AN0033
Application Note

Eclipse with GCC

Introduction

As a popular integrated development environment (IDE), Eclipse supports various computer languages including C/C++ through a wide range of plug-ins, adding to the flexibility of Eclipse platform, on which the user is able to conduct software IDE

This Application Note is written to describe how to use Eclipse plug-ins to debug AT32 series devices.

Applicable products:

Part number	AT32F403xx
	AT32F413xx
	AT32F415xx
	AT32F403A/AT32F407
	AT32F421

Content

List of tables

List of figures

1 Overview

This Application Note gives a detailed description of how to debug AT32 series chips using Eclipse, ARM-GCC compiler, GNU-ARM plug-ins, JLink and ATLink.

It mainly covers the following contents:

- Eclipse debug environment installation
- Create Eclipse project
- Eclipse compile configuration
- Eclipse+JLink debug configuration
- Eclipse+ATLink debug configuration
- Example routine

Note: This installation manual is based on WINDOWS 7 x64 system to use AT32F403ZGT6, AT32F413RCT7 and AT32F415RCT7 for debugging.

All the software kits in this document can be found in *AT32_Eclipse_Packet.zip*, which is unzipped to install and run.

AT32_Eclipse_Packet.zip contains the following files:

Figure 1. Under AT32_Eclipse_Packet.zip

The script files and example routine are packaged in *AT32_Eclipse_Source.zip*, in which the software can be used for installation and configuration.

AT32_Eclipse_Source.zip contains the following files:

Figure 2. Under AT32_Eclipse_Source.zip

GCC: gcc startup.S files

Ldscripts: script files

SVD: AT32 svd files

Example: This folder contains AT32F403, AT32F413 and AT32F415 example codes that can be opened through Eclipse for direct use. Please refer to Section 7 for more information on Example.

Figure 3. AT32 Examples

2 Eclipse debug environment preparation

First of all, the following software needs to be installed:

- Eclipse IDE for C/C++ Developers
- GNU ARM Eclipse plug-ins
- GCC ARM compiler
- GNU ARM Eclipse Build Tools (make, rm and others）

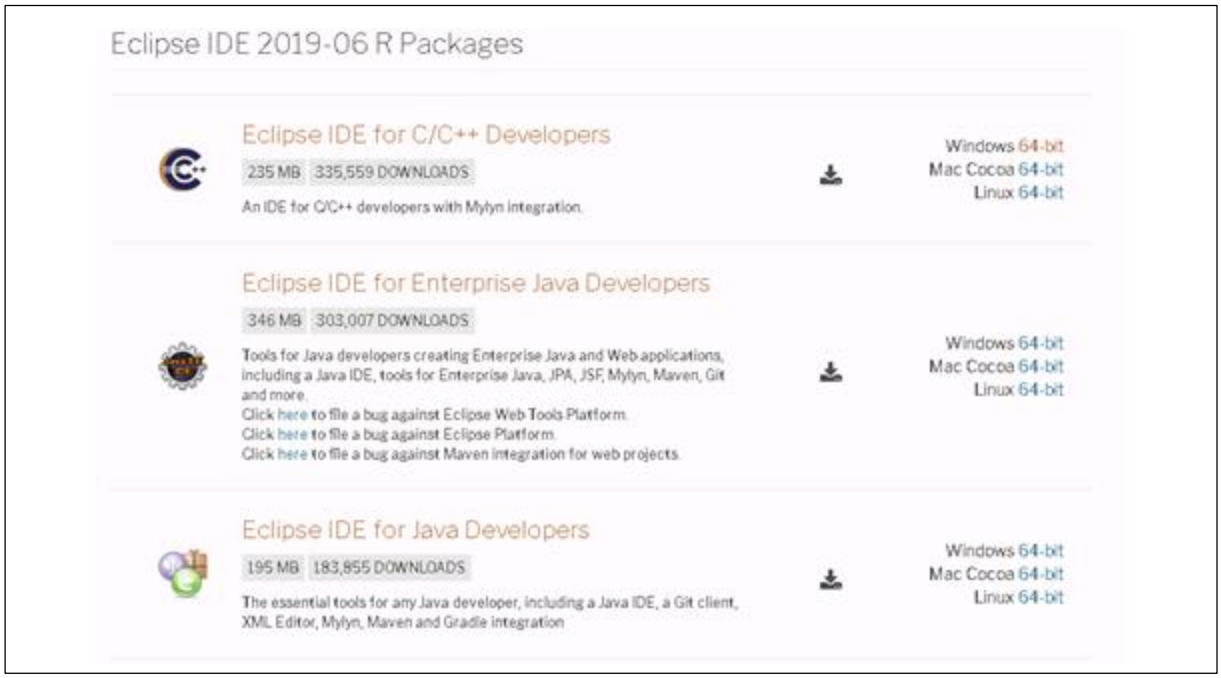
The subsequent sections will show how to install these software in detail.

2.1 Eclipse IDE for C/C++ Developers

We only need to download the latest version of C/C++ from Eclipse. There is one version available in *AT32_Eclipse_Packet.zip*, that is, *eclipse-cpp-2019-06-R-win32-x86_64.zip.*

Download link：http://www.eclipse.org/downloads/eclipse-packages/

Figure 4. Eclipse download page



Download and unzip the eclipse-cpp-2019-06-R-win32-x86_64.zip

The unzipped package contains the following directories:

Figure 5. Eclipse directory

Under this directory, we click on *eclipse.exe* to run Eclipse, but not available for code debugging at this time without installing plug-ins.

2.2 GNU ARM Eclipse plug-ins installation

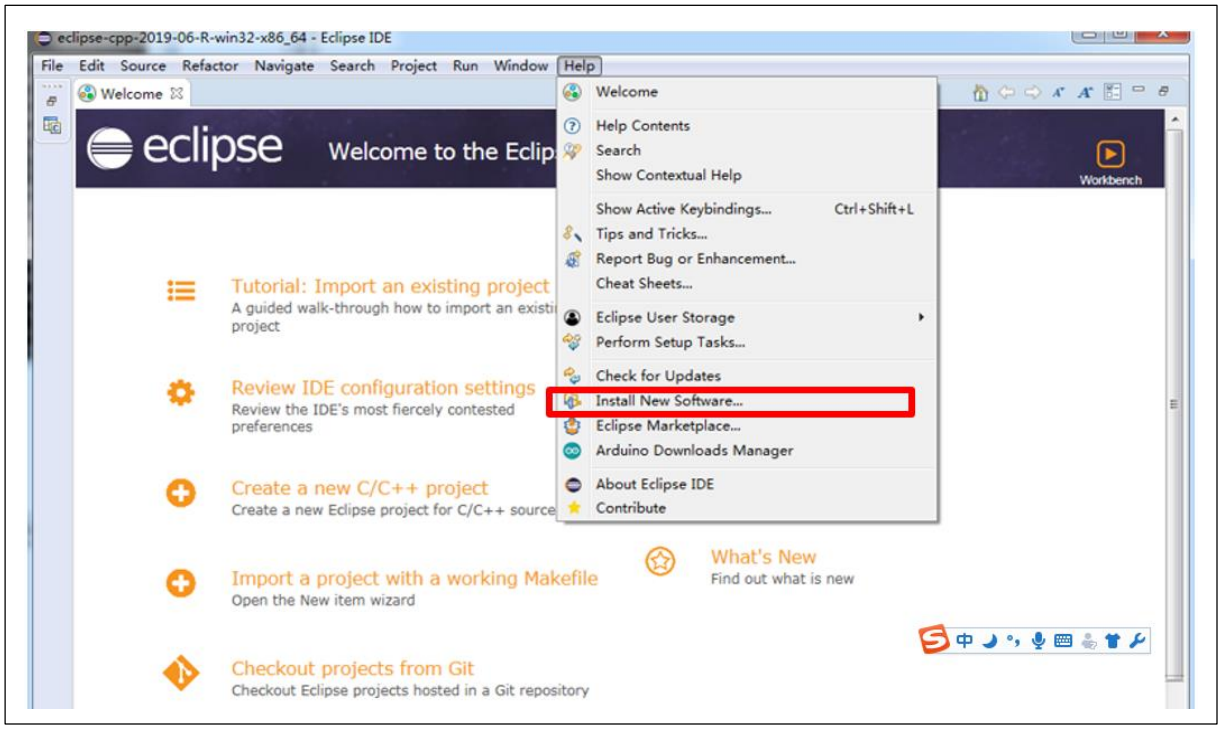
Download and unzip the latest version of GNU ARM Eclipse plug-in: *ilg.gnumcueclipse.repository-4.5.1-201901011632.zip*, which is available in the *AT32_Eclipse_Packet.zip*.

Download link：https://github.com/gnu-mcu-eclipse/eclipse-plugins/releases

To start installation:

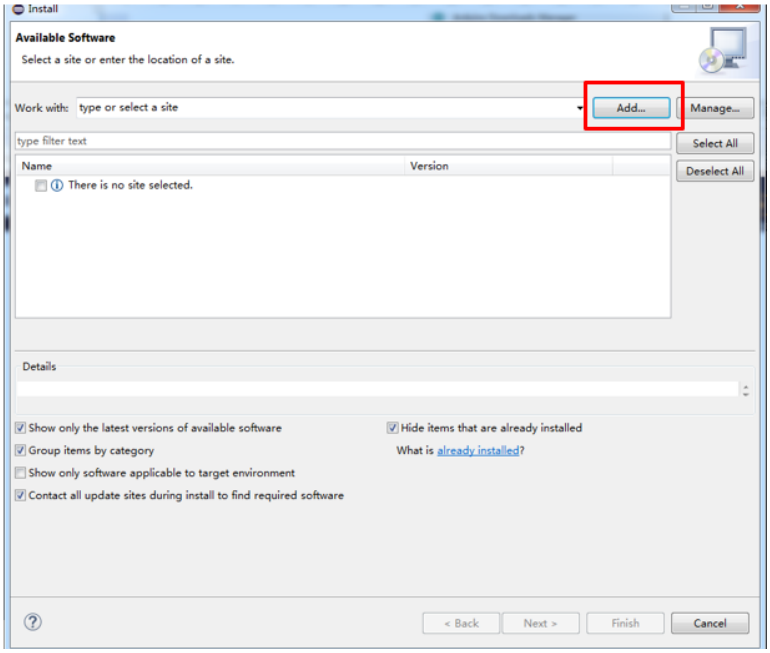
1. Go to ***Eclipse Help->Install New Software.***

Figure 6. Click on Install New Software



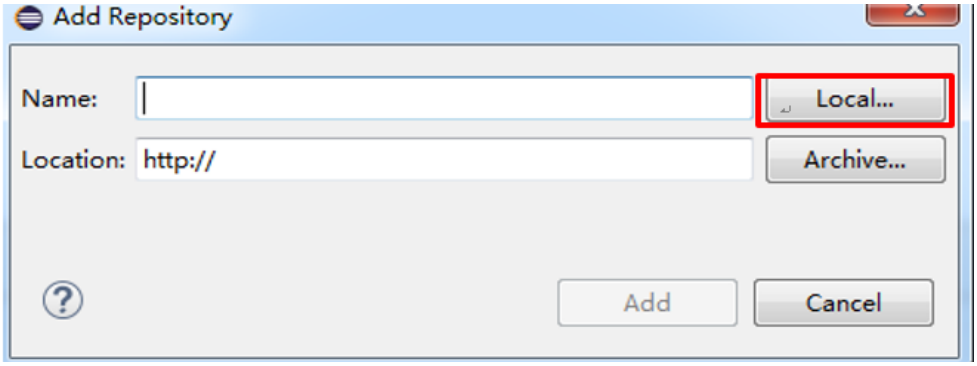
2. Click on ***Add...***

Figure 7. Click on Add



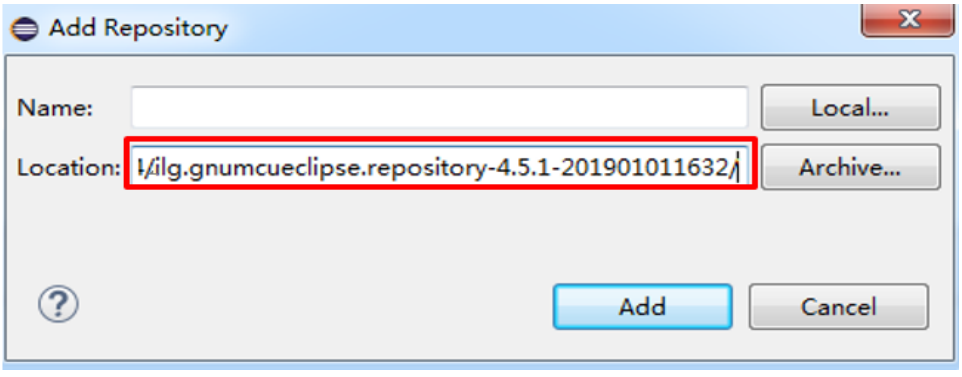
3. Add a local plug-in, or use internet path to automatically download and install.

Figure 8. Add Repository



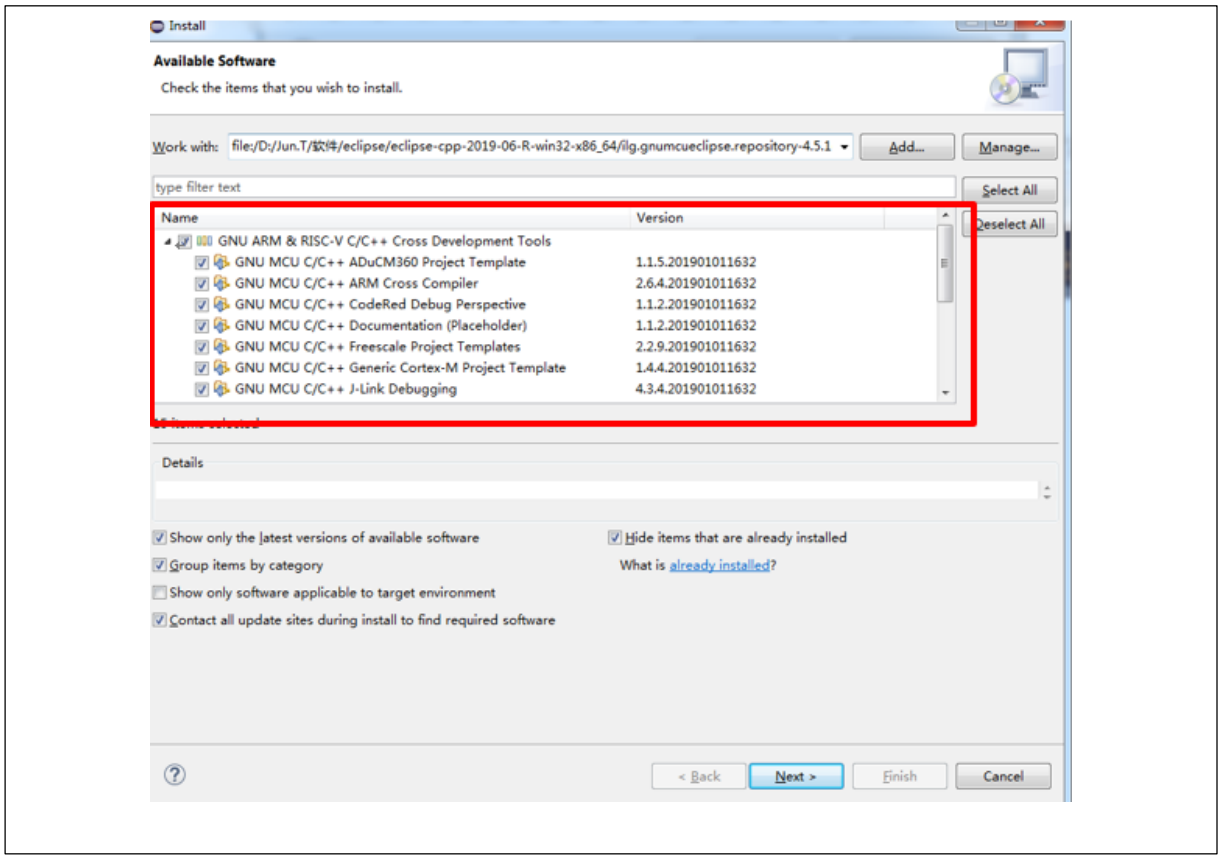
4. Select a local plug-in directory, and click on ***Add.***

Figure 9. Select plug-in directory



5. Tick all plug-ins, and click on ***Next.***

Figure 10. Tick plug-ins



6. Installation is completed, and click on ***Next.***

Figure 11. Installation finished

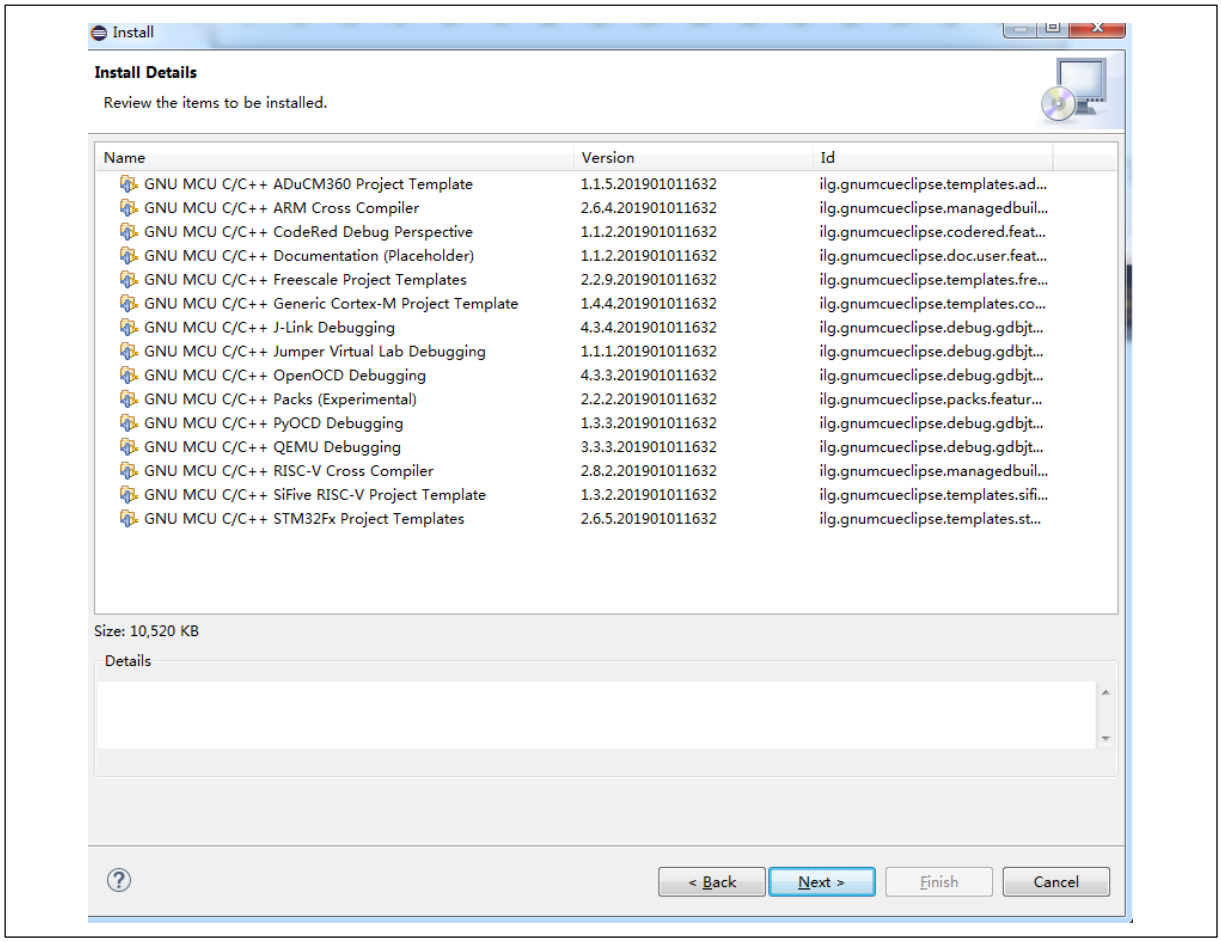
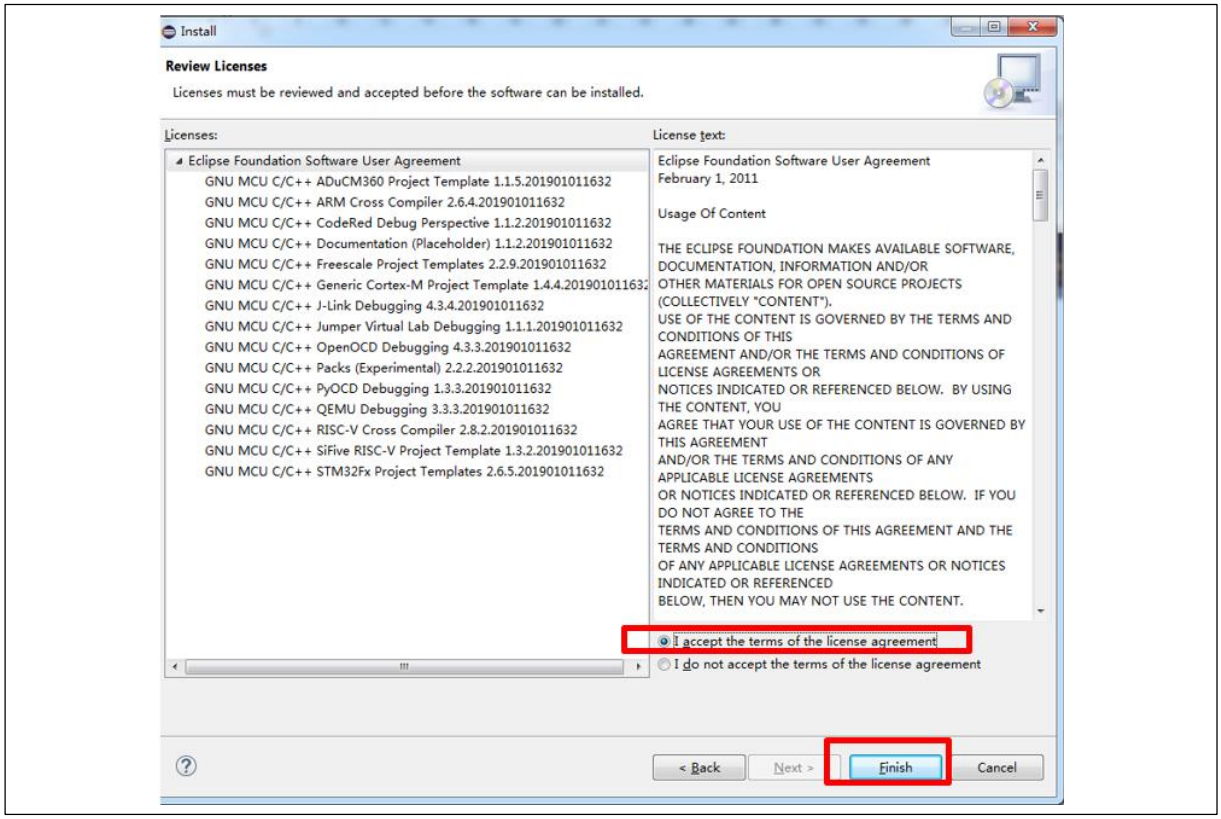
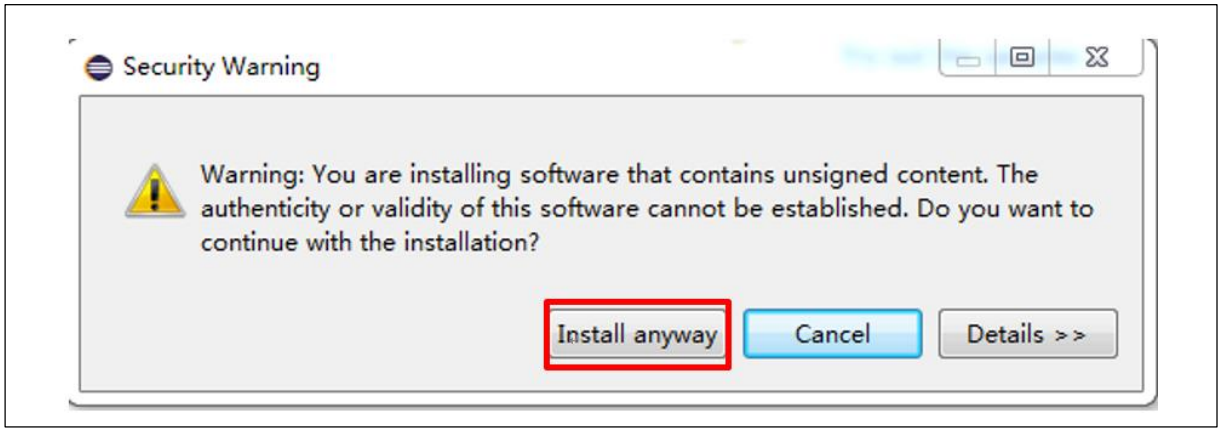


Figure 12. Accept the terms of license agreement



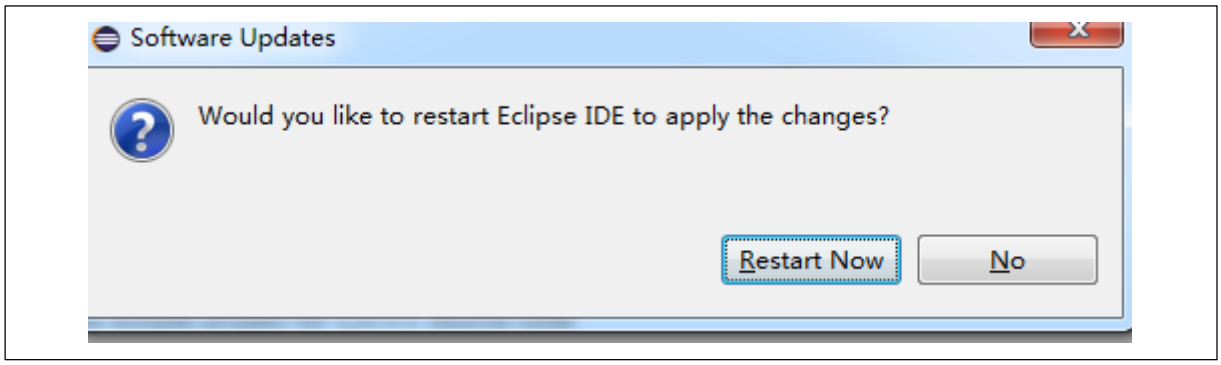
7. Go to ***Install anyway***

Figure 13. Install anyway



8. ***Restart*** Eclipse

Figure 14. Restart Eclipse



2.3 ARM GCC compiler tool chains installation

Download the latest compiler tool chain: *gcc-arm-none-eabi-8-2019-q3-update-win32-sha2.exe*, which is also included in *AT32_Eclipse_Packet.zip*.

