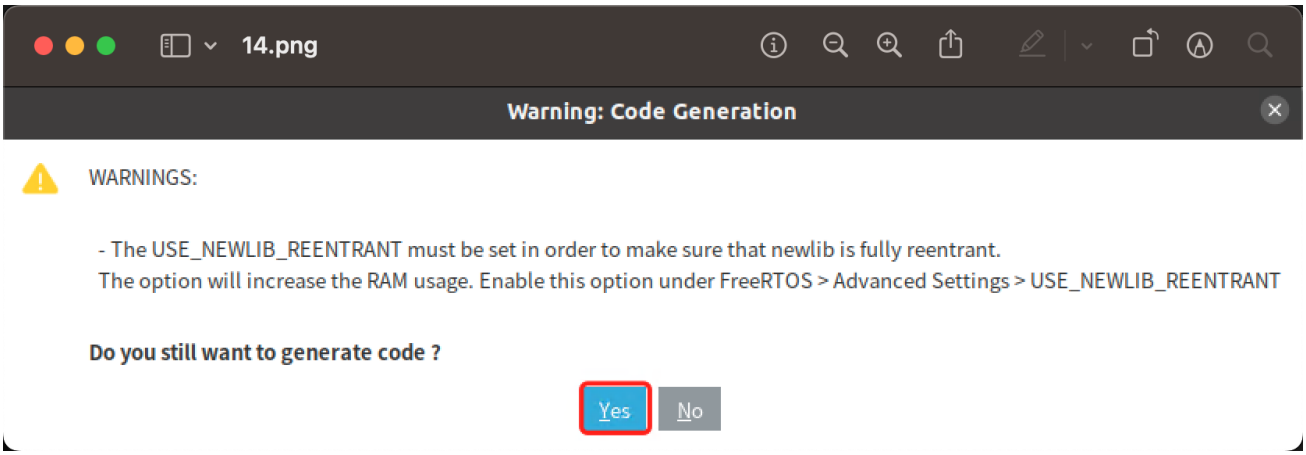


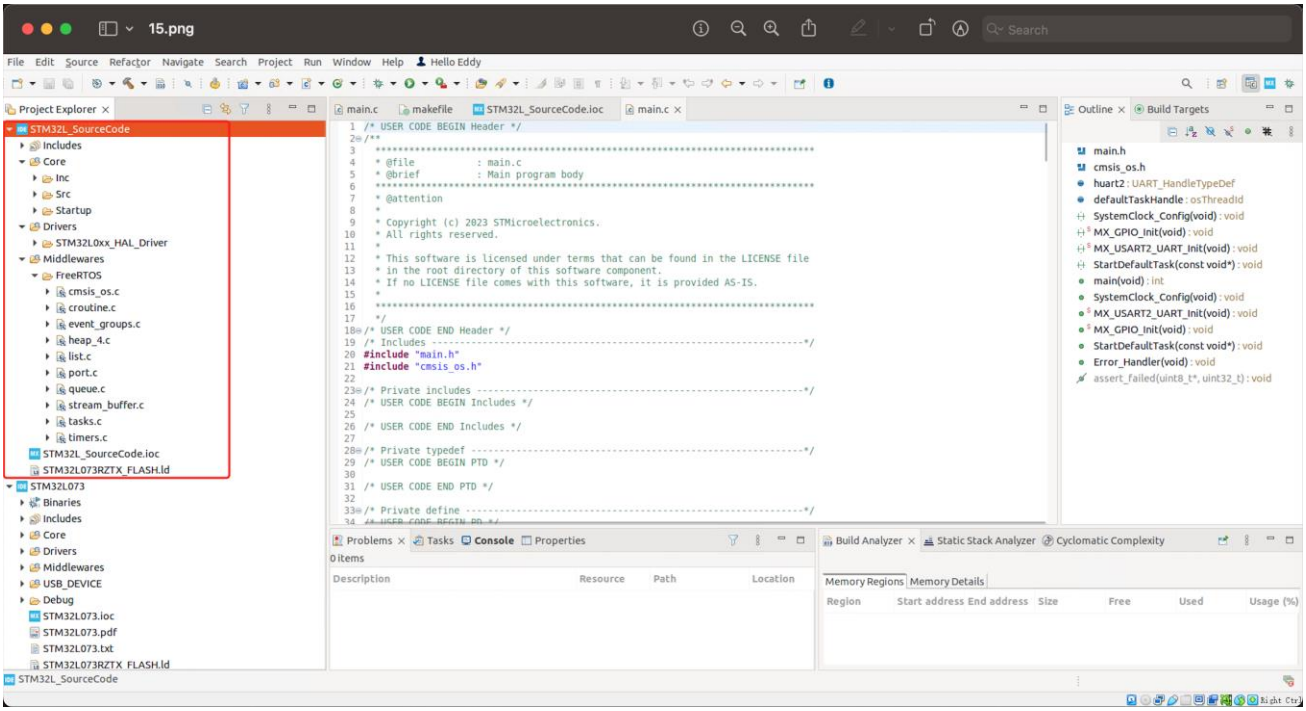
Figure 11 - Generate Code

5.2.10. Confirm the system's reminding





Now the generated source code can be observed.



5.2.11. Build the project to generate make files

However the make file of IDE can only be generated in building. In order to generate the makefile, the project is to be built in IDE.

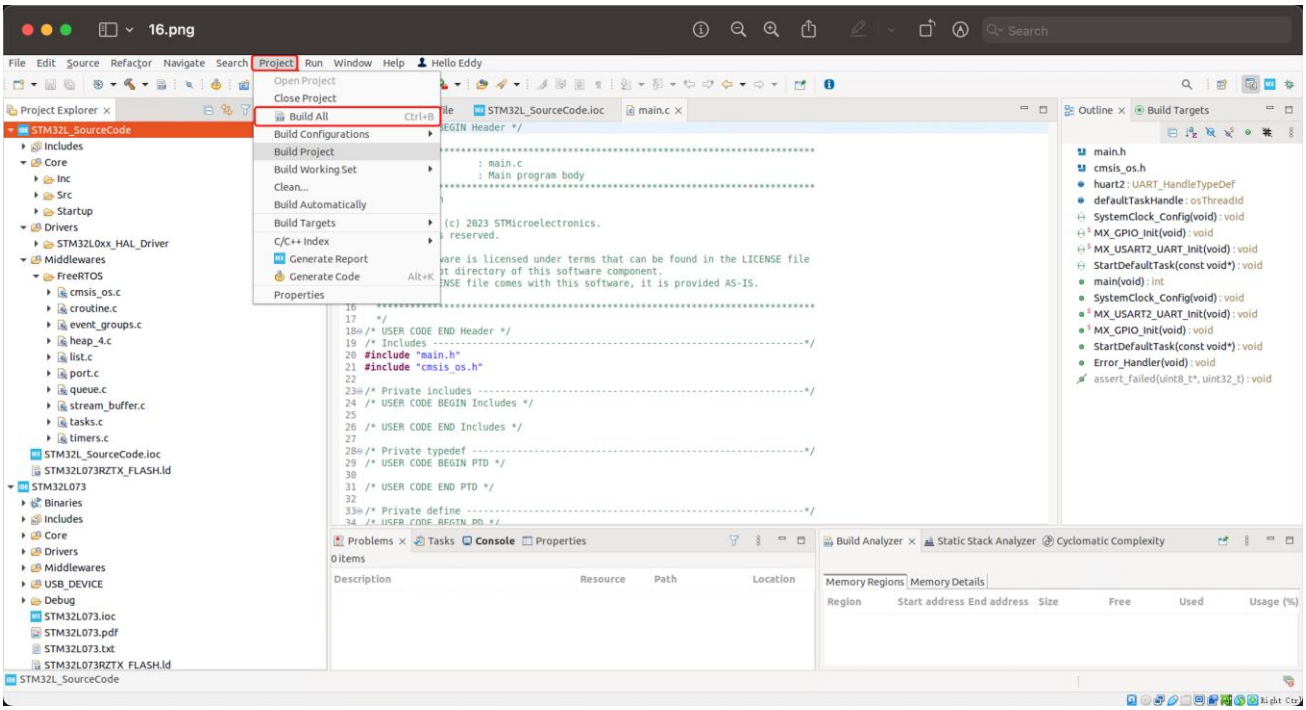


Figure 12 - Build the Project in IDE

5.2.12. Building is completed – now the