Download link: https://launchpad.net/gcc-arm-embedded/+download

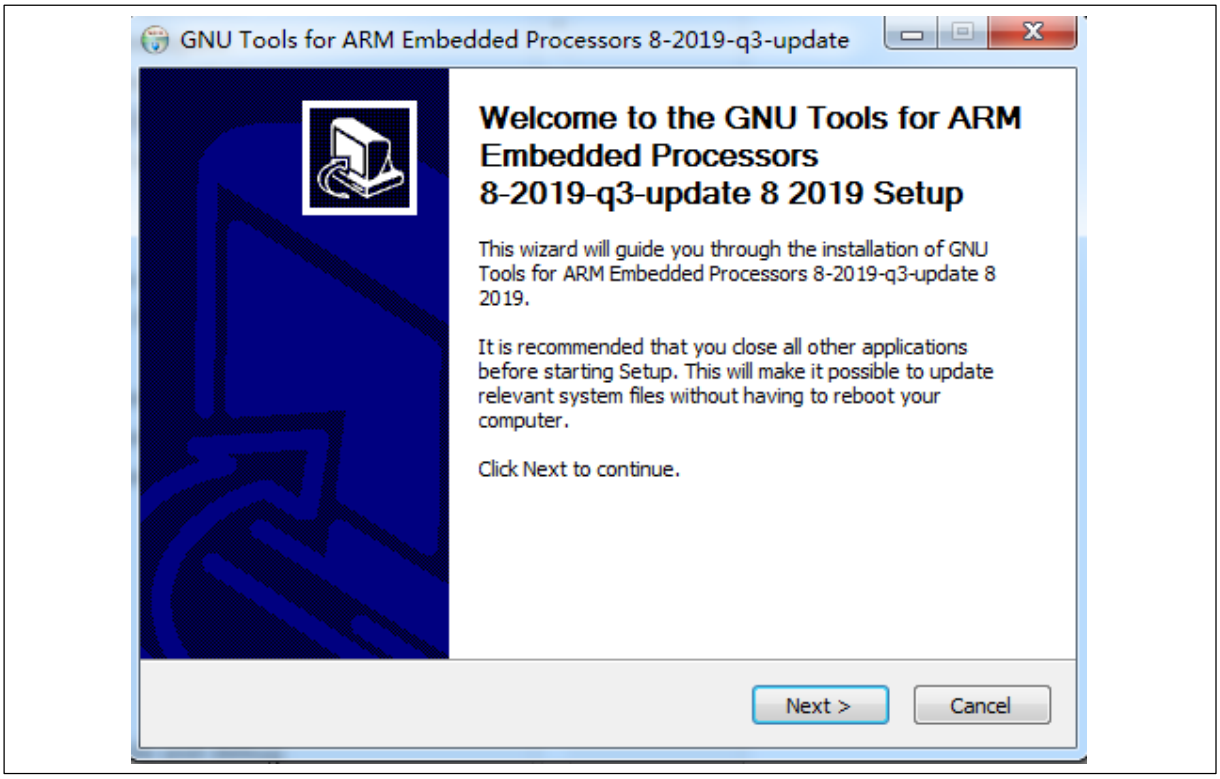
To start installation:

1. Select ***Chinese (Simplified)***

Figure 15. Select Chinese (Simplified)

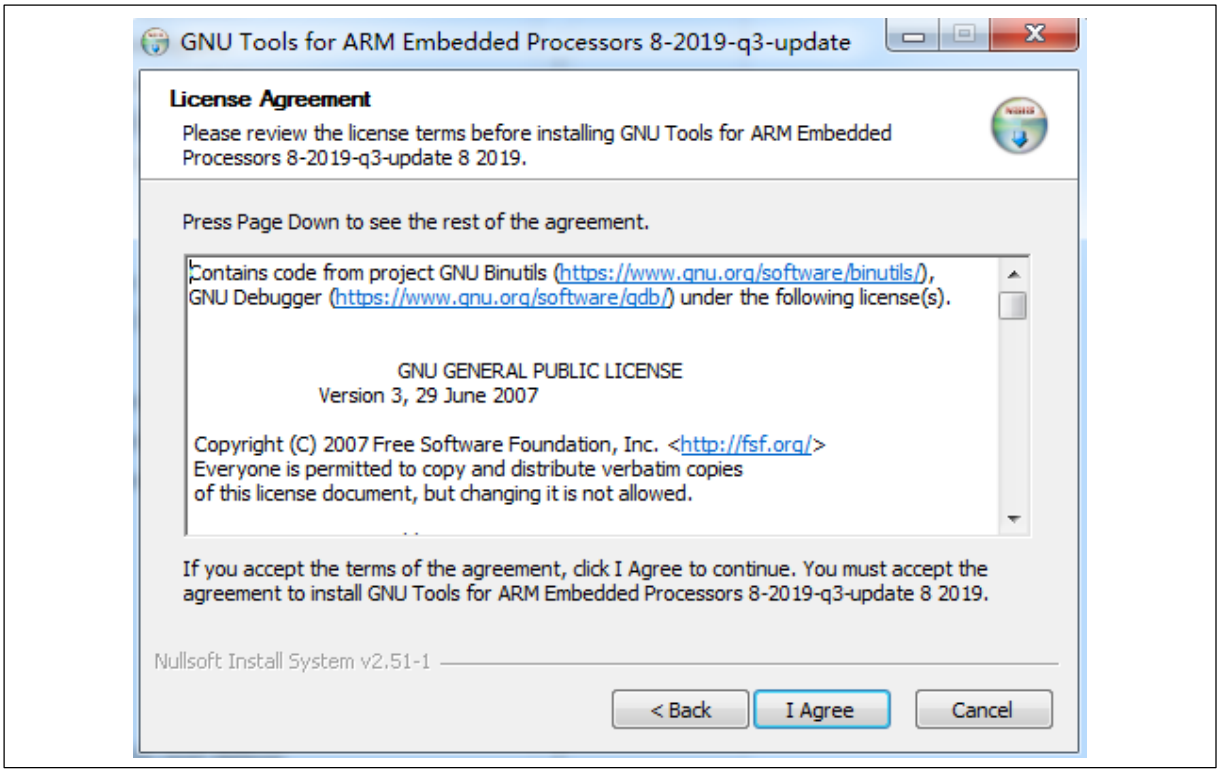
2. Go to ***Setup wizard***, click on ***Next***.

Figure 16. Setup wizard



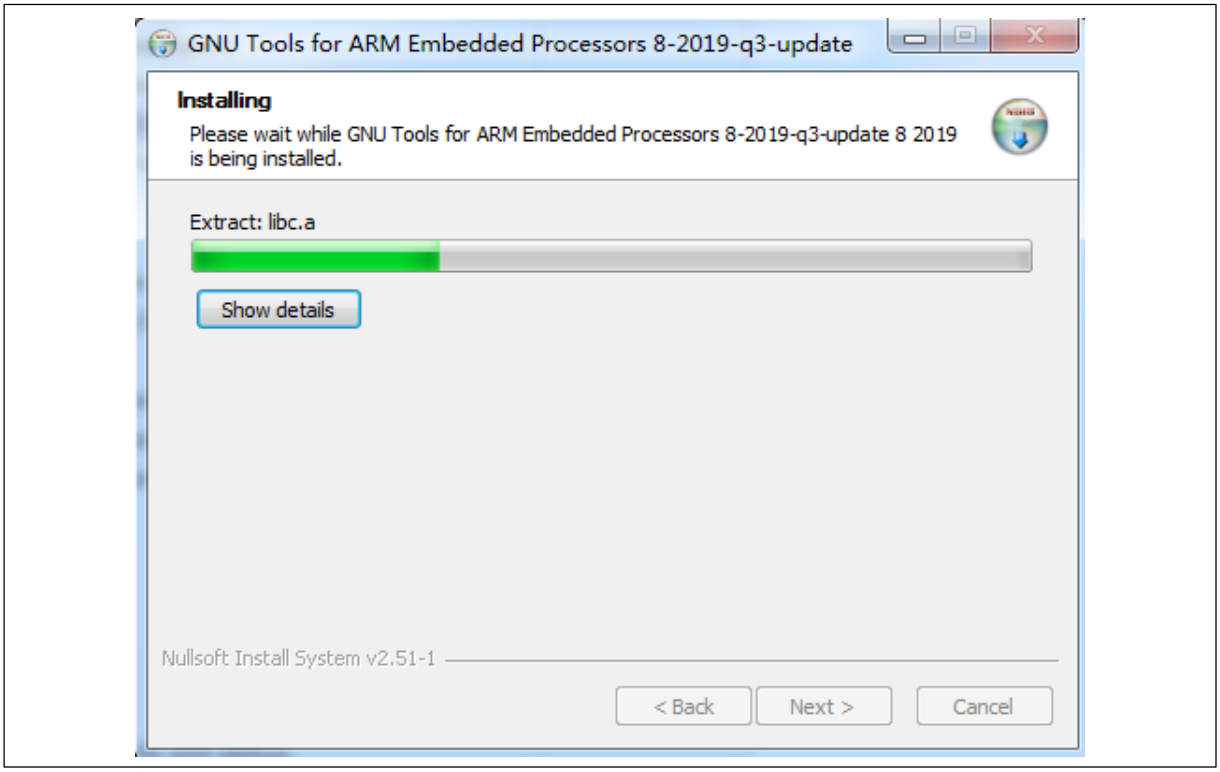
3. Click on ***Accept*** for the license agreement.

Figure 17. Accept license agreement



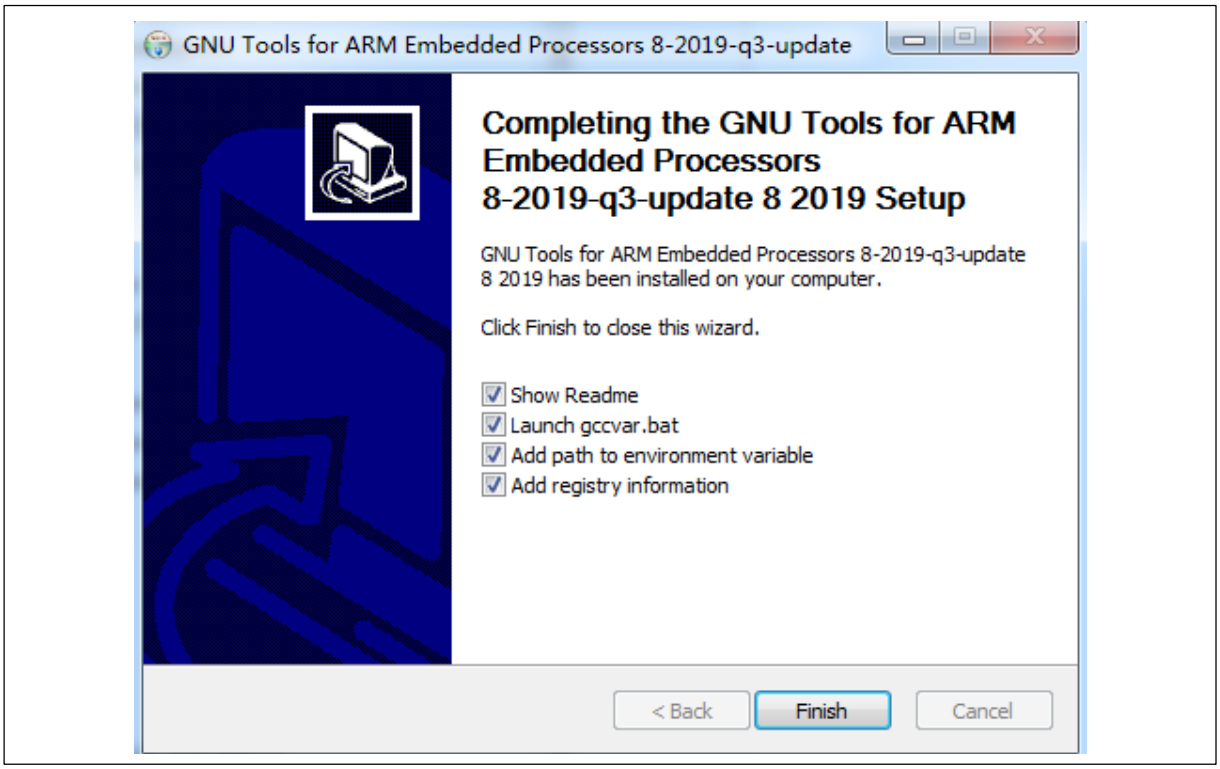
4. Select the default installation location. Click on ***Install***.

Figure 18. Installation progress



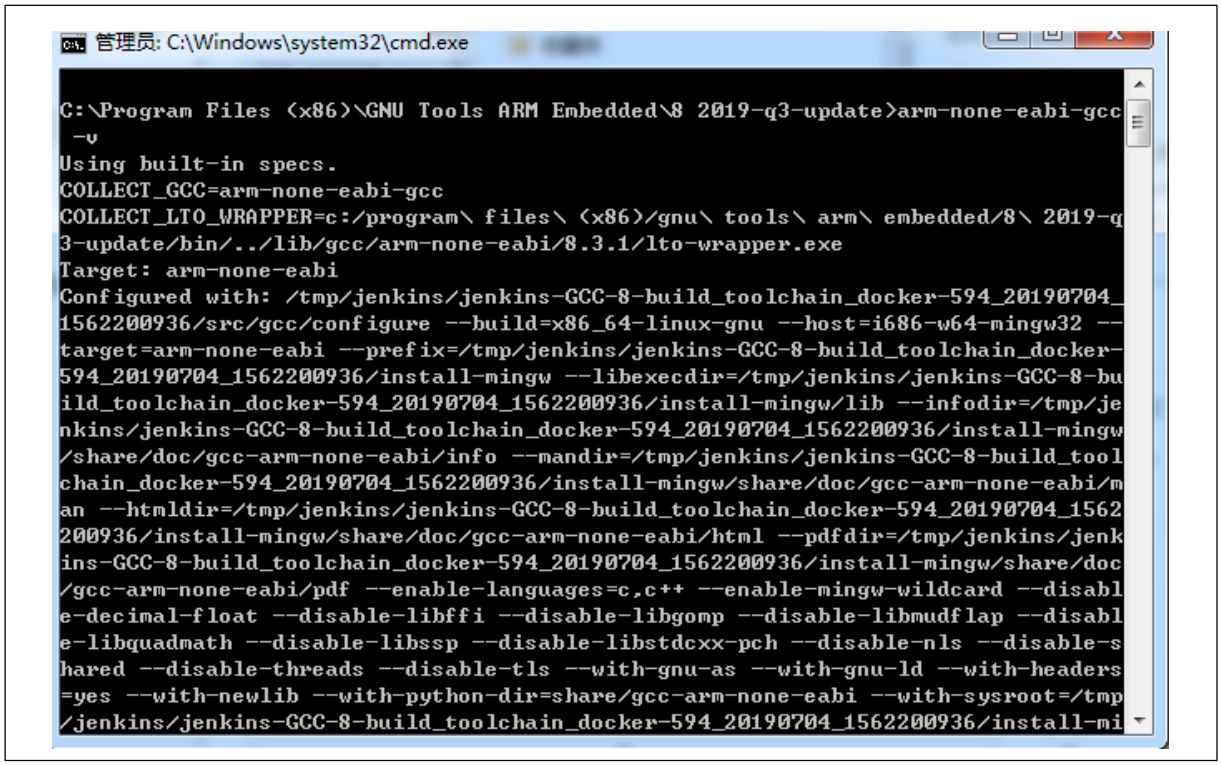
5. In the pop-up installation window, tick ***Add path to environment variable*** for auto add**,** or else, you need to do so by manual operation.

Figure 19. Tick *Add path to environment variable*



6. After finishing installation, enter ***arm-none-eabi-gcc –v*** in the pop-up command window, and some information including version code will be displayed, indicating that it is a successful installation.

Figure 20. Installation result displayed



2.4 GNU ARM Eclipse Build Tools installation

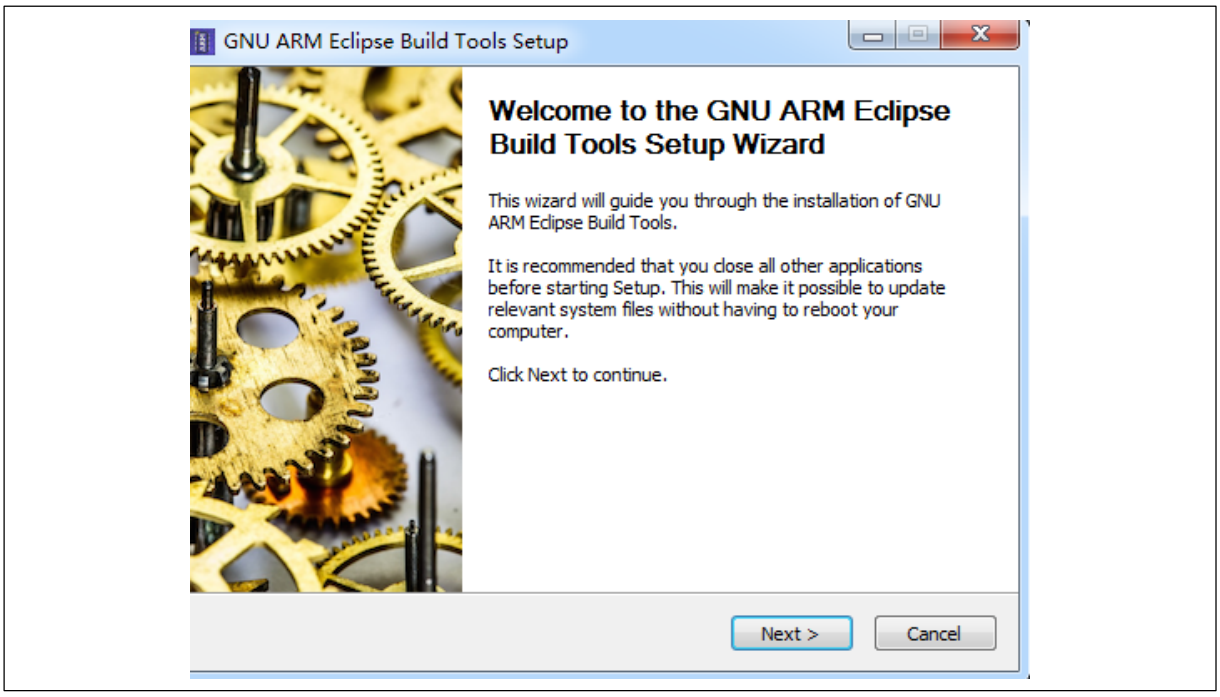
This section provides information about the setup of such commands as make and rm.

Download link：https://sourceforge.net/projects/gnuarmeclipse/files/Build Tools/

AT32_Eclipse_Packet.zip contains a usable version: *gnuarmeclipse-build-tools-win64-2.6-201507152002-setup.exe.*

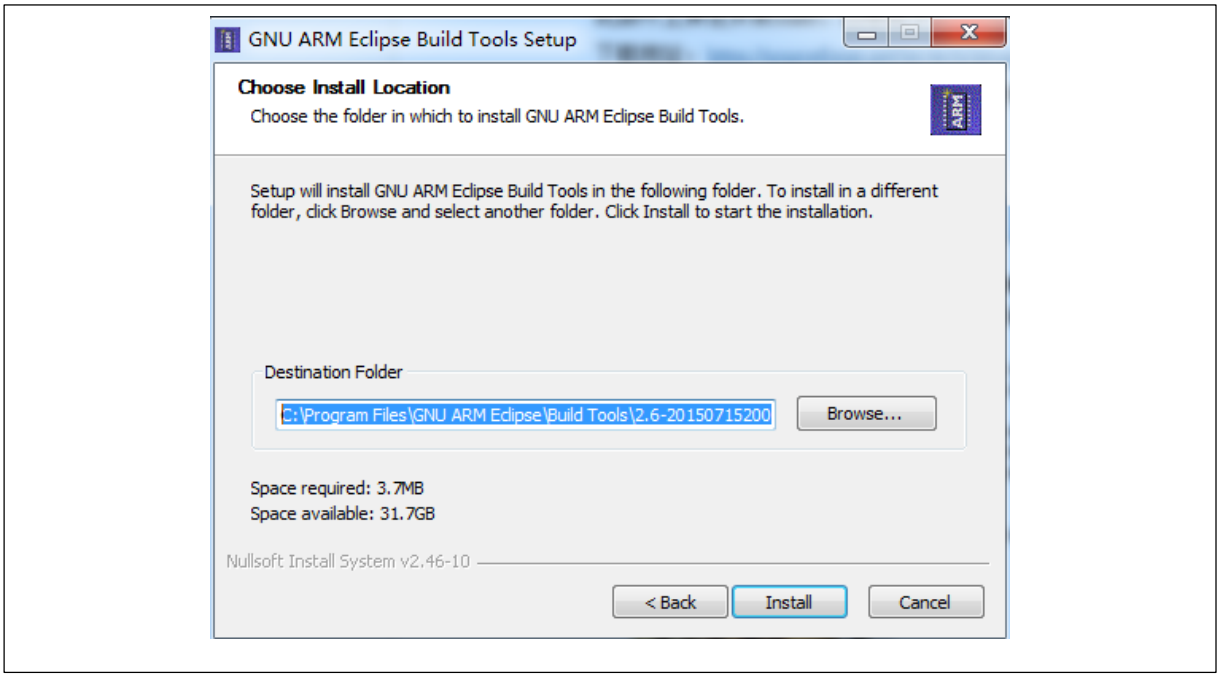
1. Run the installation package.

Figure 21. Run installation package



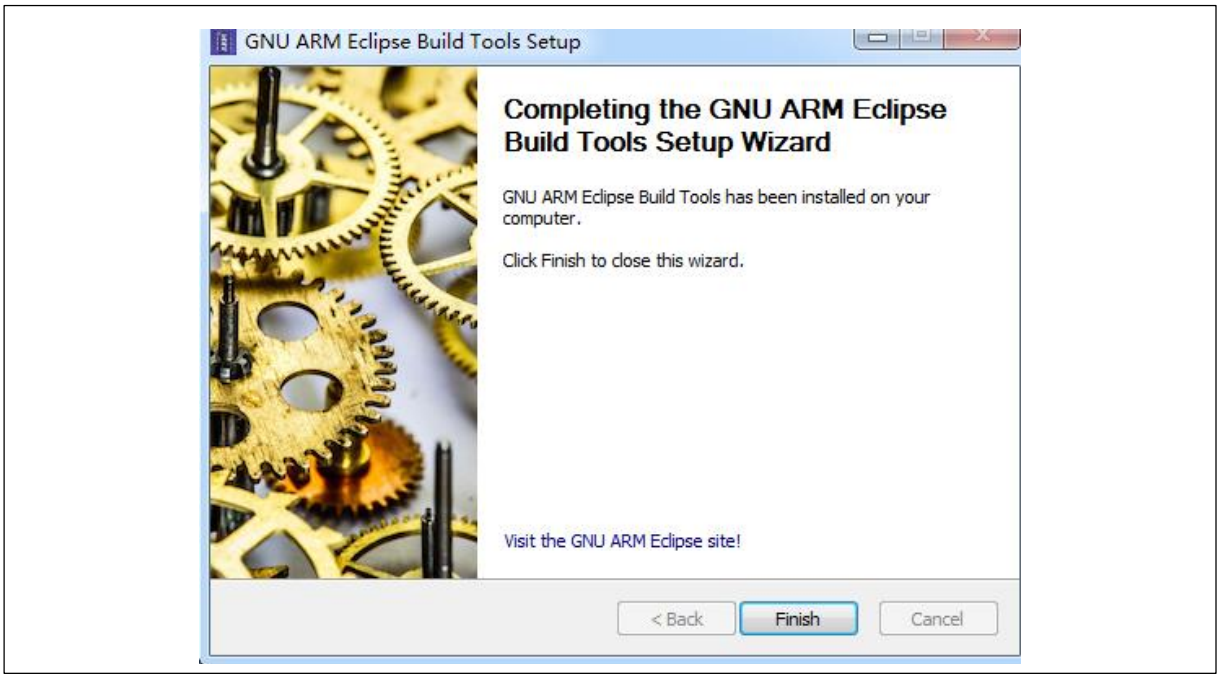
2. Select destination folder.