binary can be observed.

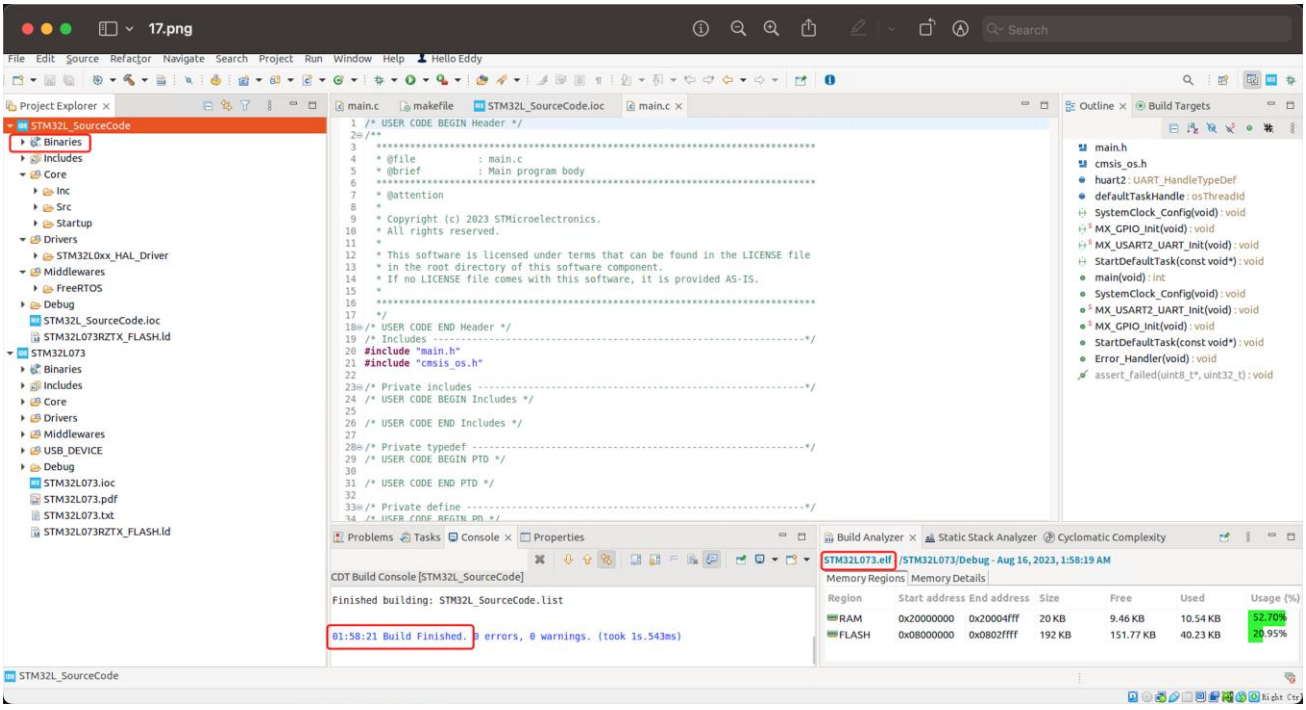


Figure 13 - Build succeeds

Now make files are generated.