Figure 22. Select destination folder



3. Restart Eclipse after finishing installation.

Figure 23. Installation completed



2.5 Install JLink

It is necessary to copy AT32 series chips to JLink directory through ICP.

1. JLink installation (omitted)

   Download the latest version of JLink to install.

2. Copy algorithm file

   To recognize and download program to AT32 series through JLink, the AT32 algorithm file should be downloaded to JLink directory through ICP tool.

3 Create Eclipse project

In this section, we will demonstrate how to create an Eclipse project using AT32 BSP library, which mainly covers:

- AT32 BSP structure
- To create a new Eclipse project, proceed as follows
  - Migrate BSP Templates, and copy BSP files to new project
  - Create script files: ldscripts/AT32F4xx_FLASH.ld
  - Use new startup .S files (refer to Sxx)

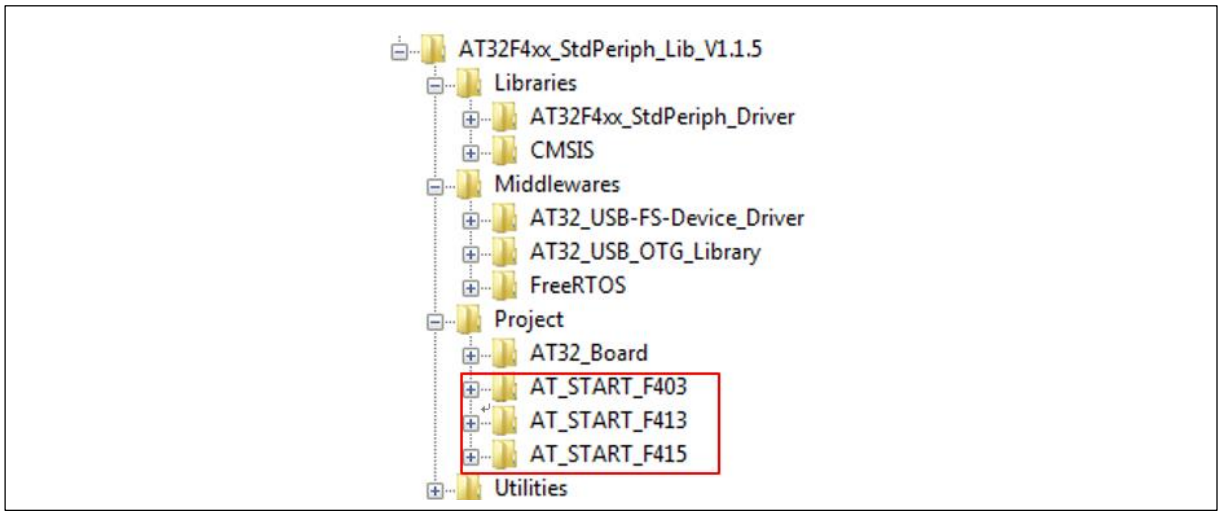
Note: Do not use the Chinese path to create project, otherwise failed to find Eclipse files.

3.1 AT32 BSP library structure

We need to copy the files in BSP to Eclipse directory when using Eclipse to create project. In this example, we use AT32F4xx_StdPeriph_Lib_V1.1.5. For the latest library, please download it from our official website.

AT32F4xx_StdPeriph_Lib_V1.1.5 Library structure:

Figure 24. AT32 library structure



For example, if we want to migrate Templates to Eclipse project, the following files need to be copied:

AT32 library file: AT32F4xx_StdPeriph_Lib_V1.1.5Libraries/AT32F4xx_StdPeriph_Driver

CMSIS: AT32F4xx_StdPeriph_Lib_V1.1.5/Libraries/ CMSIS/ CM4

Note: There is no need to copy .s files in CM4 because they would be modified.

AT32F4xx_StdPeriph_Lib_V1.1.5/Project/AT32_Board

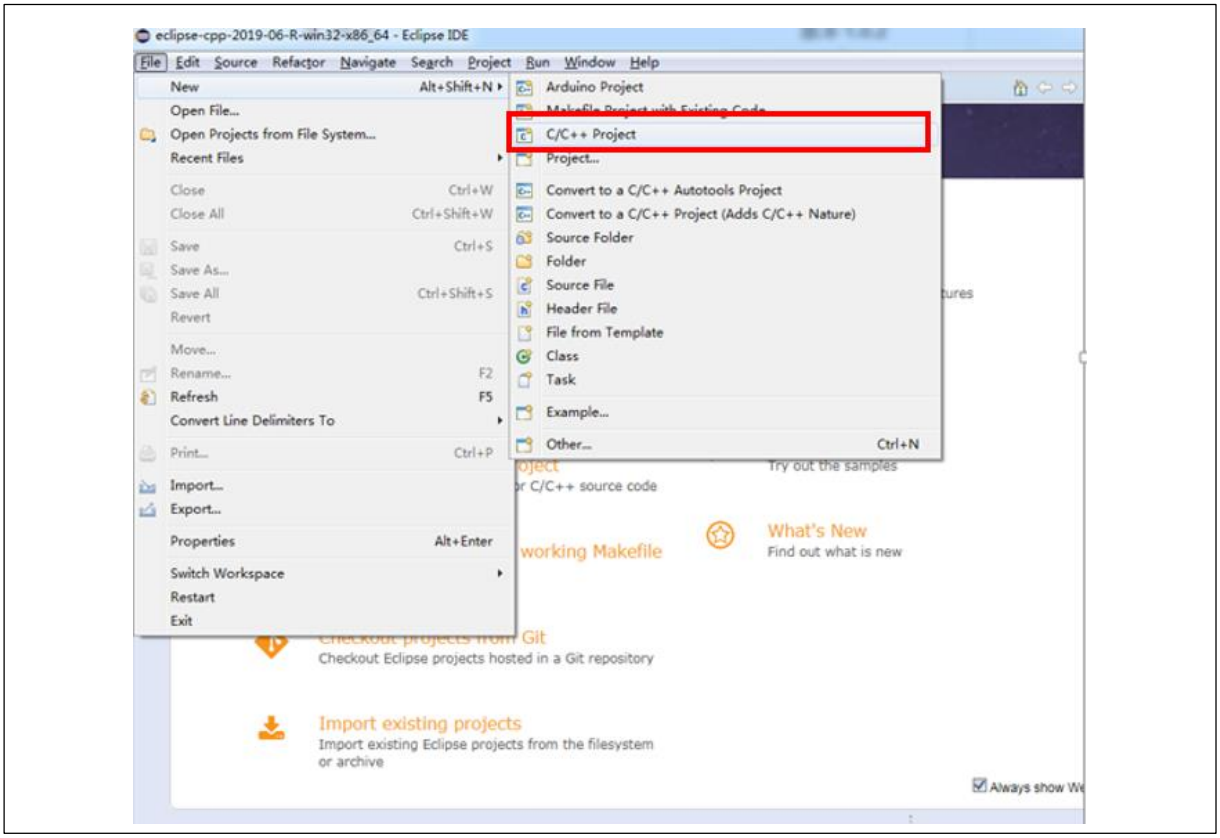
AT32F4xx_StdPeriph_Lib_V1.1.5/Project/AT_START_F4xx/Templates/at32f4xx_it.c at32f4xx_it.h main.c readme.txt

3.2 Create new project using Eclipse

This section demonstrates how to migrate Templates in BSP to Eclipse.

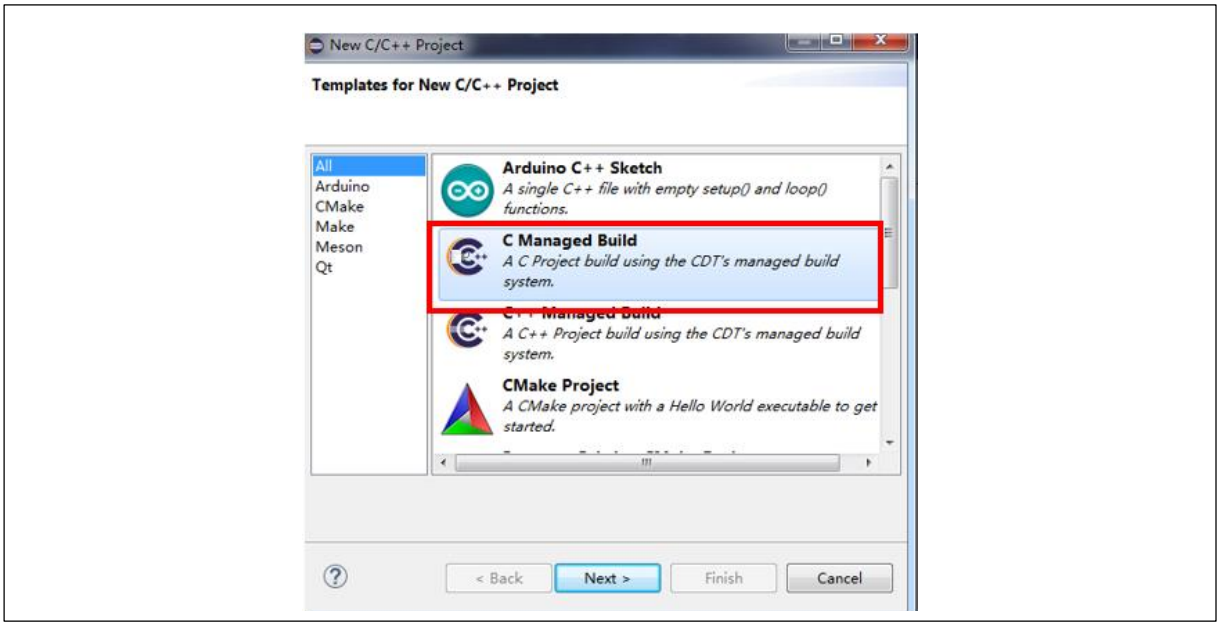
1. Go to ***File->New->C/C++ Protect***

Figure 25. Create new project using Eclipse



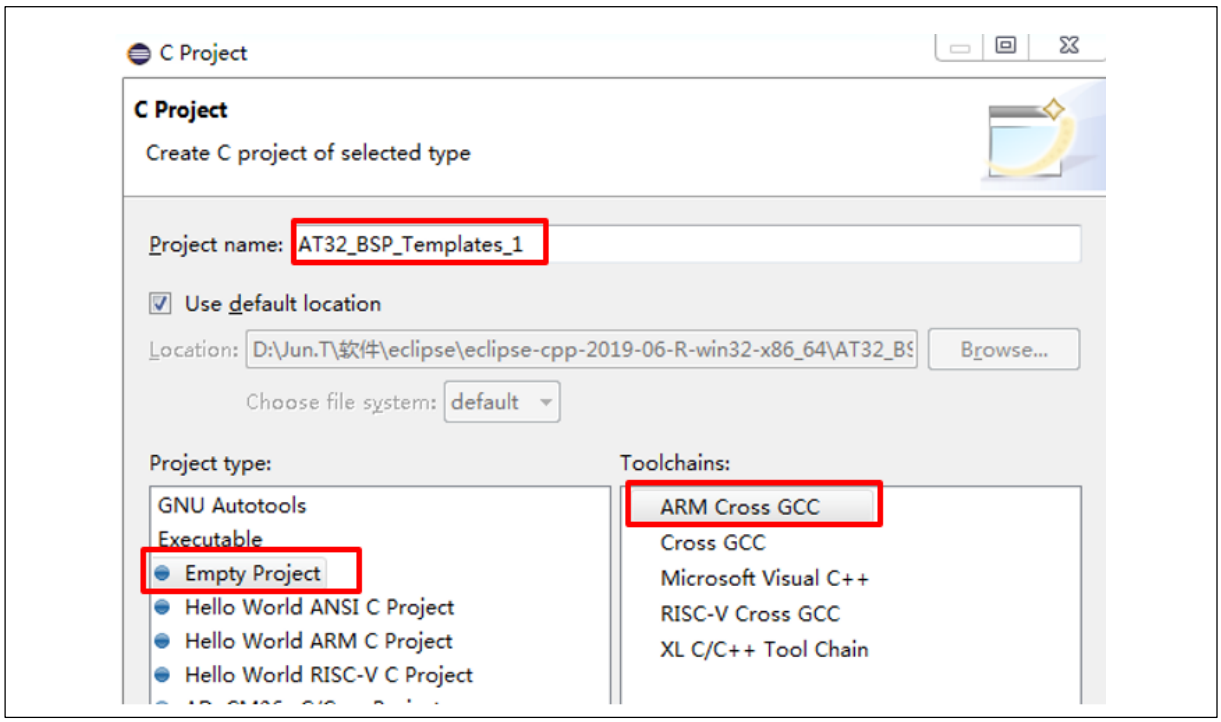
2. C Project->Next

Figure 26. Select C Project->Next



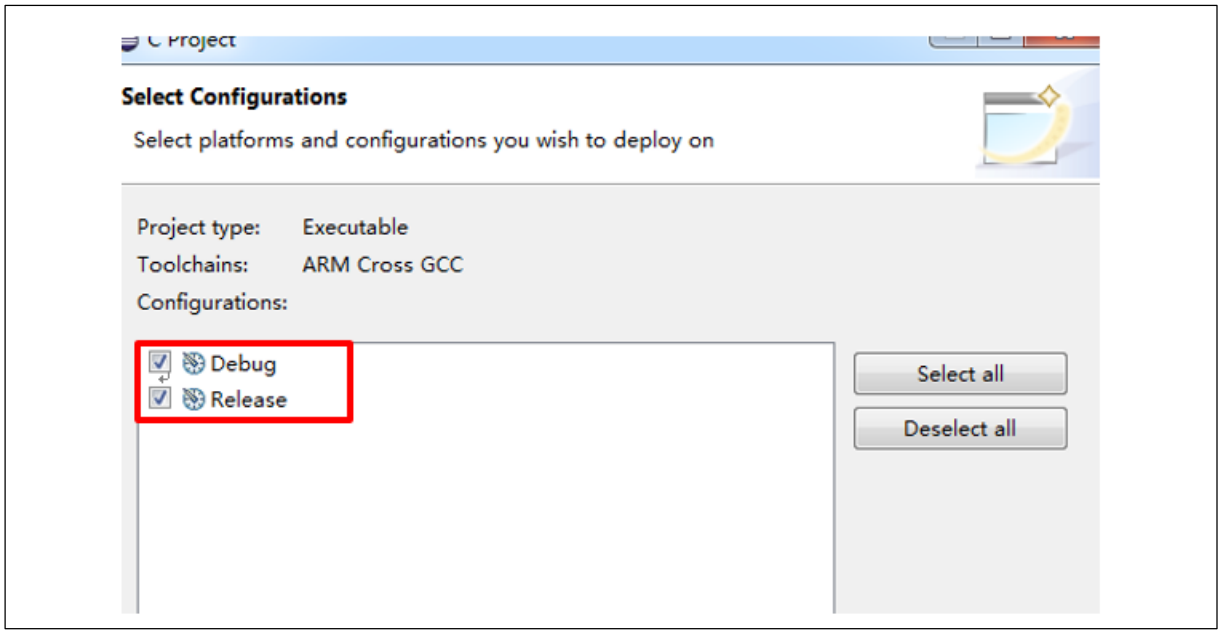
3. Create an ***Empty Project***, select ***ARM Cross GCC***

Figure 27. Create Empty Project



4. Next

Figure 28. Tick Debug\Release

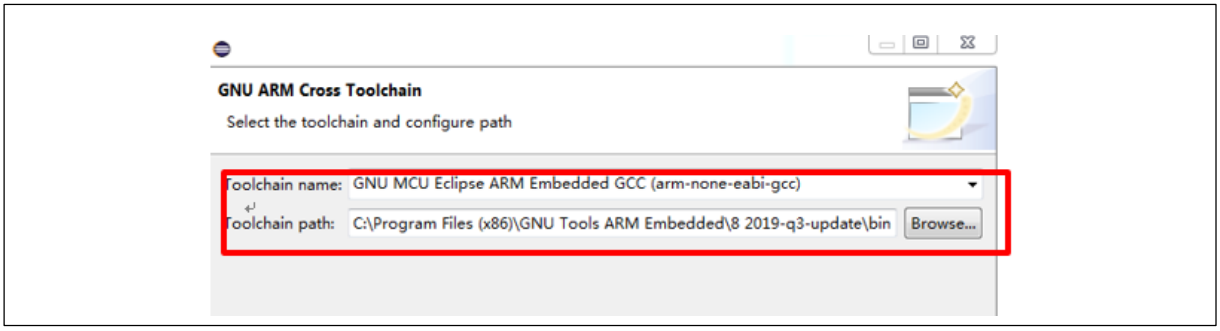


5. ***Next***, select the installed GCC compiler toolchain path.

   Path: C:\Program Files (x86)\GNU Tools ARM Embedded\8 2019-q3-update\bin

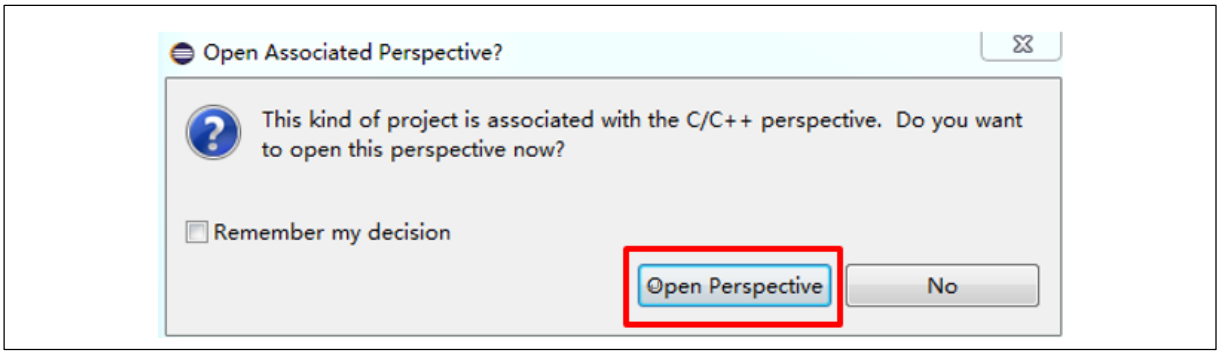
   This is the installation path of compiler. The path here changes with installation path.

Figure 29. Select toolchain path



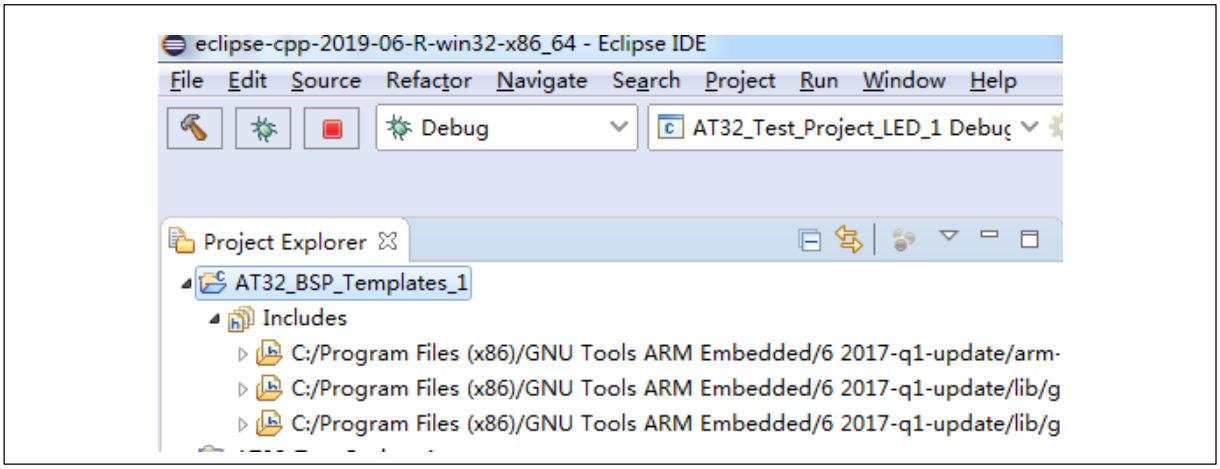
6. Finish

Figure 30. Click Open Perspective



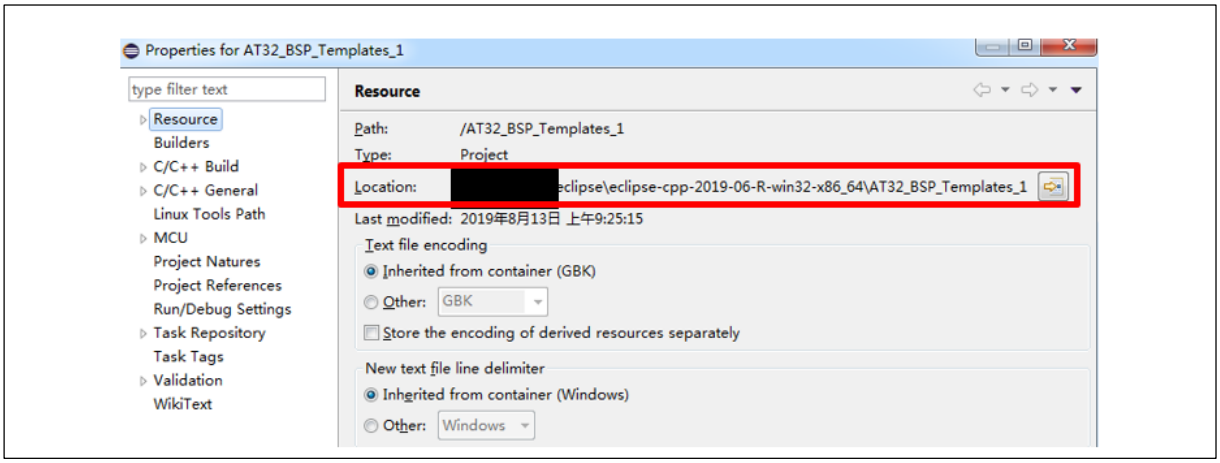
7. A new Empty project is displayed under the project explorer.

Figure 31. Empty project is created



8. You can view the project directory through ***Project -> Properties***.

Figure 32. View project directory



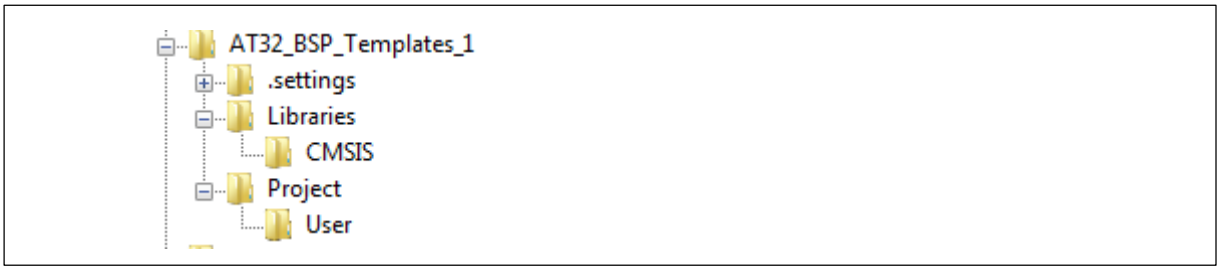
9. Original files under project directory

Figure 33. Original files under project directory

10. Create a directory

    Create new directories such as Libraries, Libraries\CMSIS, Project and Project\User

Figure 34. Create a new directory



11. Copy files (Taking AT32F403 Templates as an example, others are the same)

    Find the new project directory: copy Libraries in BSP, create ***User*** directory, copy AT32_Board directory, and copy .c and .h files under the corresponding Templates directory to ***User***.

    Taking AT32F403 Templates as an example:

    Copy code files:

    Copy *AT32F4xx_StdPeriph_Lib_V1.1.5/Libraries/AT32F4xx_StdPeriph_Driver* to Libraries

    Copy *AT32F4xx_StdPeriph_Lib_V1.1.5/Libraries/ CMSIS/ CM4* to Libraries/CMSIS

    Copy *AT32F4xx_StdPeriph_Lib_V1.1.5/Project /AT32_Board* to Project

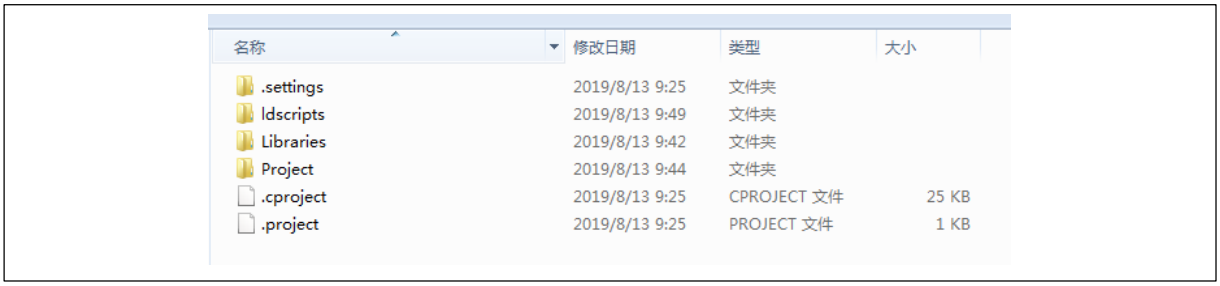
    Copy *AT32F4xx_StdPeriph_Lib_V1.1.5/Project/AT_START_F403/Templates/at32f4xx_it.c at32f4xx_it.h main.c readme.txt* to Project/User

Note: When copying AT32F4xx_StdPeriph_Driver, at32f4xx_acc.c, at32f4xx_comp.c and at32f4xx_ertc.c need to be deleted, otherwise, the compilation error would occur. AT32F403 does not have these peripheral IPs.