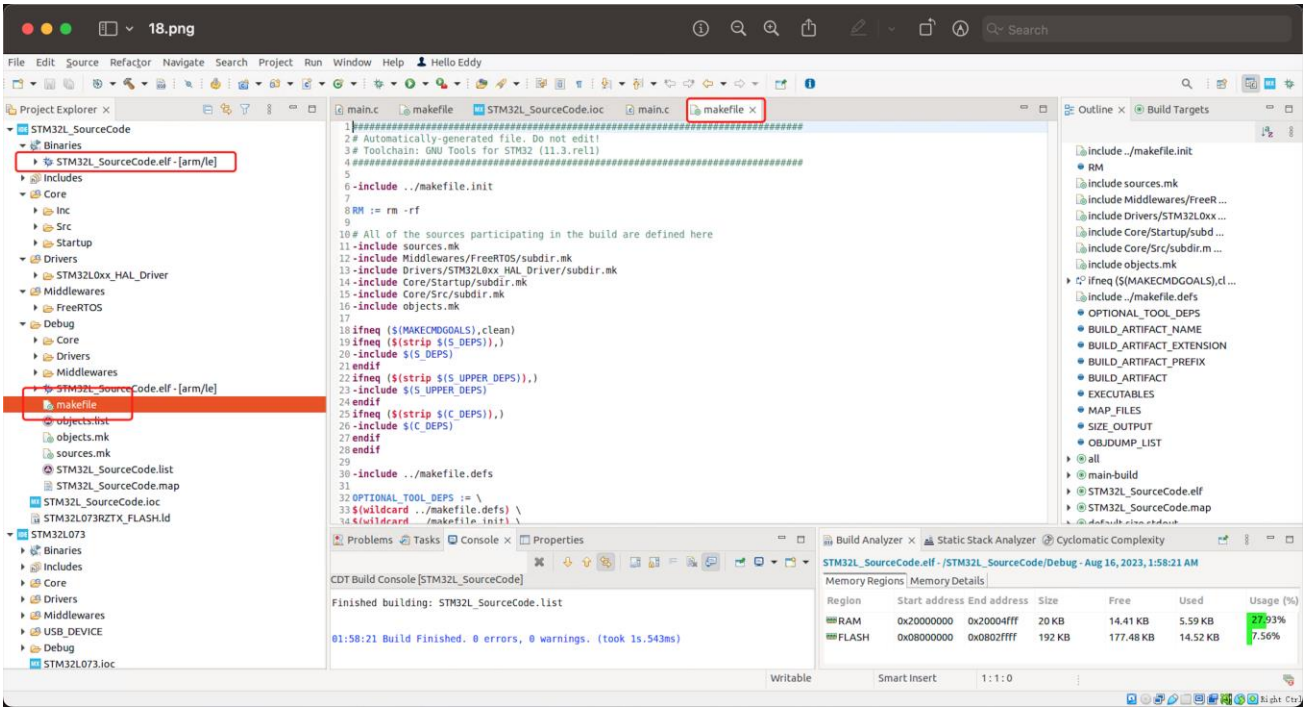


Figure 14 - Make files available

5.3. Cross Build Toolchain

5.3.1. Introduction of GNU toolchains

Gcc-arm-none-eabi, gcc-arm-none-gdb etc. are basically the compiling, linking, debugging.
The abbreviation of toolchain looks like following,
Arch [-vendor] [-os] [-(gnu)eabi]

According to this naming rule, gcc-arm-none-eabi, means it is a GNU C compiler for ARM Coretex microcontroller, with no Operation System(e.g. Linux) supported. EABI means “embedded application binary interface”.

GNU ARM toolchain can be downloaded in following hyperlink,
https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads

The Linux Debian binary are uploaded into project Git server,
https://git-master.quectel.com/mcu-lpwa-dev-project/quectel_bg95_reference_design.git

STM32 source is normally built in CubeIDE. The benefit is obvious – IDE enables graphically configurating the peripherals and features. The HAL/software packs are generated according to this configurations so

IDE building can save efforts.

However, when a hardware relevant source code, depending on application and hardware design, were fixed, the application development would be difficult, in particular when continuous integration is expected. So an off-IDE build system is necessary.

Normally GNU ARM toolchain can be adopted – it is what ST CubeIDE takes by default. Linux or MacOS support a flexible shell/environmental variables configuration – highly recommended.

5.3.2. Install from the toolchain package(Linux as an example)

```
eddy@ubuntu-20:~/Workspace/arm-gnu-toolchain$:tar xJvf arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi.tar.xz
eddy@ubuntu-20:~/Workspace/arm-gnu-toolchain/arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi/bin$ ls
arm-none-eabi-addr2line     arm-none-eabi-gcc-nm            arm-none-eabi-lto-dump
arm-none-eabi-ar            arm-none-eabi-gcc-ranlib        arm-none-eabi-nm
arm-none-eabi-as            arm-none-eabi-gcov              arm-none-eabi-objcopy
arm-none-eabi-c++           arm-none-eabi-gcov-dump         arm-none-eabi-objdump
arm-none-eabi-c++filt       arm-none-eabi-gcov-tool         arm-none-eabi-ranlib
arm-none-eabi-cpp           arm-none-eabi-gdb               arm-none-eabi-readelf
arm-none-eabi-elfedit       arm-none-eabi-gdb-add-index     arm-none-eabi-size
arm-none-eabi-g++           arm-none-eabi-gfortran          arm-none-eabi-strings
arm-none-eabi-gcc           arm-none-eabi-gprof             arm-none-eabi-strip
arm-none-eabi-gcc-12.3.1    arm-none-eabi-ld
arm-none-eabi-gcc-ar        arm-none-eabi-ld.bfd
eddy@ubuntu-20:~/Workspace/arm-gnu-toolchain$ cd -
/home/eddy/Workspace/arm-gnu-toolchain/arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi/bin
eddy@ubuntu-20:~/Workspace/arm-gnu-toolchain/arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi/bin$ arm-none-eabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-none-eabi-gcc
COLLECT_LTO_WRAPPER=/home/eddy/Workspace/arm-gnu-toolchain/arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi/bin/../libexec/gcc/arm-none-eabi/12.3.1/lto-wrapper
Target: arm-none-eabi
Configured with: /data/jenkins/workspace/GNU-toolchain/arm-12/src/gcc/configure --target=arm-none-eabi --prefix=/data/jenkins/workspace/GNU-toolchain/arm-12/build-arm-none-eabi/install --with-gmp=/data/jenkins/workspace/GNU-toolchain/arm-12/build-arm-none-eabi/host-tools --with-mpfr=/data/jenkins/workspace/GNU-toolchain/arm-12/build-arm-none-eabi/host-tools --with-mpc=/data/jenkins/workspace/GNU-toolchain/arm-12/build-arm-none-eabi/host-tools --with-isl=/data/jenkins/workspace/GNU-toolchain/arm-12/build-arm-none-eabi/host-tools --disable-shared --disable-nls --disable-threads --disable-tls --enable-checking=release --enable-languages=c,c++,fortran
```

```
--with-newlib --with-gnu-as --with-gnu-ld --with-sysroot=/data/jenkins/workspace/GNU-toolchain/arm-12/build-arm-none-eabi/install/arm-none-eabi --with-multilib-list=aprofile,rmprofile --with-pkgversion='Arm GNU Toolchain 12.3.Rel1 (Build arm-12.35)' --with-bugurl=https://bugs.linaro.org/
Thread model: single
Supported LTO compression algorithms: zlib
gcc version 12.3.1 20230626 (Arm GNU Toolchain 12.3.Rel1 (Build arm-12.35))
```

5.3.3. Set PATH environment

```
Add toolchain executive path into $PATH,e.g. edit “~/.bashrc”
export PATH=/home/eddy/Workspace/arm-gnu-toolchain/arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi/bin:/home/eddy/Workspace/openocd-0.12.0-rc1/openocd-0.12.0-rc1/build/bin:$PATH
eddy@ubuntu-20:~$ source ~/.bashrc
```

5.3.4. Build project

Change working space to the STM32 source folder, e.g. “/home/eddy/Workspace/Quectel_BG95_Reference_Design/quectel_bg95_reference_design/source/STM32L0/Debug”.

```
eddy@ubuntu-20:~/Workspace/Quectel_BG95_Reference_Design/quectel_bg95_reference_design/source/STM32L0/Debug$ pwd
/home/eddy/Workspace/Quectel_BG95_Reference_Design/quectel_bg95_reference_design/source/STM32L0/Debug
eddy@ubuntu-20:~/Workspace/Quectel_BG95_Reference_Design/quectel_bg95_reference_design/source/STM32L0/Debug$ ls
Core makefile objects.list sources.mk
Drivers Middlewares objects.mk USB_DEVICE
eddy@ubuntu-20:~/Workspace/Quectel_BG95_Reference_Design/quectel_bg95_reference_design/source/STM32L0/Debug$ make all
```

5.3.5. Multiple Host Operation Systems Support

The easiest way is to build/flash via STCubeIDE. The STCubeIDE has difference deliveries on MacOS, Linux and Windows. The users of Quectel reference SDK, can import the project out of it.