Copy script files

Script files corresponding to AT32F403ZGT6 is kept at *ldscripts* of *AT32_Eclipse_Source.zip*. Copy *ldscripts* directory and put it under *AT32_BSP_Templates_1 directory*.

Figure 35. AT32_BSP_Templates_1 directory



Startup .S files

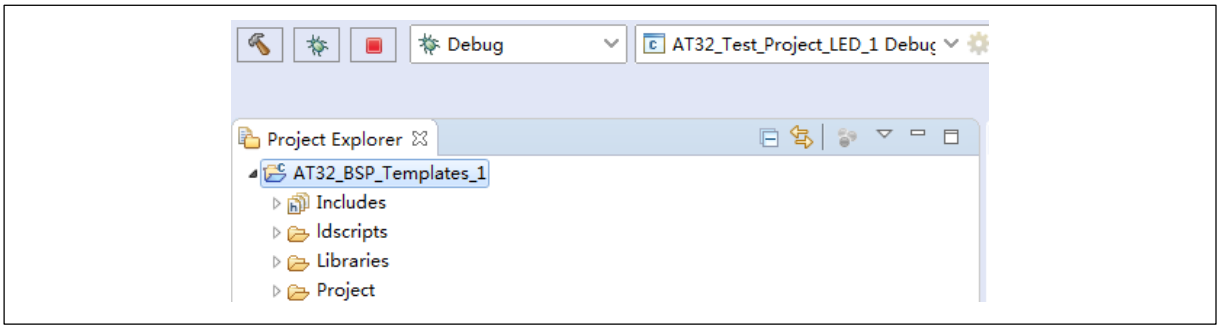
The GCC version of .S files need to be used, and .S should be capitalized, instead of the ones in MDK or IAR.

The .S example code is located under GCC directory in the *AT32_Eclipse_Source.zip*. Copy one of the .S files corresponding to certain product model to Libraries/CMSIS for direct use.

12. Project refresh

Press F5 to refresh Eclipse project directory to find out the previously created directories and copied files

Figure 36. Refresh project directory



4 Compile configuration

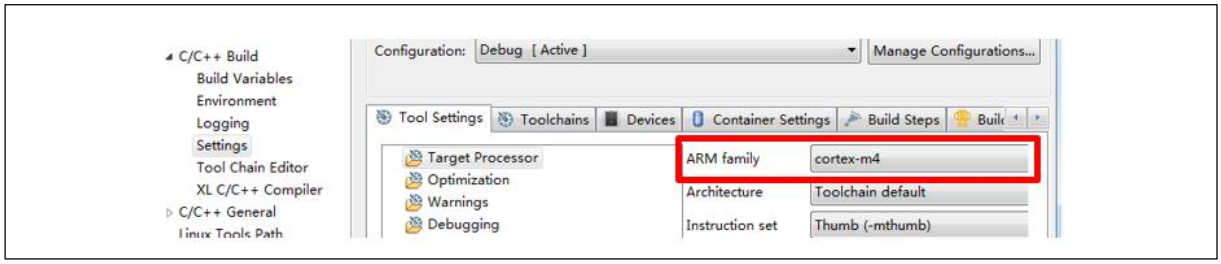
This section describes how to configure compiling options.

- Chip configuration
- Header file path configuration
- Macro configuration
- Script file configuration

4.1 AT32F403 configuration and compiling

1. ***Project->Properties -> C/C++ Build -> Settings→Target Processor*** → select ***cortex-m4*** in ***ARM family***

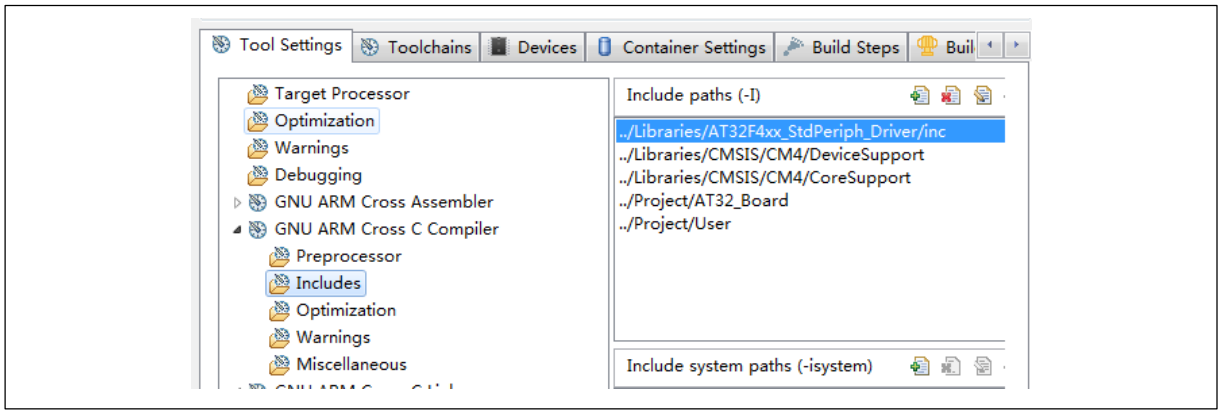
Figure 37. AT32F403 ARM family selection



2. Header file setup: ***Project->Properties -> C/C++ Build -> Setting***

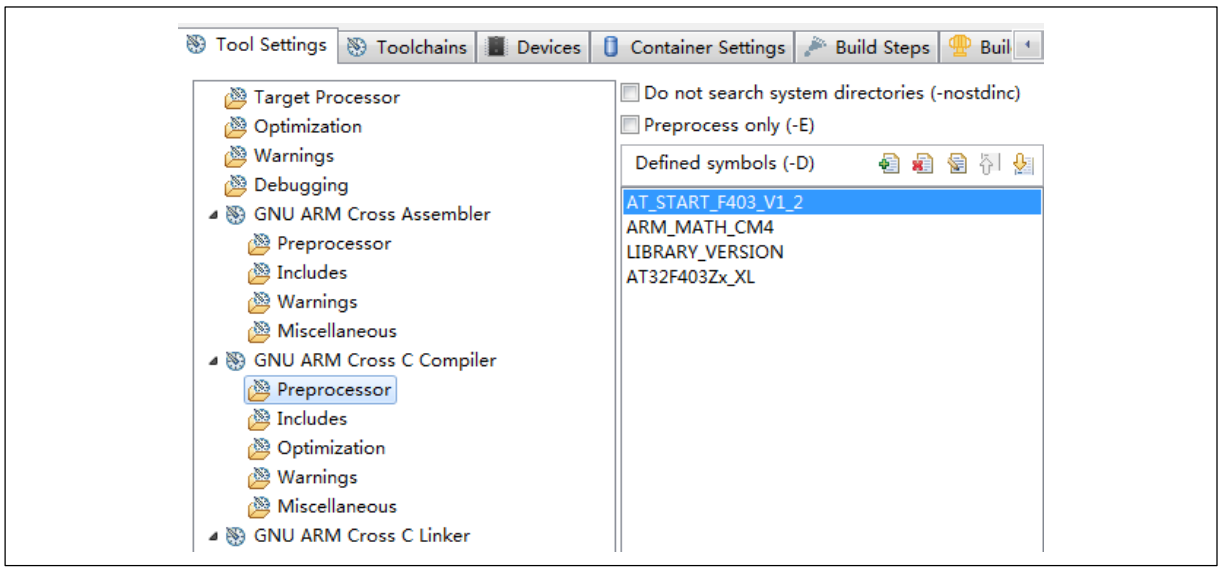
   GNU ARM Cross C Compiler→Includes→ Add header file path →***Apply***

Figure 38. AT32F403 header file configuration



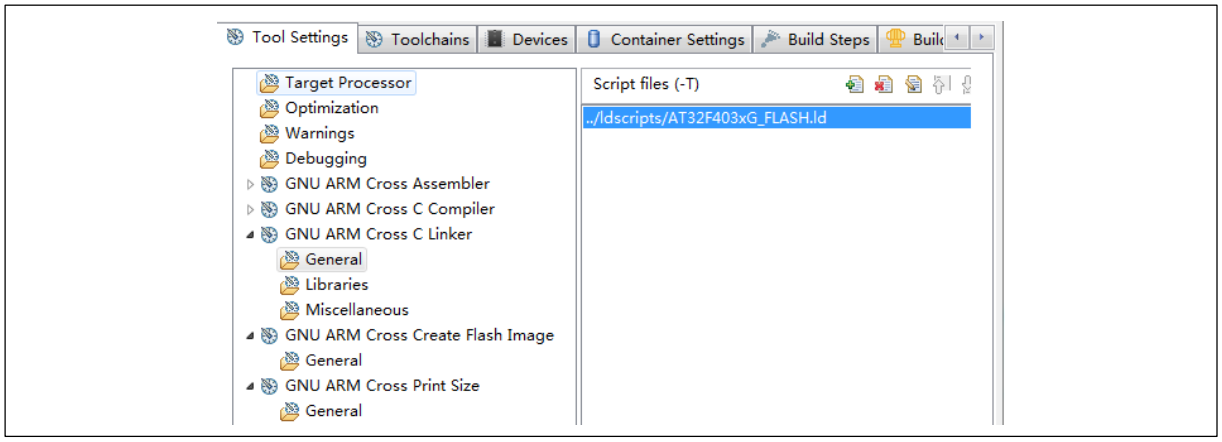
3. Add macro: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Compiler → Preprocessor →*** Add macro

Figure 39. AT32F403 macro definition



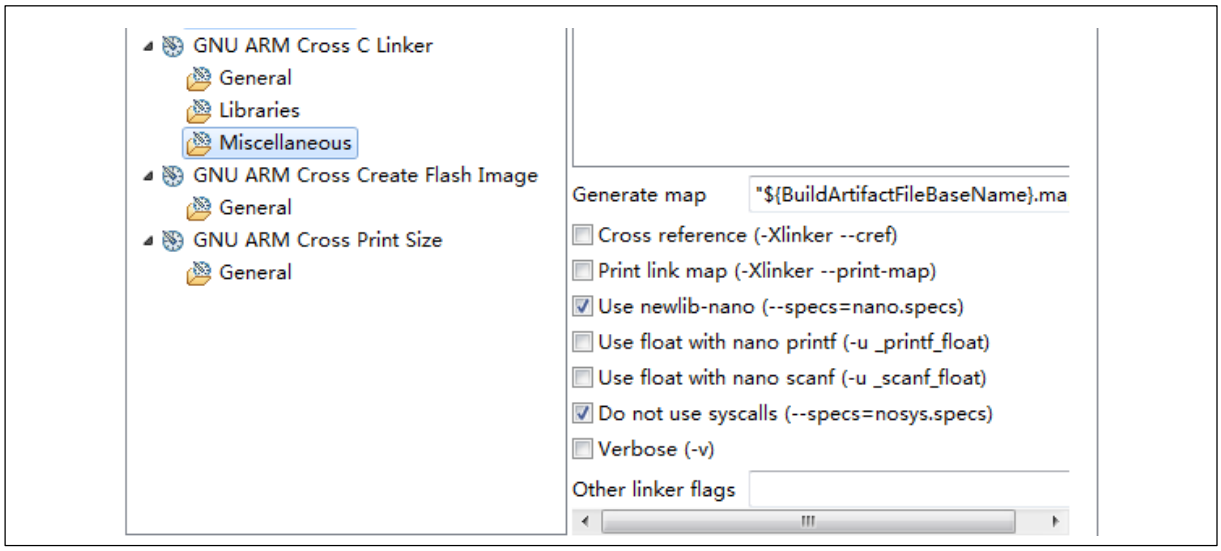
4. Add script files:***Project->Properties -> C/C++ Build -> Setting-> GNU ARM Cross C Linker→ General →*** select script files

Figure 40. Add script files for AT32F403



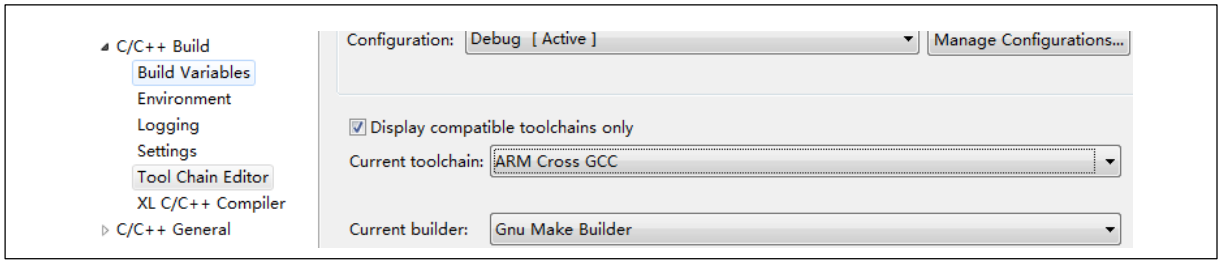
5. Set up other options: ***Project->Properties -> C/C++ Build -> Setting-> GNU ARM Cross C Linker → Miscellaneous →***Tick ***Use newlib-nano & Do not use syscalls***

Figure 41. Other configurations for AT32F403



6. Make compiler tool selection: ***Project->Properties -> C/C++ Build -> Tool Chain Editor→ Current toolchains*** → select ***ARM Cross GCC*** from drop-down menu →***Current builder***→ select ***GNU Make Builder*** from drop-down menu

Figure 42. AT32F403 Make compiler tool selection

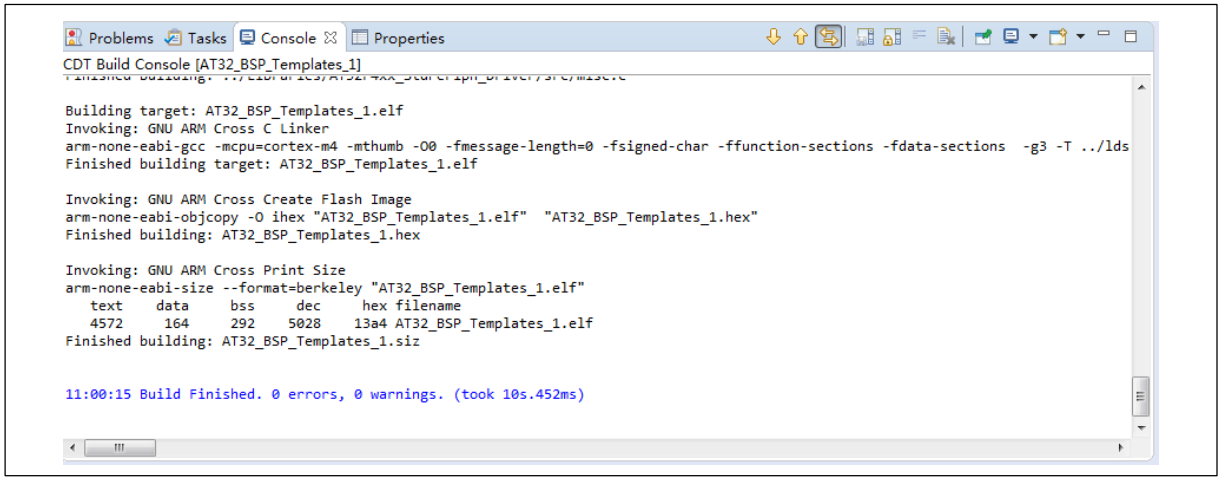


7. After finishing above setup, start compiling code:

   Project → Build Project → start compiling

   Click ***Console***, the compiling details will be displayed.

Figure 43. AT32F403 code compiling



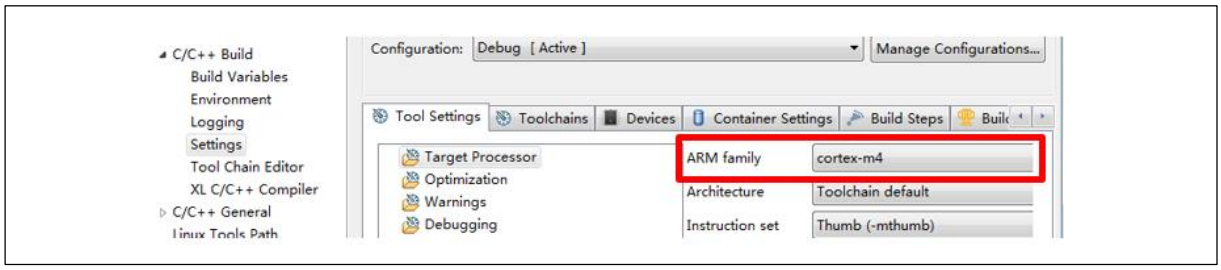
4.2 AT32F413 configuration and compiling

To use a new project, the procedures below need to be followed:

1. Project->Properties -> C/C++ Build -> Setting

→Target Processor → select ***cortex-m4*** in ***ARM family***

Figure 44. AT32F413 ARM family selection

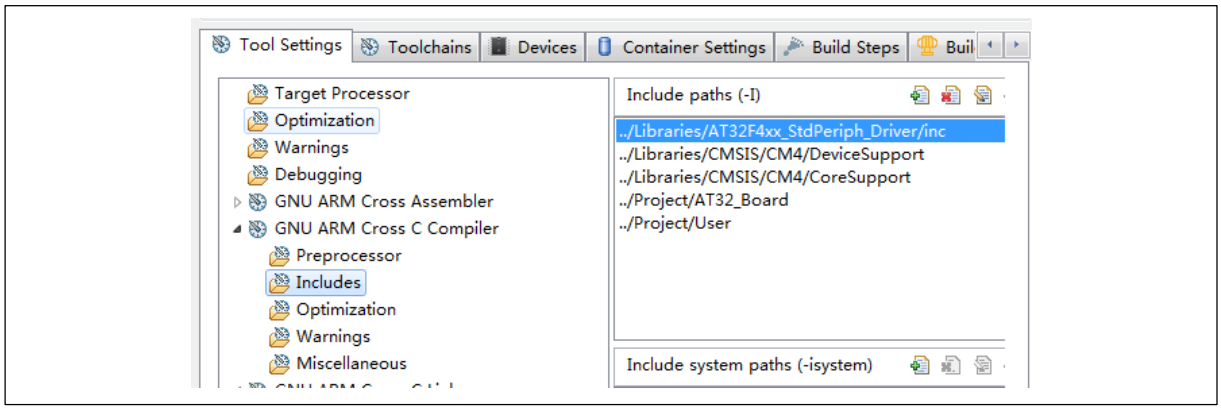


2. Header file configuration: ***Project->Properties -> C/C++ Build -> Setting→***

GNU ARM Cross C Compiler → Includes → Add header file path

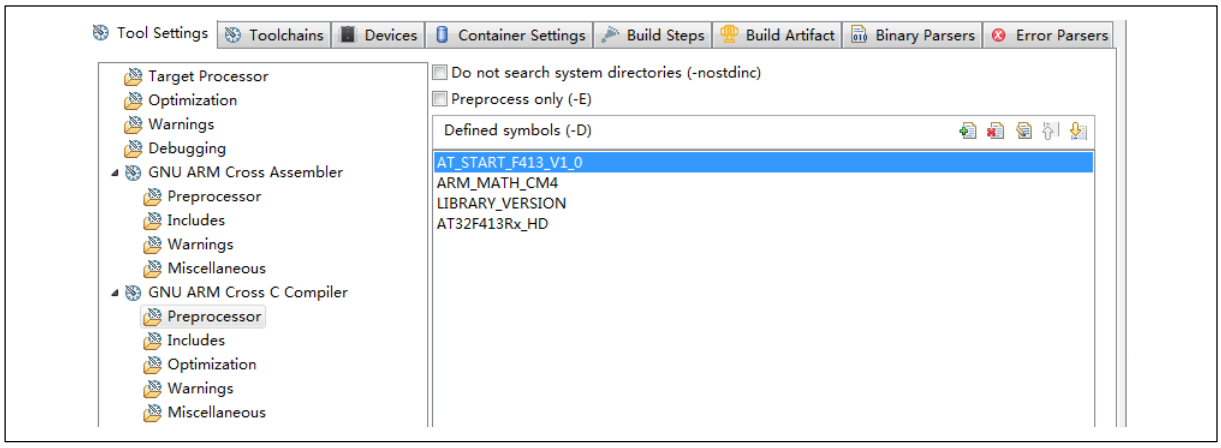
→ Apply

Figure 45. AT32F413 header file setup



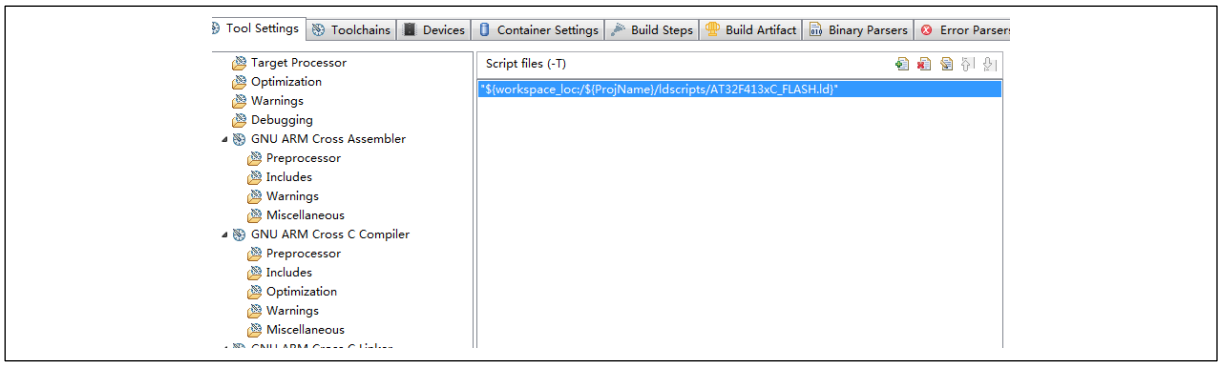
3. Add macro: ***Project->Properties -> C/C++ Build -> Setting→ GNU ARM Cross C Compiler → Preprocessor →*** Add macro

Figure 46. Add macro for AT32F413



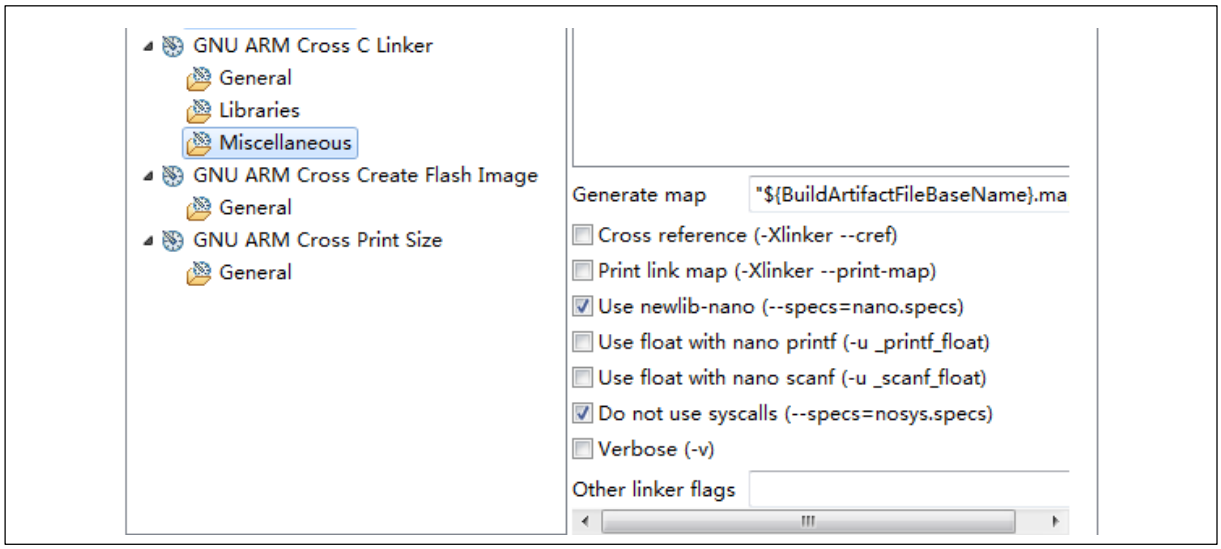
4. Add script files: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Linker → General →*** Select script files

Figure 47. Add script files for AT32F413



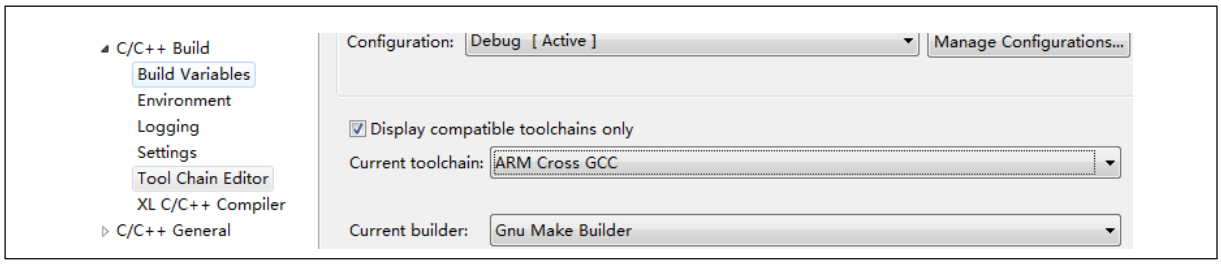
5. Other configuration: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Linker → Miscellaneous →***Tick ***Use newlib-nano & Do not use syscalls***

Figure 48. Other configuration for AT32F413



6. Make compiler toolchain selection: ***Project->Properties -> C/C++ Build -> Tool Chain Editor Current toolchains →*** Select ARM Cross GCC → Current builder→ Select ***GNU Make Builder***

Figure 49. AT32F413 Make compiler toolchain selection

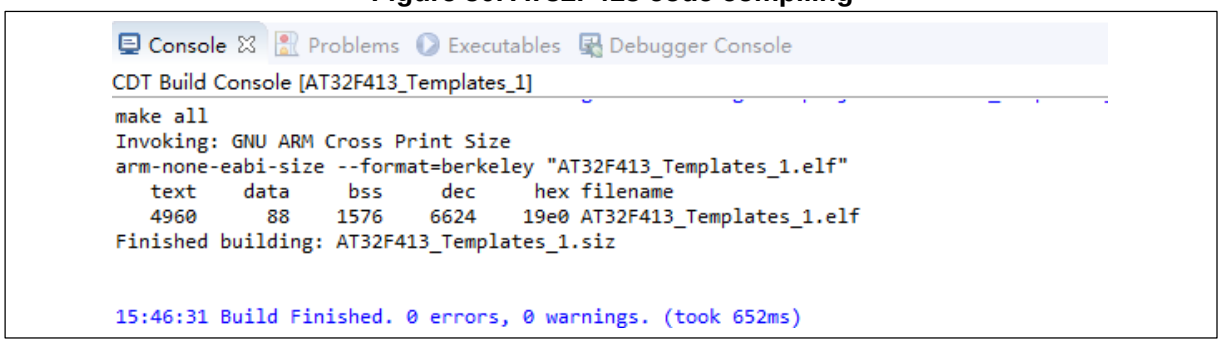


7. After completing above operations, start compiling code:

   Project → Build Project → Start compiling

   Click ***Console***, the compiling information will be displayed.

Figure 50. AT32F413 code compiling

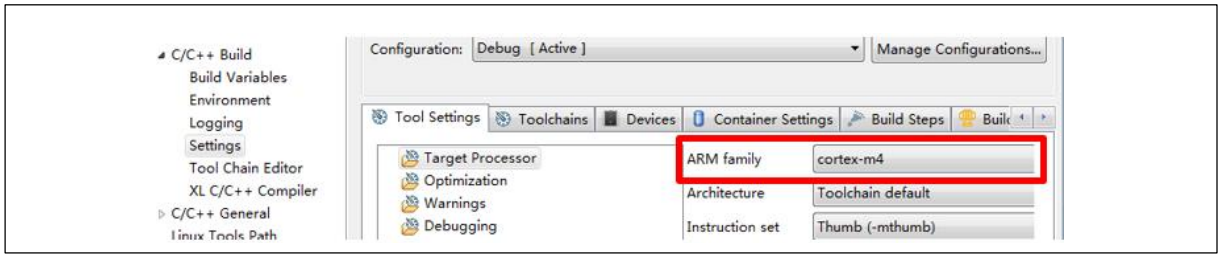


4.3 AT32F415 configuration and compiling

To use new AT32F415 project, proceeds as follows:

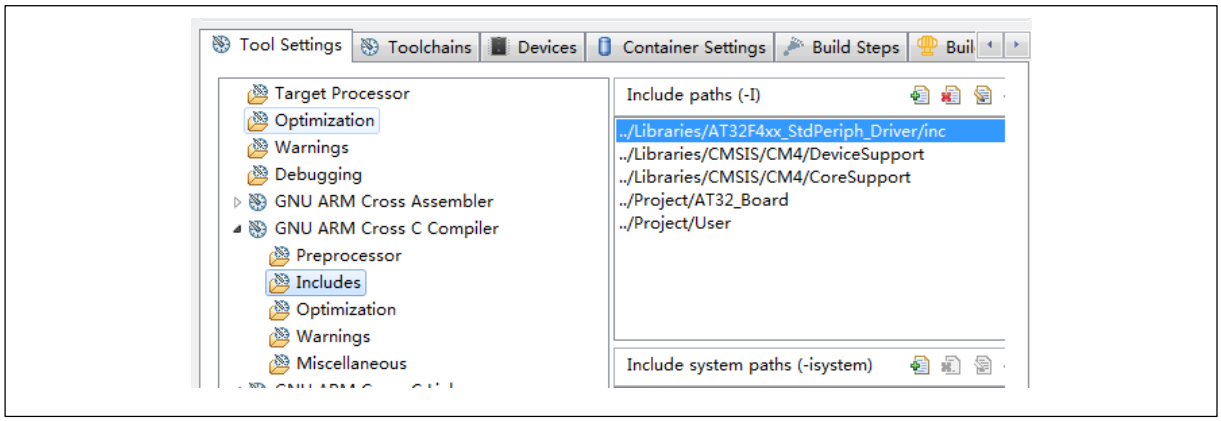
1. Project->Properties -> C/C++ Build -> Setting → Target Processor → Select ***cortex-m4*** in ***ARM family***

Figure 51. AT32F415 ARM family selection



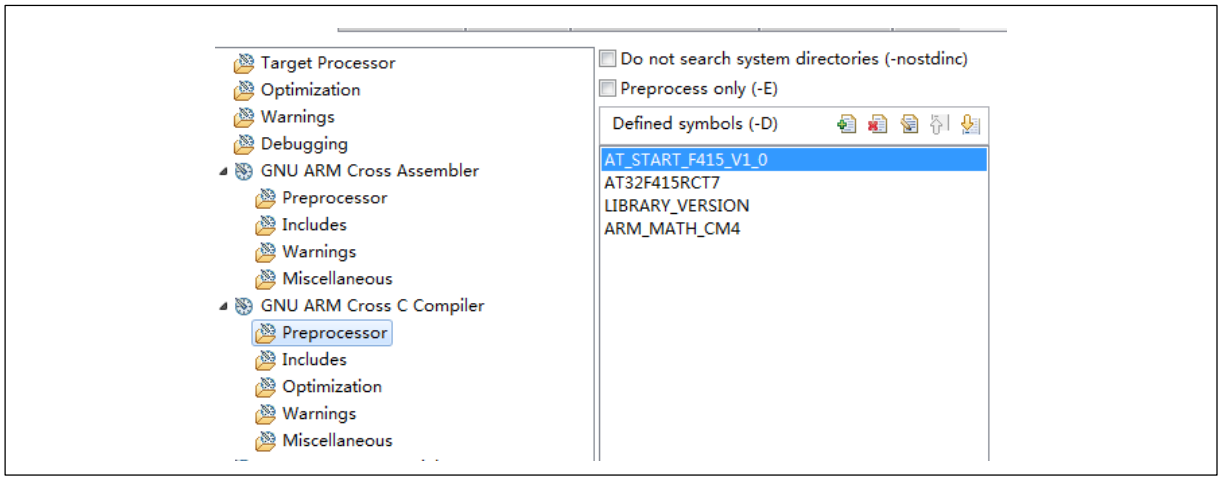
2. Header file configuration: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Compiler → Includes → Add header file path →Apply***

Figure 52. AT32F415 header file configuration



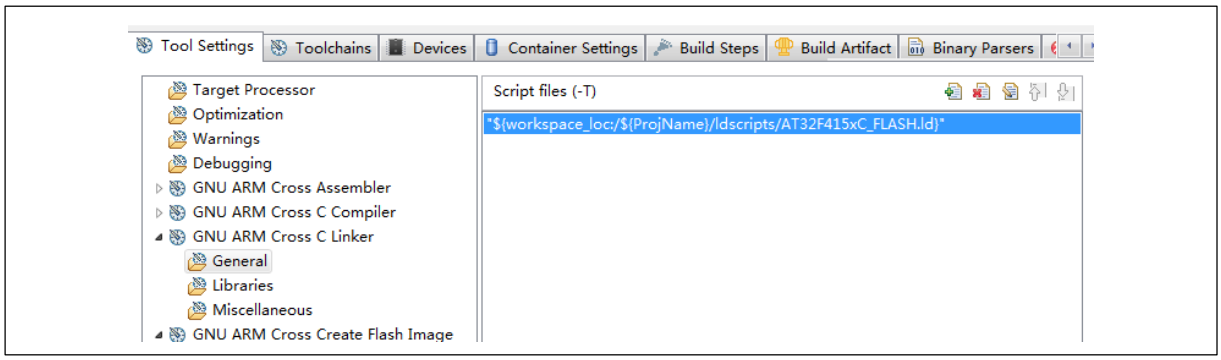
3. Add macro: ***Project->Properties -> C/C++ Build -> Setting → GNU ARM Cross C Compiler → Preprocessor*** → Add macro

Figure 53. Add macro for AT32F415



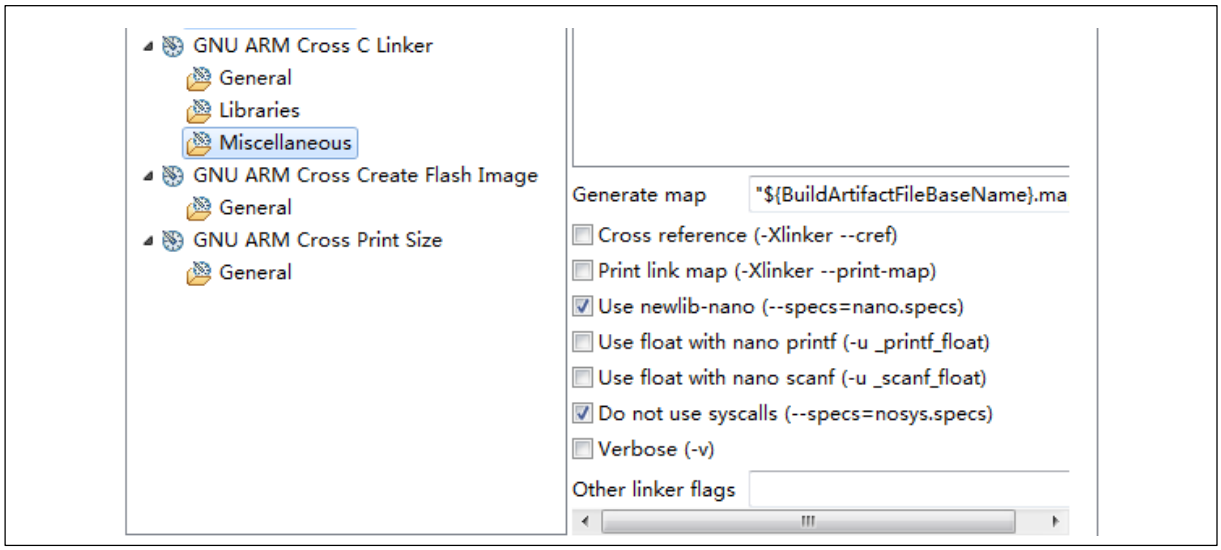
4. Add script files: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Linker → General →*** Select script files

Figure 54. Add script files for AT32F415



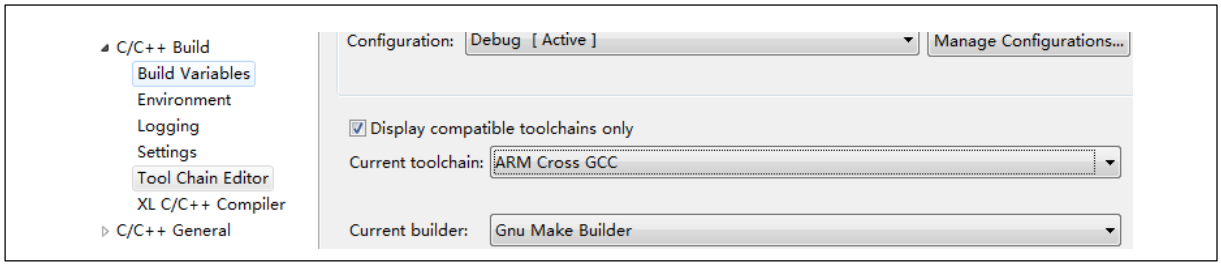
5. Other configuration: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Linker → Miscellaneous →*** Tick ***Use newlib-nano & Do not use syscalls***

Figure 55. Other configuration for AT32F415



6. Make compiler toolchain selection: ***Project->Properties -> C/C++ Build -> Tool Chain Editor Current toolchains → Select ARM Cross GCC →Current builder →*** Select ***GNU Make Builder***

Figure 56. AT32F415 Make compiler toolchain selection

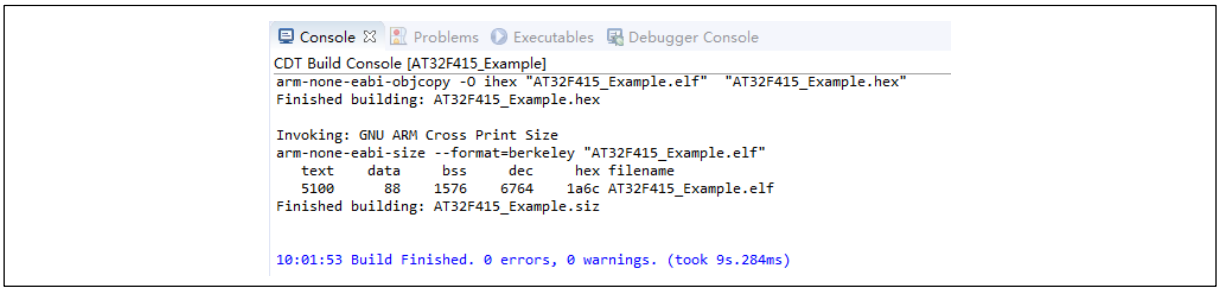


7. After completing above configuration, start compiling code:

   Project → Build Project → Start compiling

   Click ***Console***, the compiling information will be displayed.

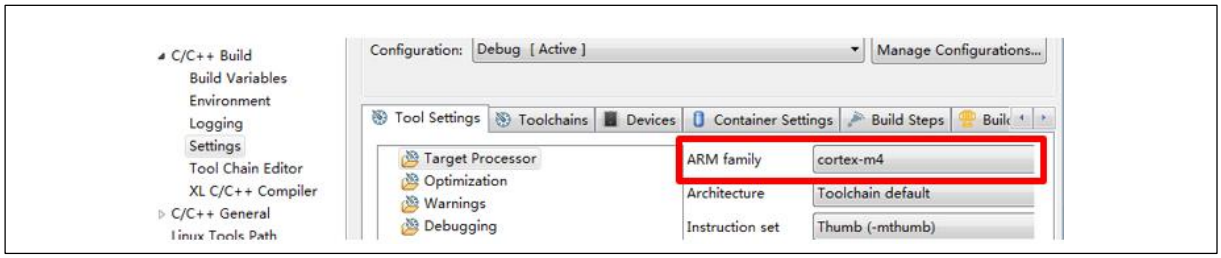
Figure 57. AT32F415 code compiling



4.4 AT32F403A/407 configuration and compiling

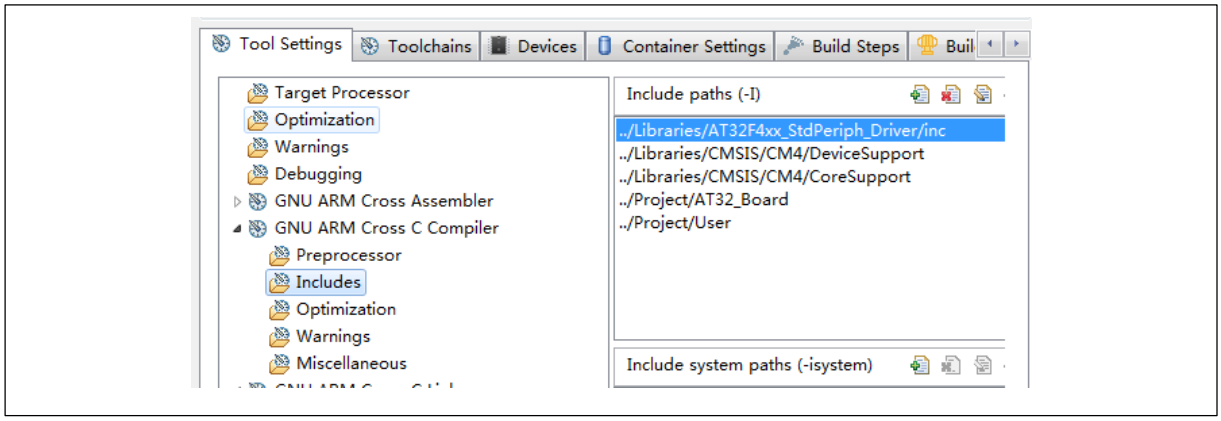
8. Project->Properties -> C/C++ Build -> Setting → Target Processor → Select ***cortex-m4*** in ***ARM family***

Figure 58. AT32F403A/407 ARM family selection



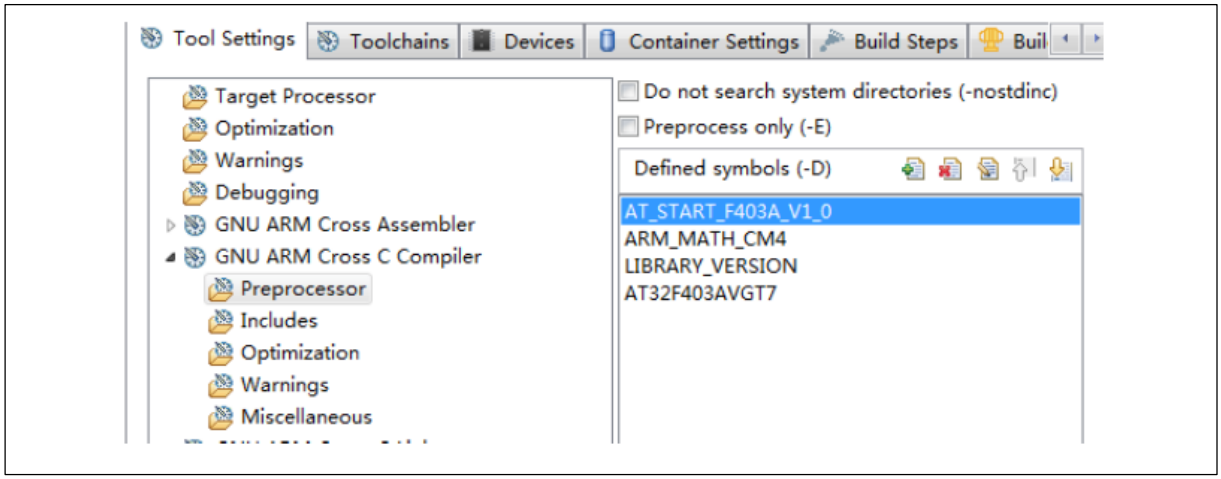
9. Header file configuration: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Compiler → Includes → Add header file path →Apply***

Figure 59. AT32F403A/407 header file configuration



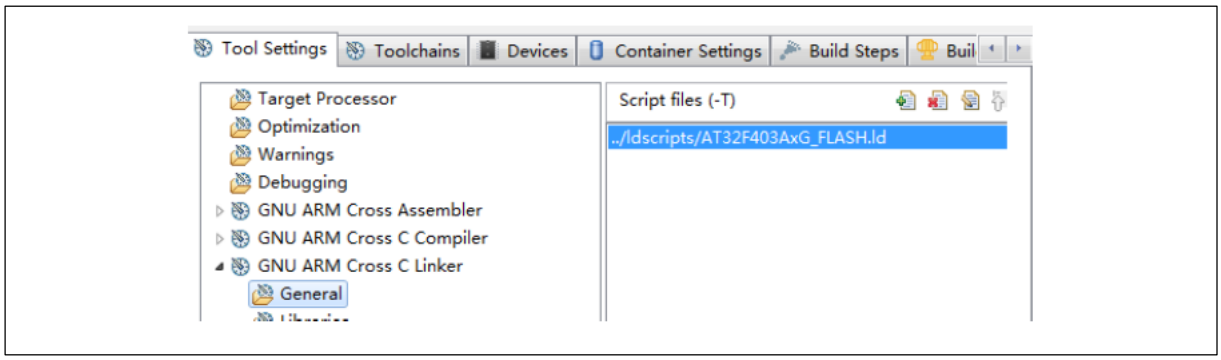
10. Add macro: ***Project->Properties -> C/C++ Build -> Setting → GNU ARM Cross C Compiler → Preprocessor*** → Add macro

Figure 60. Add macro for AT32F403A/407



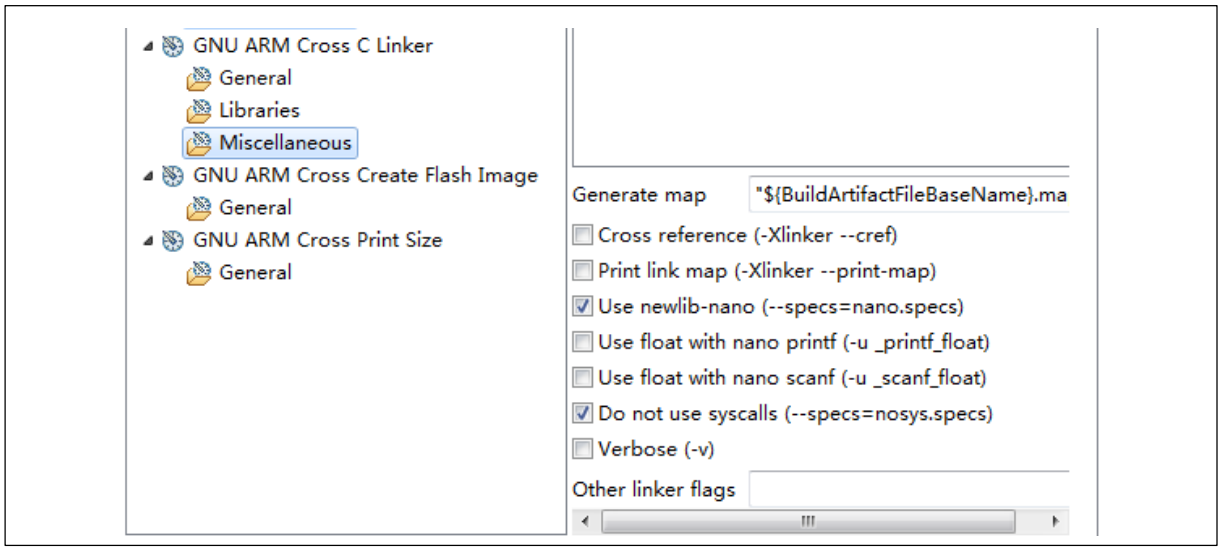
11. Add script files: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Linker → General →*** Select script files

Figure 61. Add script files for AT32F403A/407



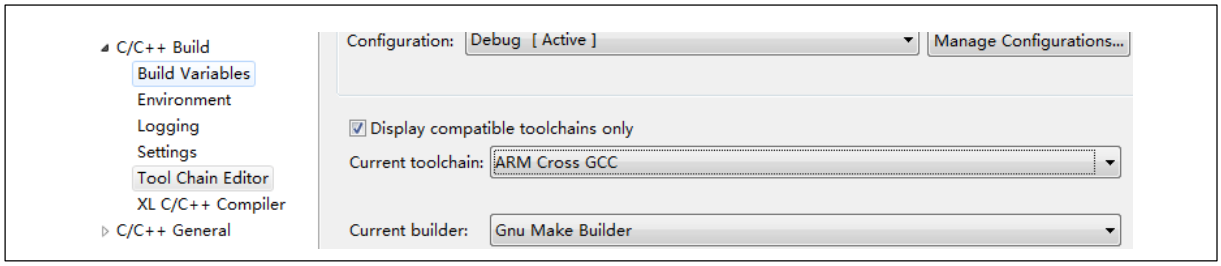
12. Other configuration: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Linker → Miscellaneous →*** Tick ***Use newlib-nano & Do not use syscalls***

Figure 62. Other configuration for AT32F403A/407



13. Make compiler toolchain selection: ***Project->Properties -> C/C++ Build -> Tool Chain Editor Current toolchains → Select ARM Cross GCC →Current builder →*** Select ***GNU Make Builder***

Figure 63. AT32F403A/407 Make compiler toolchain selection

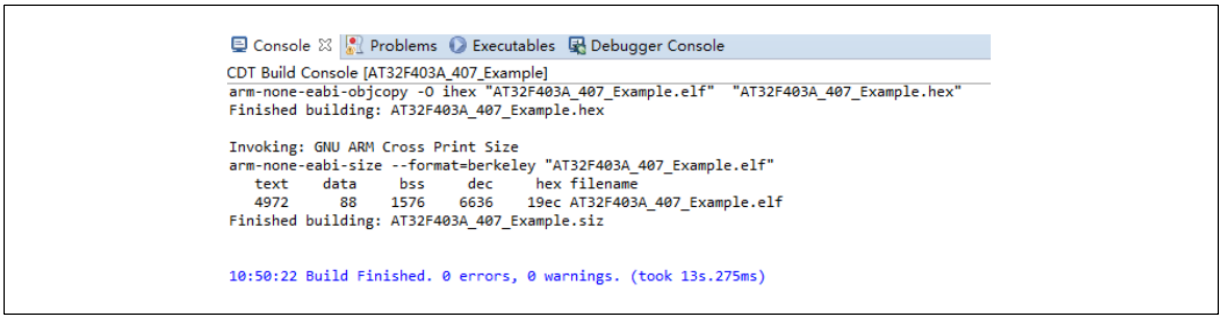


14. After completing above configuration, start compiling code:

    Project → Build Project → Start compiling

    Click ***Console***, the compiling information will be displayed.