However, STCubeIDE does not support MacOS (Apple Silicon CPU) for instance; some users may prefer the command-line development environment, how to deal with it?
This chapter will talk about the cross building environment on other operation systems. As the compiler and linker both come from GNU ARM toolchain. The only dependency are located in the tools path in system vaviables and makefile. Although ST does not recommend to edit the automatically generated makefile, but it is possible to customize it to remove the dependency.

Following are what needs to be done on MacOS (apple silicon M1 based). The other operation systems work the similar way.

First, the proper (Apple Silicon Version) GNU ARM GCC tool has to be retrieved and put in the $(PATH) folders.
https://developer.arm.com/-/media/Files/downloads/gnu/12.3.rel1/binrel/arm-gnu-toolchain-12.3.rel1-darwin-arm64-arm-none-eabi.tar.xz?rev=c22a1b092d0d401291232d21e24cf986&hash=CBB534DAF3233E46A9C0BAD1A3D68740

Second, following two files should be customized according to your environment.

```
maoxinhua@Mac-mini Debug % pwd
/Users/maoxinhua/Workspace/mirror/misc/Workspace/fae_git/quectel_bg95_reference_design/source/STM32F401RET6/Debug
maoxinhua@Mac-mini Debug % ls
Core            Middlewares     build.sh        objects.list    sources.mk
Drivers         Quectel         makefile        objects.mk
```

```
maoxinhua@Mac-mini Debug % cat build.sh
#!/bin/sh

#define your flash load file path here!
export
STM32F_FLASH_LD=/Users/maoxinhua/Workspace/mirror/misc/Workspace/fae_git/quectel_bg95_reference_design/source/STM32F401RET6/STM32F401RETX_FLASH.ld

#build command
make clean
make -j8 all
```

```
ifndef $(STM32F_FLASH_LD)
#           STM32F_FLASH_LD := \

/mnt/f/Workspace/fae_git/quectel_bg95_reference_design/source/STM32F401RET6/STM32F401RETX_F\
LASH.ld \
$(warning STM32F_FLASH_ID := $(STM32F_FLASH_ID))
endif
```

```
# All Target
all: main-build

# Main-build Target
main-build: STM32F401RET6U_CubleIDE_FreeRTOS.elf secondary-outputs

# Tool invocations
STM32F401RET6U_CubleIDE_FreeRTOS.elf STM32F401RET6U_CubleIDE_FreeRTOS.map: $(OBJS)
$(USER_OBJS) $(S\
TM32F_FLASH_LD) makefile objects.list $(OPTIONAL_TOOL_DEPS)
	arm-none-eabi-gcc -o "STM32F401RET6U_CubleIDE_FreeRTOS.elf" @"objects.list"
$(USER_OBJS) $(\
LIBS) -mcpu=cortex-m4 -T$(STM32F_FLASH_LD) --specs=nosys.specs -Wl,-
Map="STM32F401RET6U_CubleIDE_Fr\
eeRTOS.map" -Wl,--gc-sections -static --specs=nano.specs -mfpu=fpv4-sp-d16 -mfloat-abi=hard -mthumb\
  -Wl,--start-group -lc -lm -Wl,--end-group
	@echo 'Finished building target: $@'
	@echo ' '
```

5.4. Debugging and Flashing

OpenOCD
https://openocd.org/doc/doxygen/html/index.html

6 Appendix A Reference

Text text text text text
Text text text text text

7 Appendix B Reference Documents

Ref No.	Document
[1]	Datasheet STM32L073RZT6
[2]	User Manual STM32CubeMX for STM32 configuration
[3]	User Manual STM32CubeIDE user guide