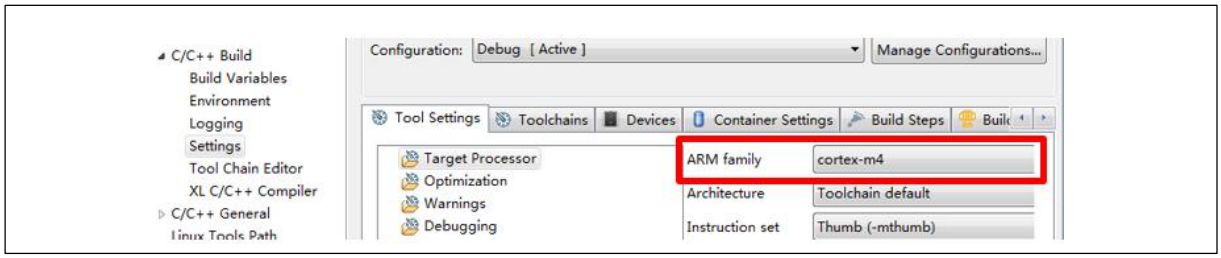
Figure 64. AT32F403A/407 code compiling



4.5 AT32F421 configuration and compiling

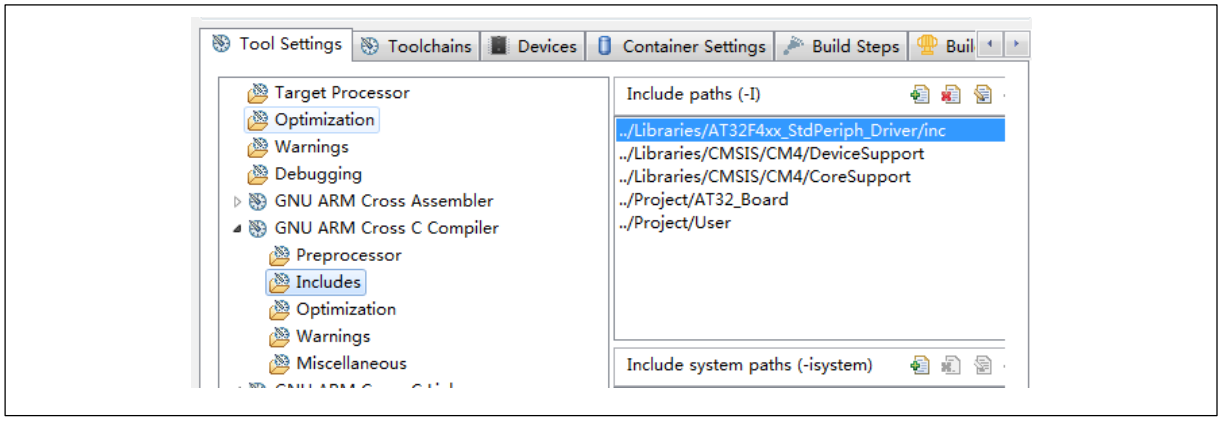
15. Project->Properties -> C/C++ Build -> Setting → Target Processor → Select ***cortex-m4*** in ***ARM family***

Figure 65. AT32F403A/407 ARM family selection



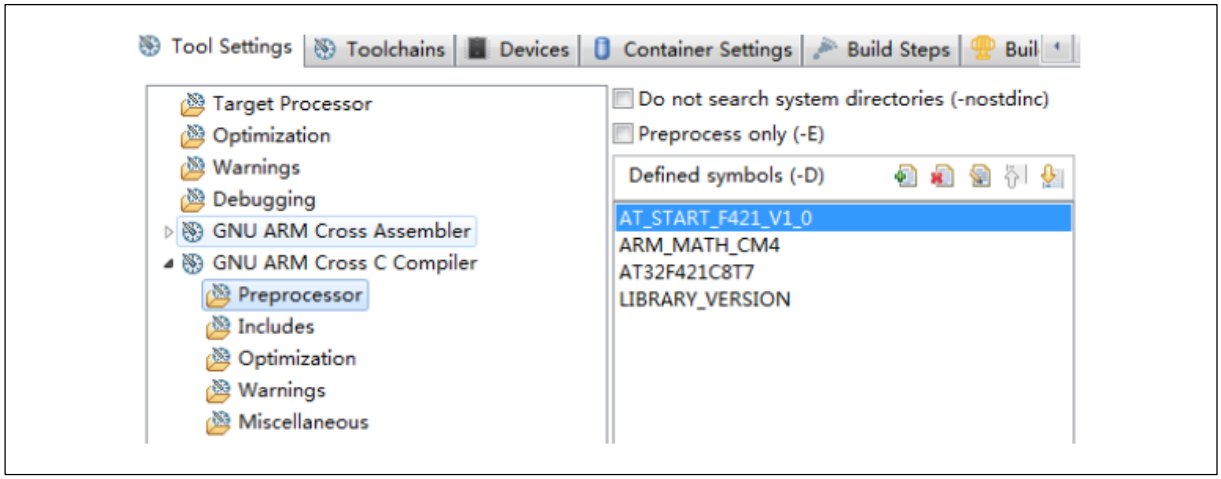
16. Header file configuration: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Compiler → Includes → Add header file path →Apply***

Figure 66. AT32F421 header file configuration



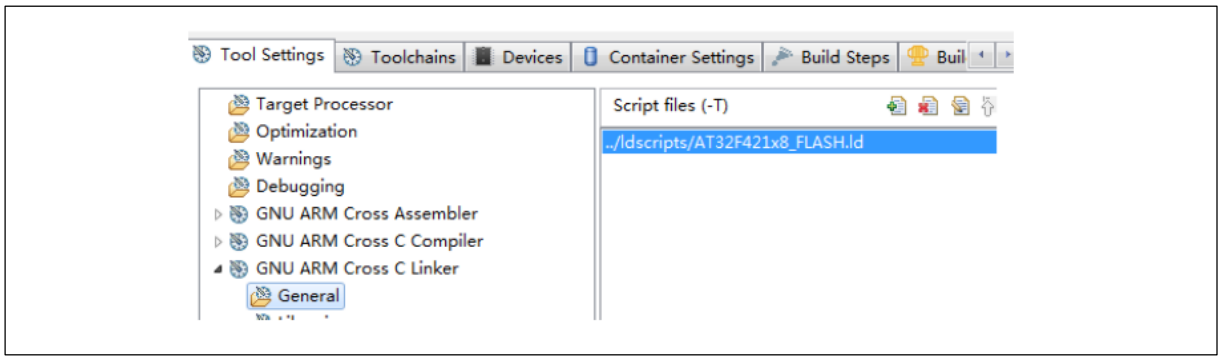
17. Add macro: ***Project->Properties -> C/C++ Build -> Setting → GNU ARM Cross C Compiler → Preprocessor*** → Add macro

Figure 67. Add macro for AT32F421



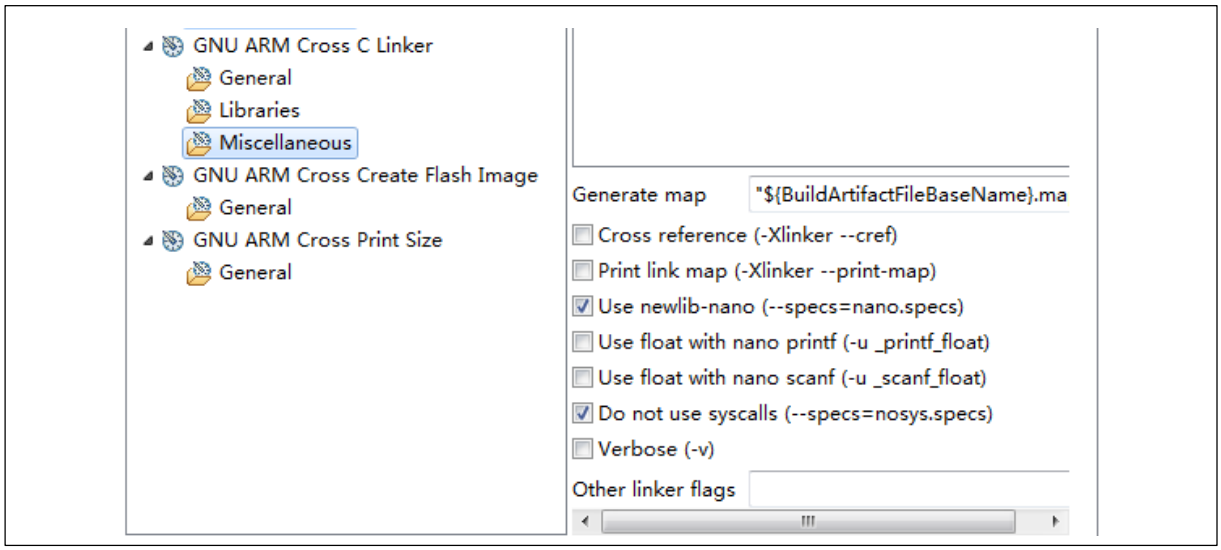
18. Add script files: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Linker → General →*** Select script files

Figure 68. Add script files for AT32F421



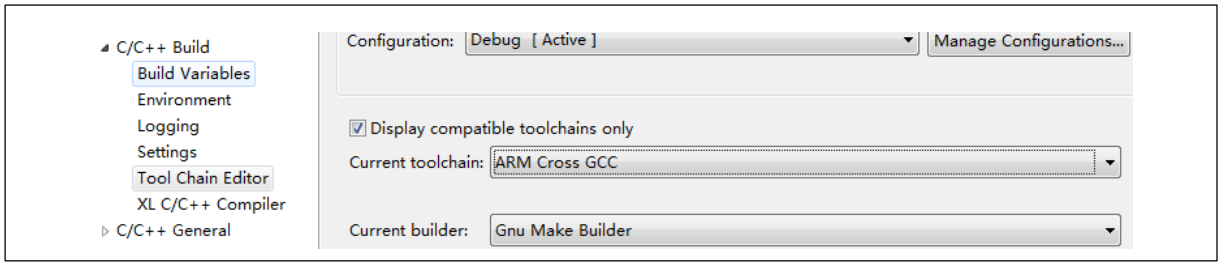
19. Other configuration: ***Project->Properties -> C/C++ Build -> Setting -> GNU ARM Cross C Linker → Miscellaneous →*** Tick ***Use newlib-nano & Do not use syscalls***

Figure 69. Other configuration for AT32F421



20. Make compiler toolchain selection: ***Project->Properties -> C/C++ Build -> Tool Chain Editor Current toolchains → Select ARM Cross GCC →Current builder →*** Select ***GNU Make Builder***

Figure 70. AT32F421 Make compiler toolchain selection

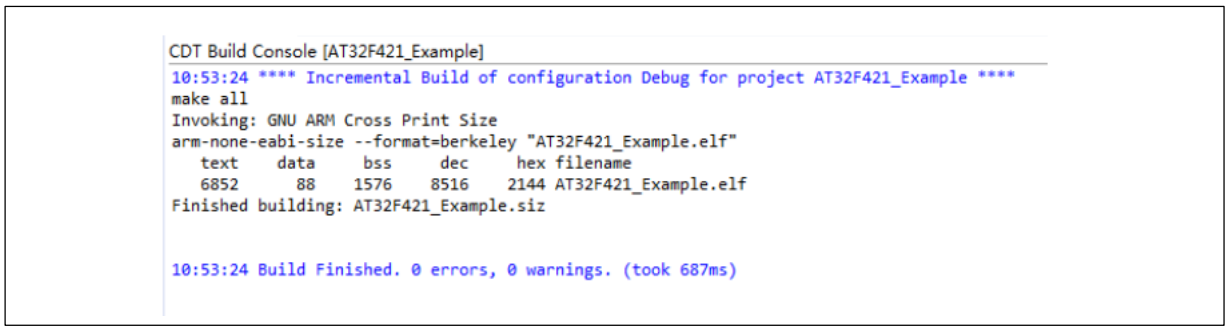


21. After completing above configuration, start compiling code:

    Project → Build Project → Start compiling

    Click ***Console***, the compiling information will be displayed.

Figure 71. AT32F421 code compiling



5 Eclipse+JLink debug

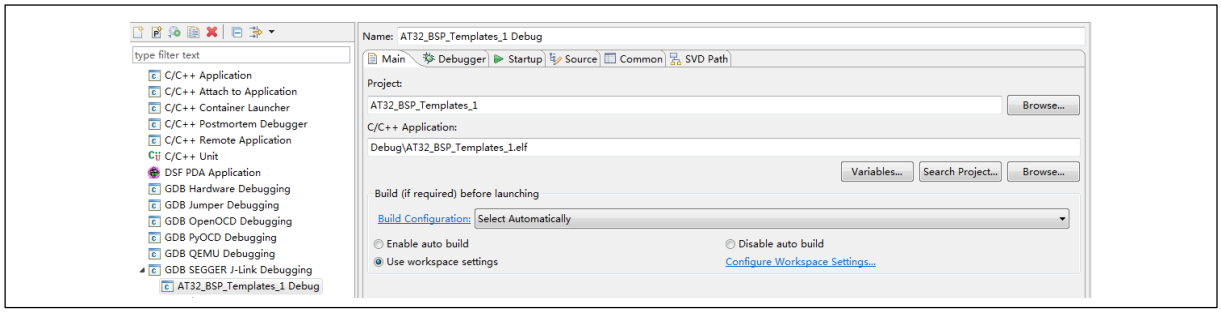
The following configuration are needed for Eclipse + JLink debug.

- JLink configuration
- GDB configuration
- SVD peripheral register configuration

5.1 Debug configuration

1. ***Run → Debug Configurations → GDB SEGGER J-Link Debugging → New Configuration***

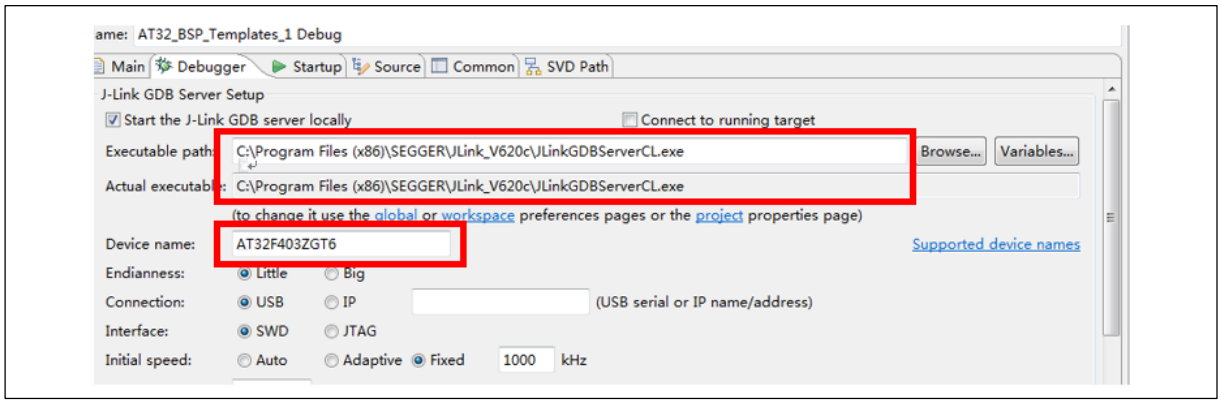
Figure 72. Debug configuration



2. Go to ***AT32_BSP_Templates_1_Debug → Debugger***:

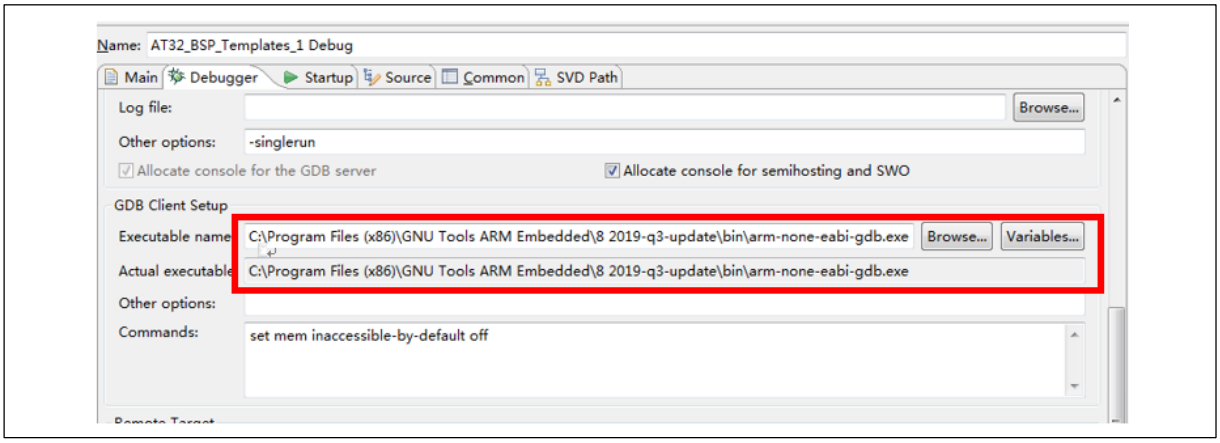
   Configure ***JlinkGDBServerCL***, fill in ***device name***, such as AT32F403ZGT6, AT32F413RCT7 or AT32F415RCT7

Figure 73. Fill in device name



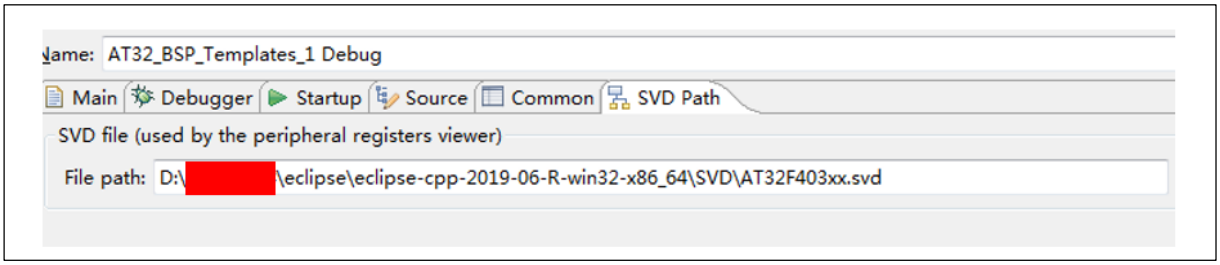
3. To set up GDB, select ***arm-none-eabi-gdb.exe*** under GCC directory.

Figure 74. Set up GDB



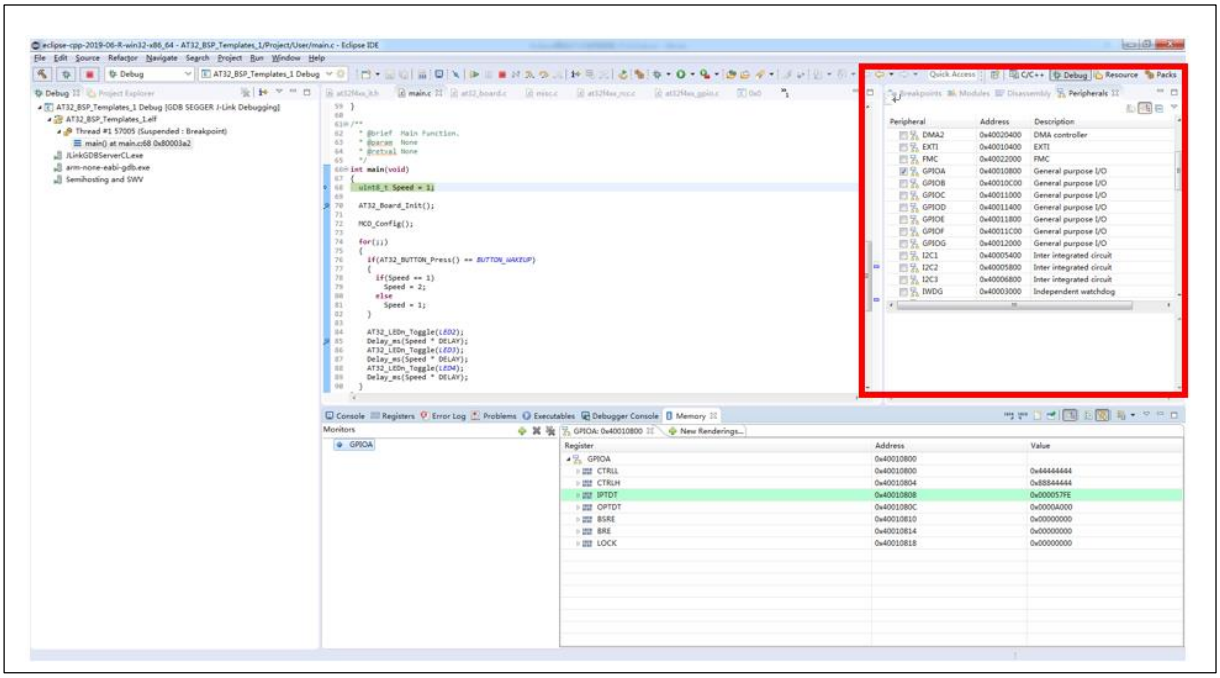
4. Select SVD Path for debug register description. You can use the svd file in keil. When AT32 keil Packet is installed, the svd file can be automatically copied to keil directory.
   AT32_Eclipse_Source.zip contains svd, go to ***AT32_BSP_Templates_1_Debug→SVD Path →select SVD AT32F403xx.svd***

Figure 75. SVD Path selection



5. Debug configuration is done, then go to ***→Apply →Debug***.

Figure 76. Debug configuration finished



6 Eclipse+ATLink debug

This section presents how to debug AT32 using OpenOCD + Eclipse + ATLink. For details on ATLink, please refer to *AT-Link_User_Manual_SC.pdf.*

This section contains the following contents:

- Eclipse Openocd configuration
- GDB configuration
- SVD peripheral register configuration

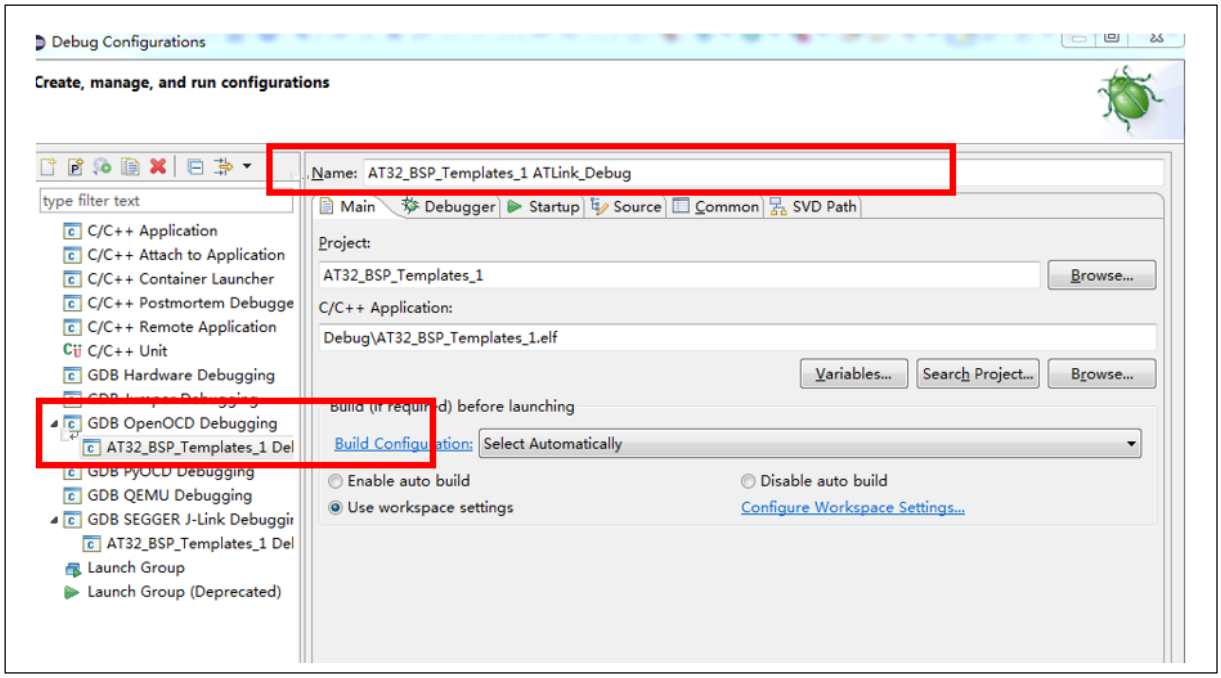
Unzip the OpenOCD package file in *AT32_Eclipse_Packet*, and it includes five directories, the bin file is executable (only support X64 environment), and scripts for configuration files.

Figure 77. OpenOCD directory

6.1 Debug configuration

1. Run → Debug Configurations → GDB OpenOCD Debugging → New Configuration

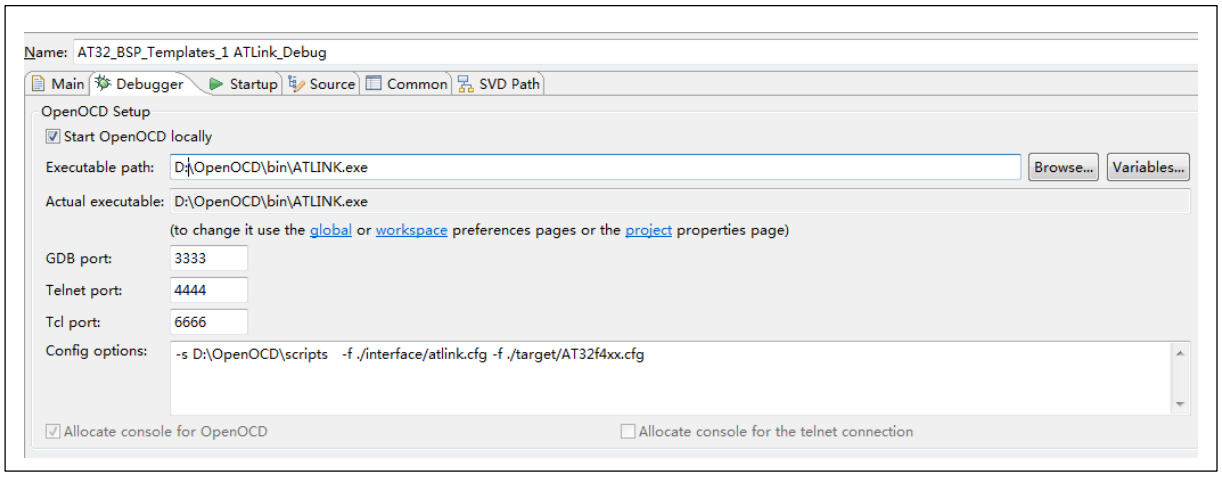
Figure 78. Create Debug configuration



2. To configure OpenOCD path, go to ***AT32_BSP_Templates_1_ATLink_Debug → Debugger***, select *ATLINK.exe* under *OpenOCD\bin*,

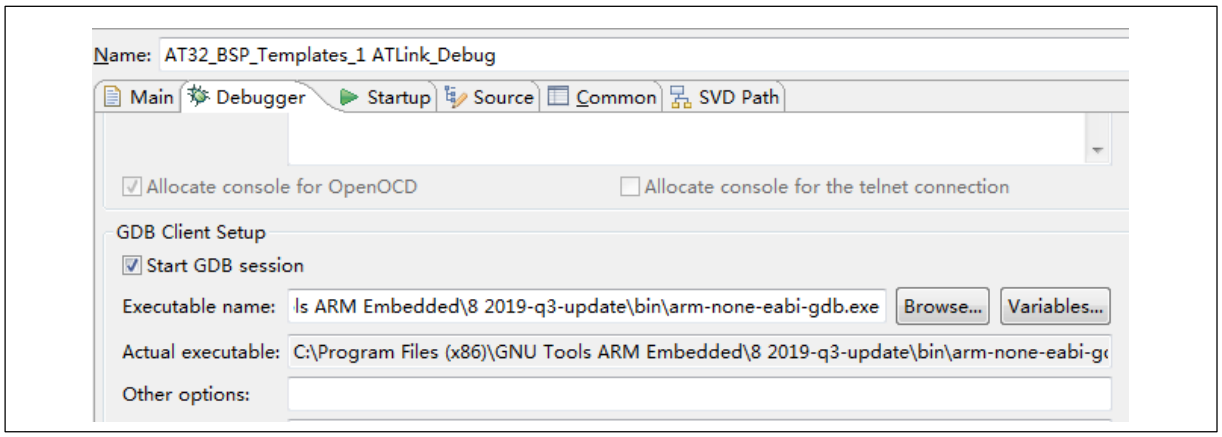
   In ***config options***, designate the directory searched and configuration files to be used, with -s meaning search directory (depend on actual OpenOCD path), -f for the configuration file to be used (-f ./interface/atlink.cfg -f ./target/AT32f4xx.cfg)

Figure 79. Set up OpenOCD path



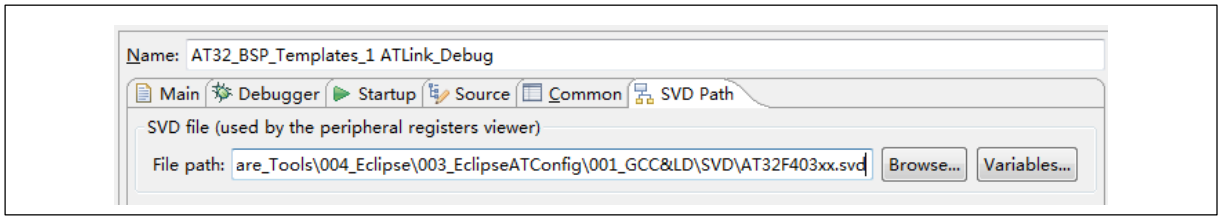
3. Configure GDB, select ***arm-none-eabi-gdb.exe*** under GCC directory.

Figure 80. Setup GDB



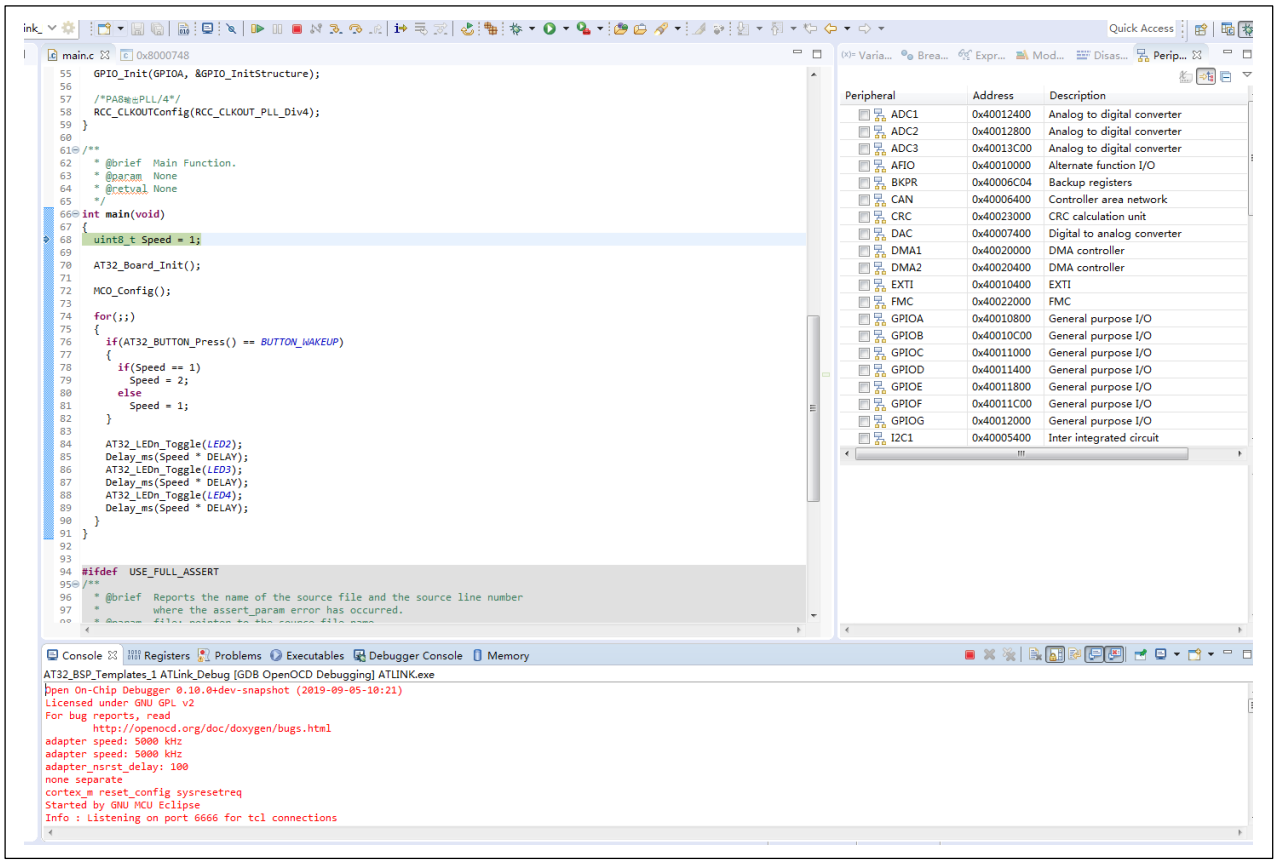
4. Add SVD files

Figure 81. Add SVD files



5. After completing Debug configuration, go to ***Apply →Debug*** to start debugging.

Figure 82. Configuration finished



7 Examples

The example codes of AT32F403, AT32F413 and AT32F415 are included in Example routine, which can be opened for compiling and debugging. If failed, please check the install path of JLink, ARM gcc and OpenOCD. If their path are different, please refer to section 5 and 6 for modification.

This section covers the following contents:

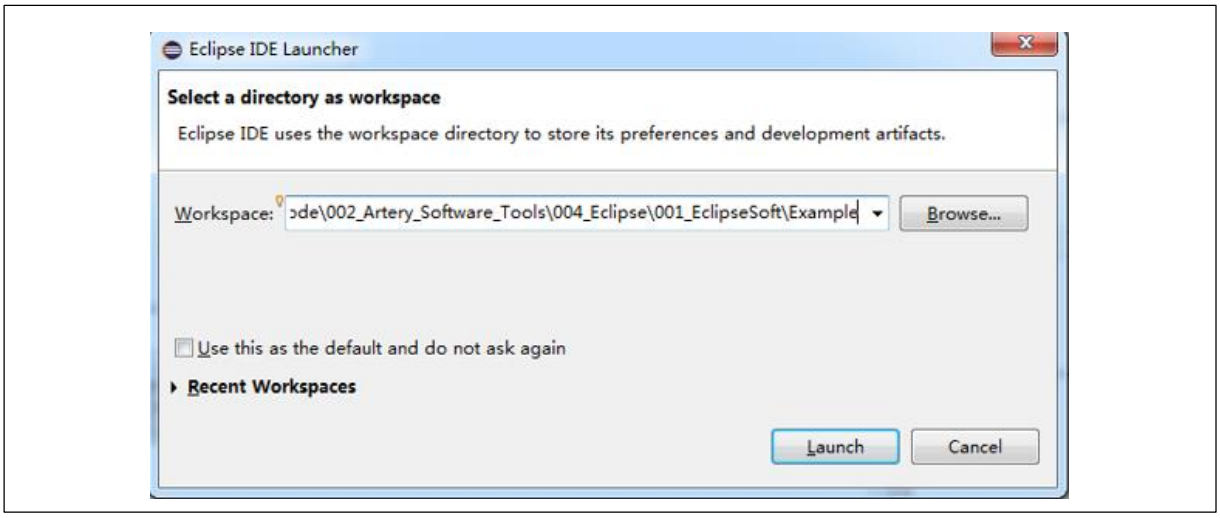
- Import Example routine
- AT32F403_Example compile and debug
- AT32F413_Example compile and debug
- AT32F415_Example compile and debug

7.1 Import Example routine

Example routine is located under *AT32_Eclipse_Source*, and can be opened for use through Eclipse.

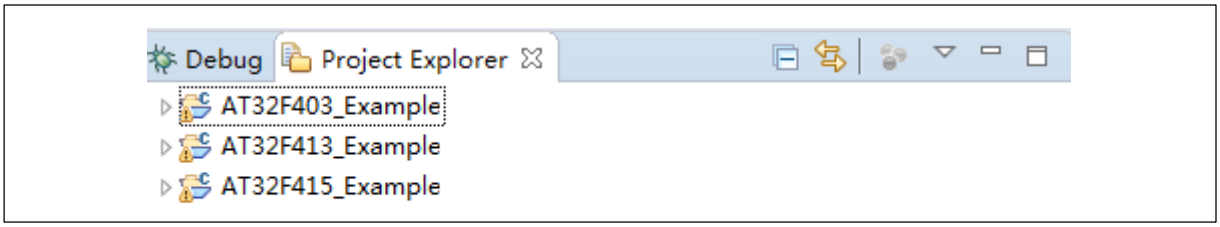
1. Open *eclipse.exe*, select Example directory as Workspace.

Figure 83. Set Workspace



2. After opening the *eclipse.exe, you can see three files:* AT32F403_Example, AT32F413_Example and AT32F415_Example.

Figure 84. Project Explorer

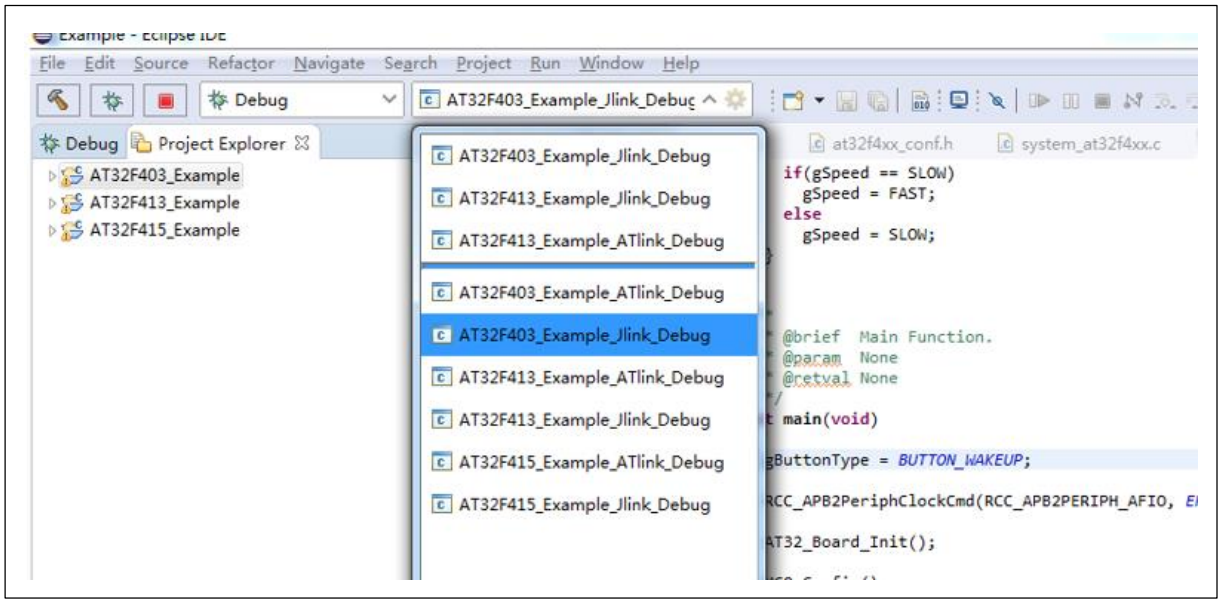


7.2 AT32F403_Example compile and debug

In AT32F403_Example, the chip we used is AT32F403ZGT6, and evaluation board AT-START-F403-V1.2.

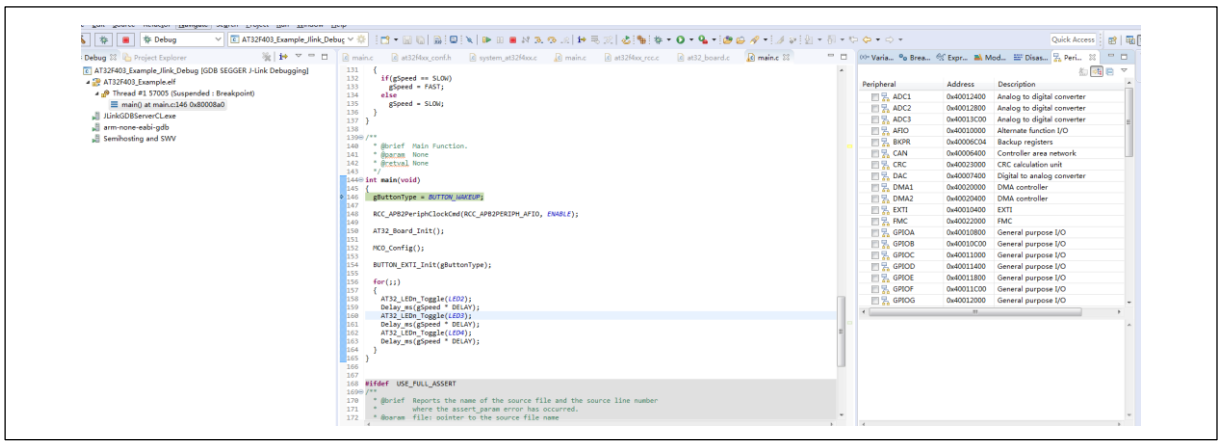
1. JLink debug (***Debug->AT32F403_Example_Jlink_Debug***)

Figure 85. JLink debug



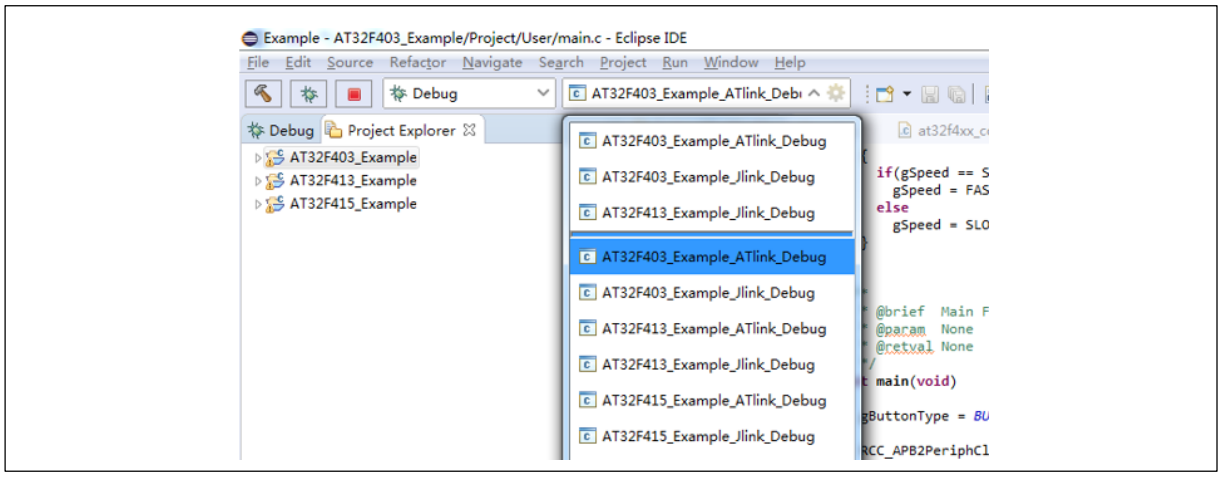
Click ***Debug,*** the project will automatically compile, download and enter Debug mode.

Figure 86. Enter Debug mode



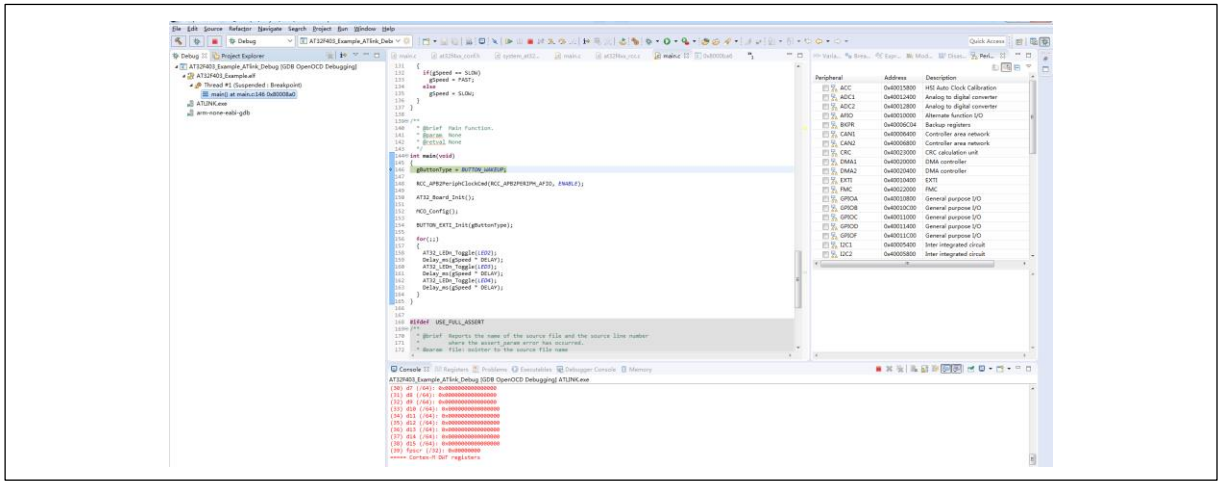
2. ATLink debug (***Debug->AT32F403_Example_ATlink_Debug***)

Figure 87. ATLink debug



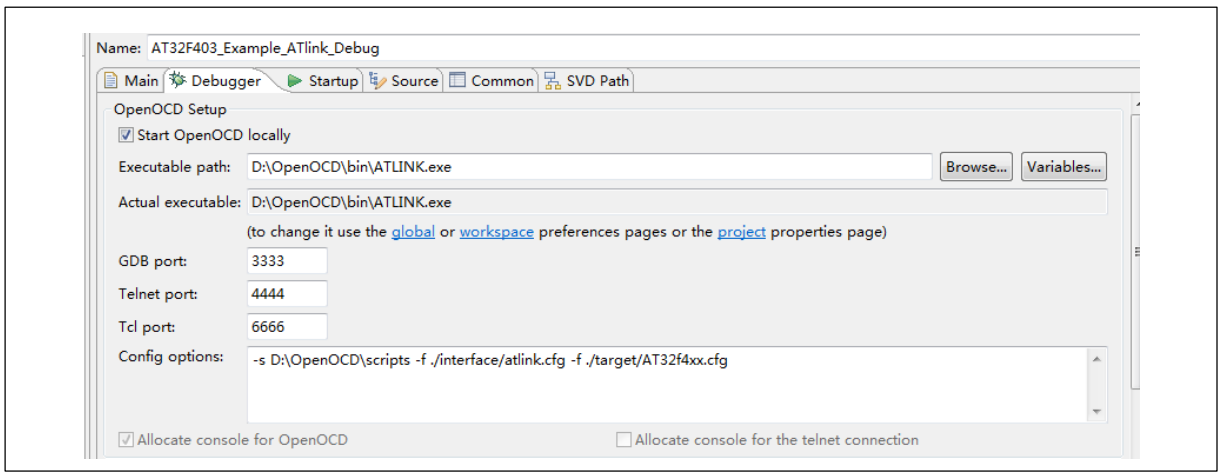
Click ***Debug***, the project will automatically compile, download and enter Debug mode.

Figure 88. Enter Debug mode



Note: ATlink default configuration is as follows. The ATLink.exe path must be the same, otherwise, follow section 6 for modification.

Figure 89. ATlink default configuration

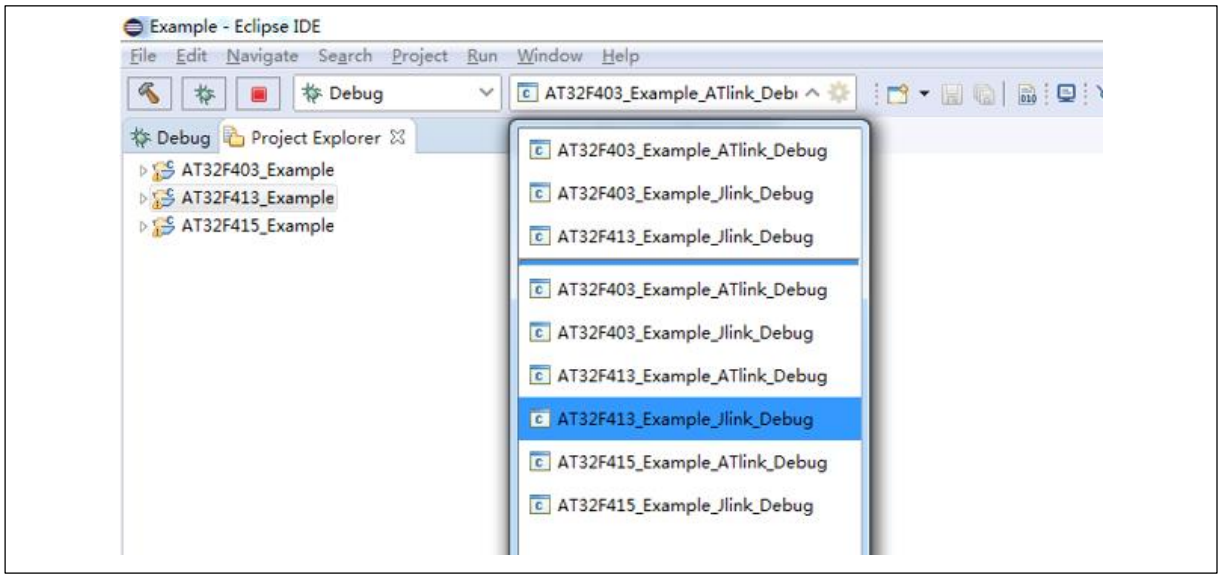


7.3 AT32F413_Example compile and debug

In AT32F413_Example, the chip and evaluation board we use are AT32F413RCT7 and AT-START-F413-V1.0 respectively.

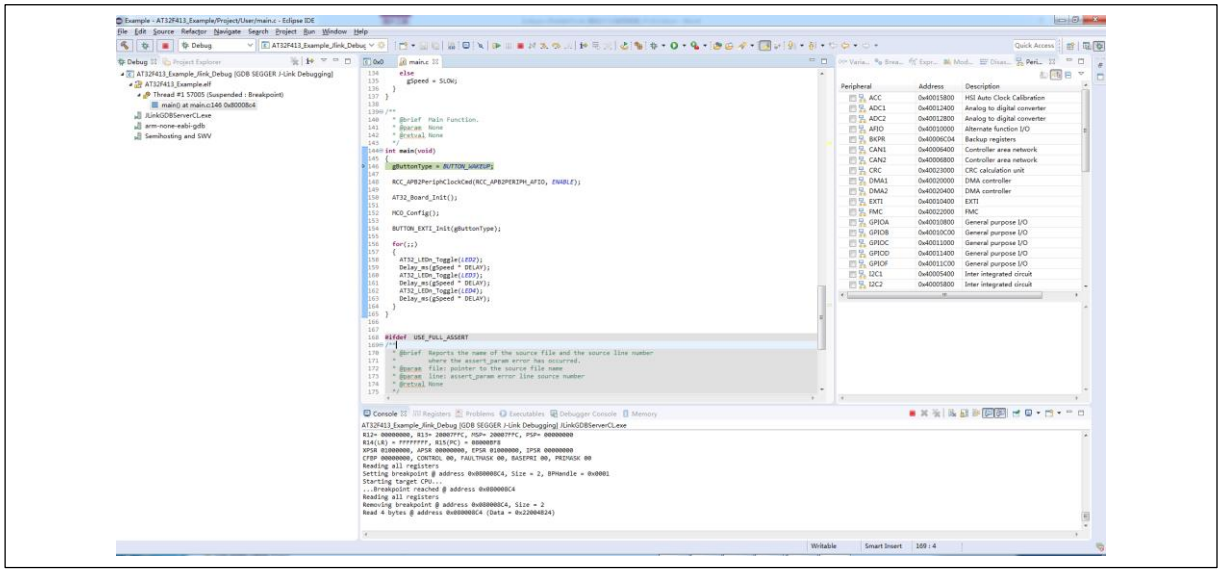
1. JLink debug (***Debug->AT32F413_Example_Jlink_Debug***)

Figure 90. JLink debug



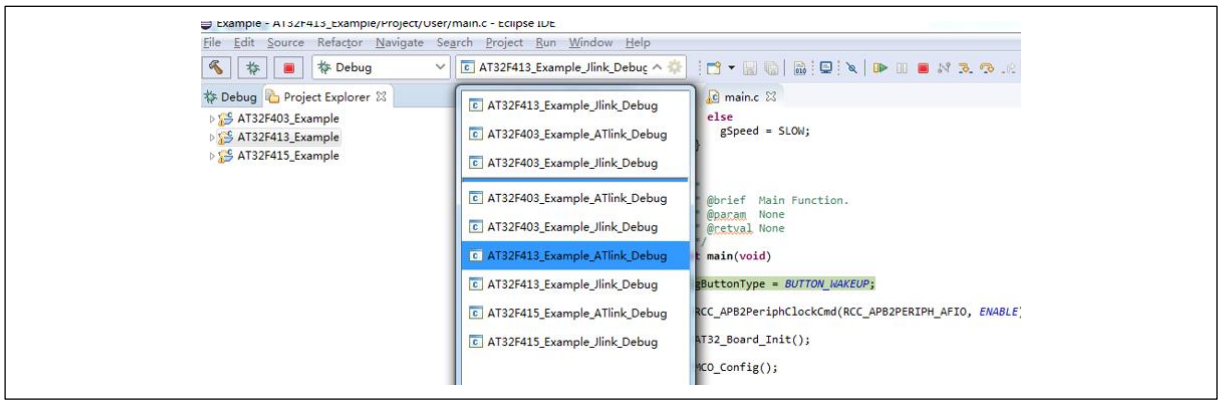
Click ***Debug***, it will automatically compile, download and enter Debug mode.

Figure 91. Enter Debug mode



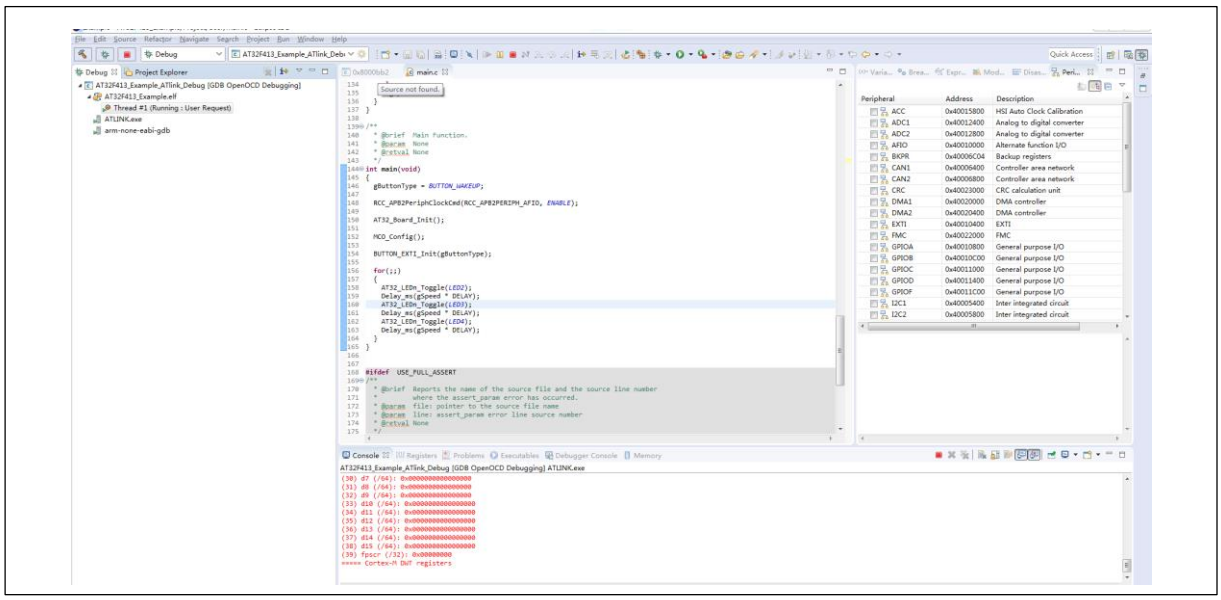
2. ATLink debug (***Debug->AT32F413_Example_ATlink_Debug***)

Figure 92. ATLink debug



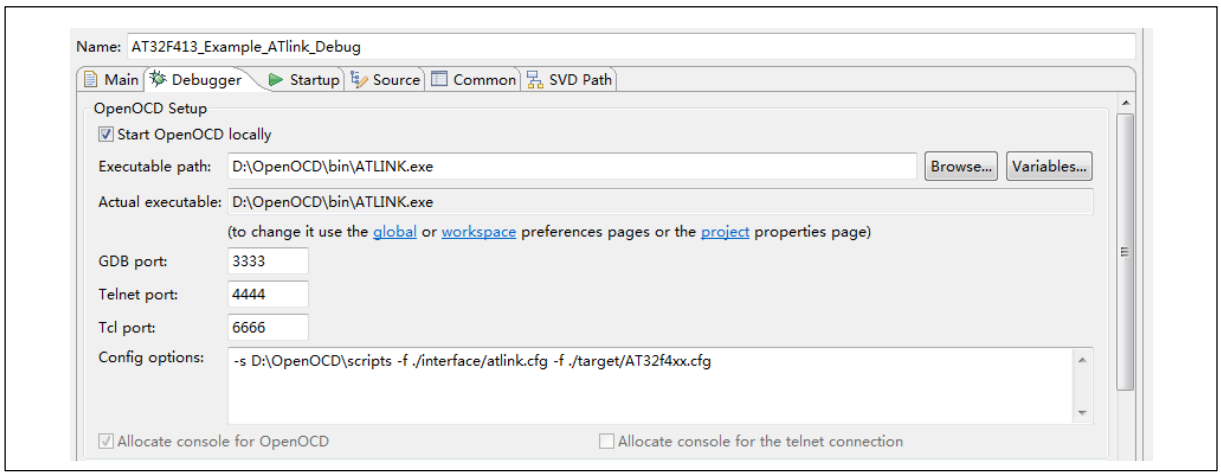
Click ***Debug***, it will automatically compile, download and enter Debug mode.

Figure 93. Enter Debug mode



Note: ATlink default configuration is as follows. The ATLink.exe path must be the same, otherwise, follow section 6 for modification.

Figure 94. ATlink default configuration

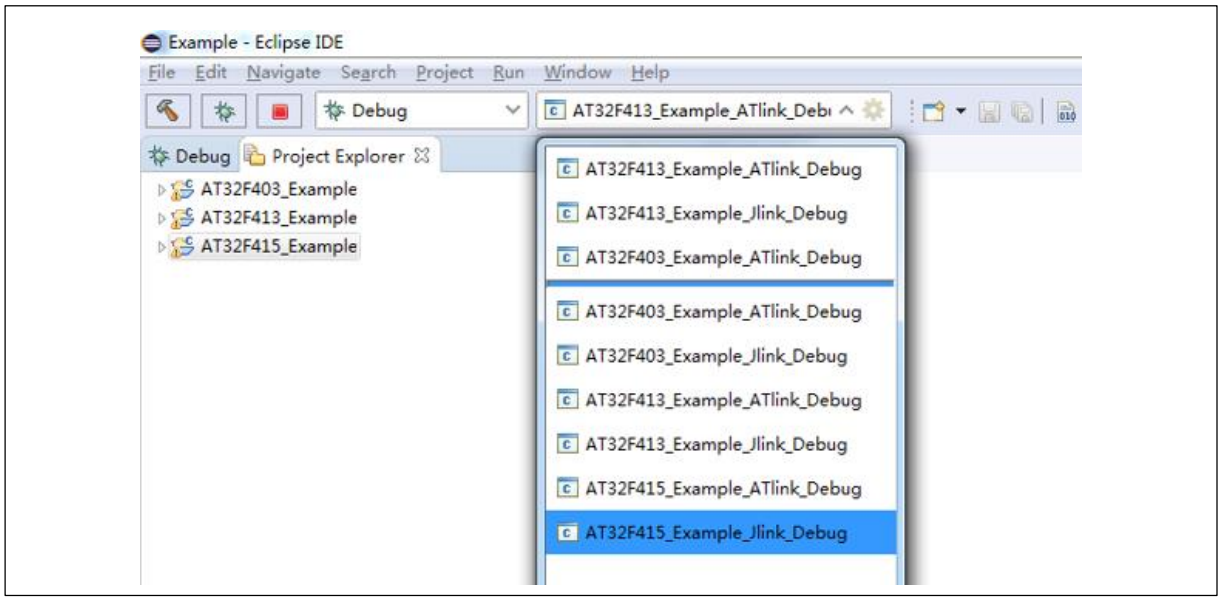


7.4 AT32F415_Example compile and debug

In AT32F415_Example, the chip we use is AT32F415RCT7, and the board AT-START-F415-V1.0.

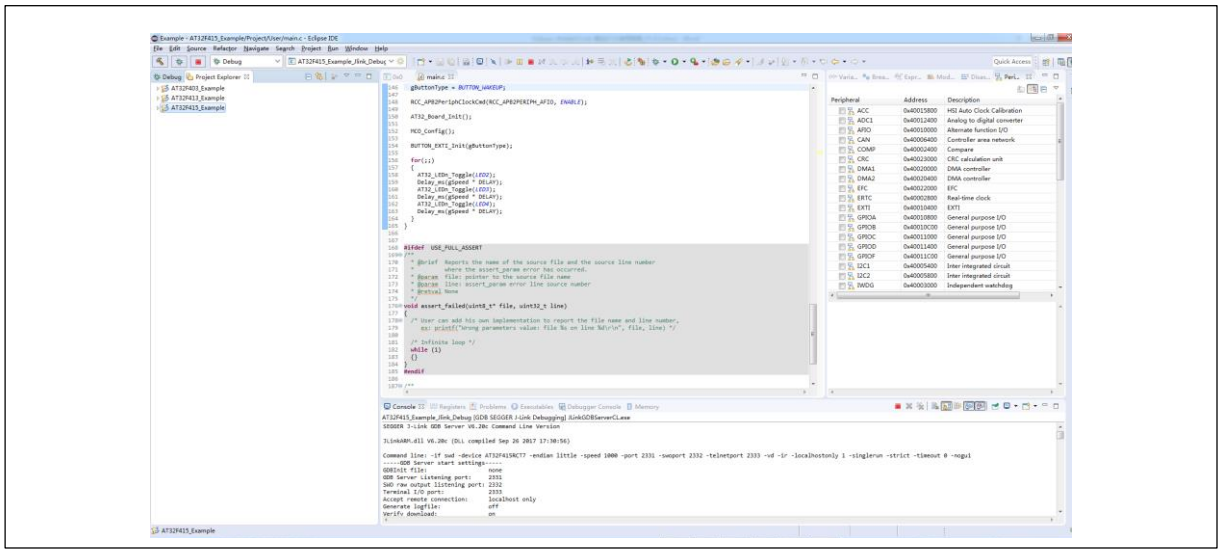
1. JLink debug (***Debug->AT32F415_Example_Jlink_Debug***)

Figure 95. JLink debug



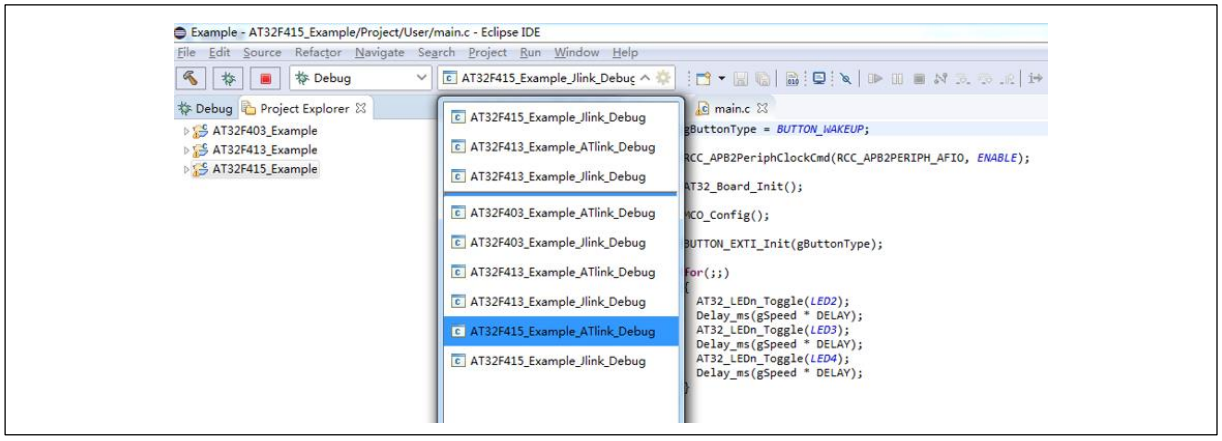
Click ***Debug***, the project will automatically compile, download and enter Debug mode.

Figure 96. Enter Debug mode



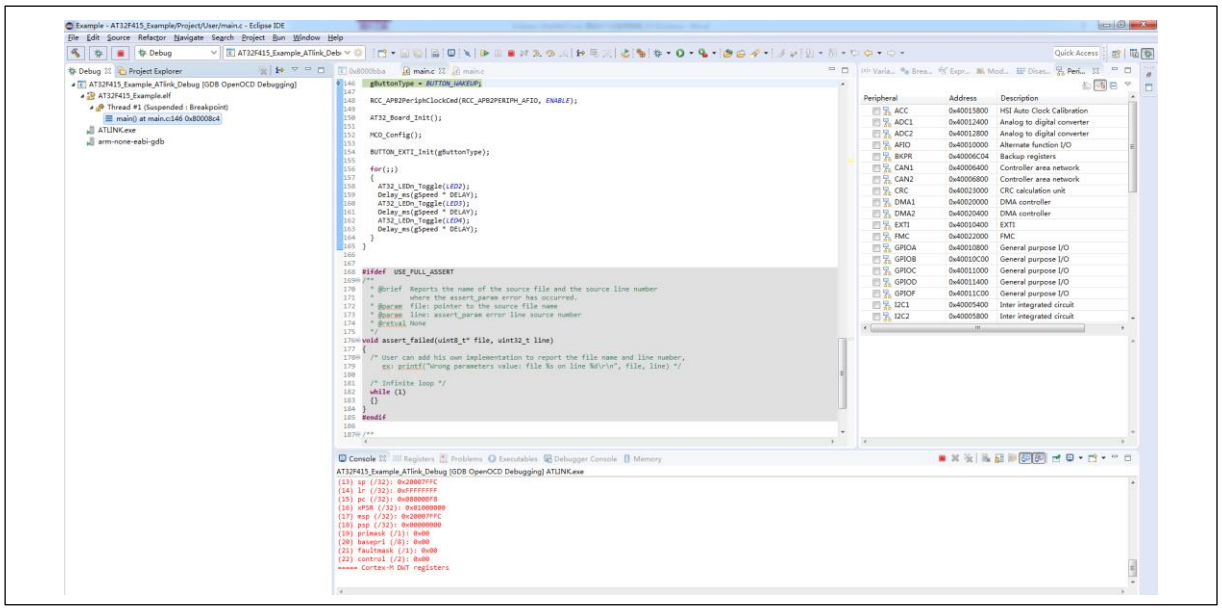
3. ATLink debug (***Debug->AT32F415_Example_ATlink_Debug***)

Figure 97. ATLink debug



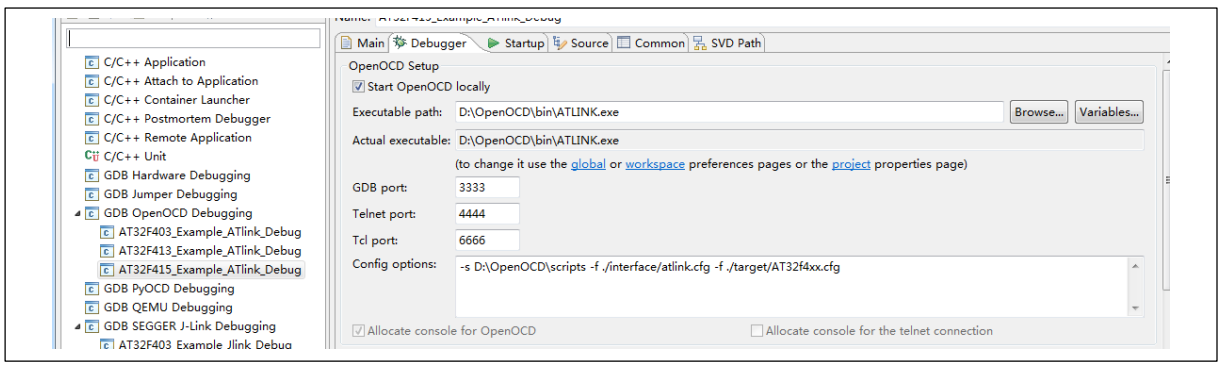
Click ***Debug***, it will automatically compile, download and enter Debug mode.

Figure 98. Enter Debug mode



Note: ATlink default configuration is as follows. The ATLink.exe path must be the same, otherwise, follow section 6 for modification.

Figure 99. ATlink default configuration

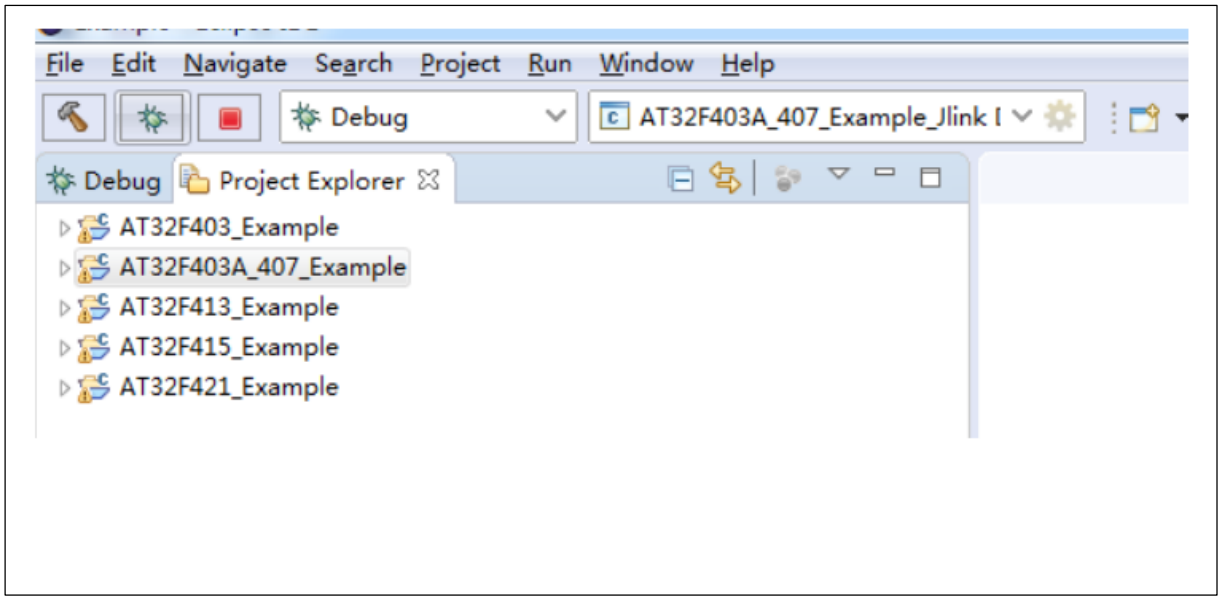


7.5 AT32F403A/407_Example compile and debug

In AT32F403A/407_Example, the chip we use is AT32F403AVGT7, and the board AT-START-F403A-V1.0.

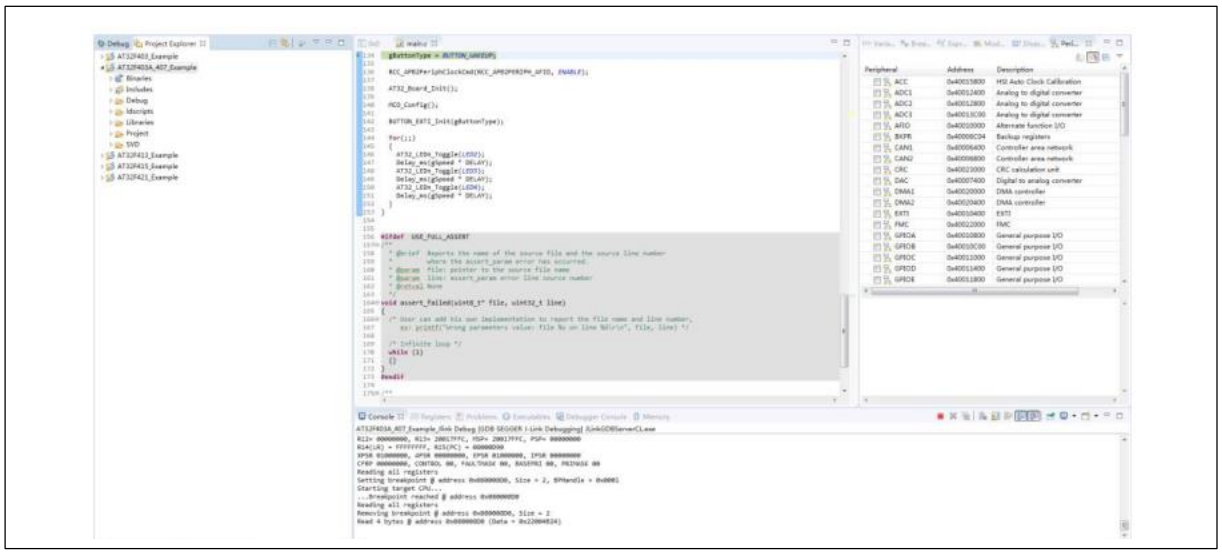
2. JLink debug (***Debug->AT32F403A_407_Example_Jlink_Debug***)

Figure 100. JLink debug



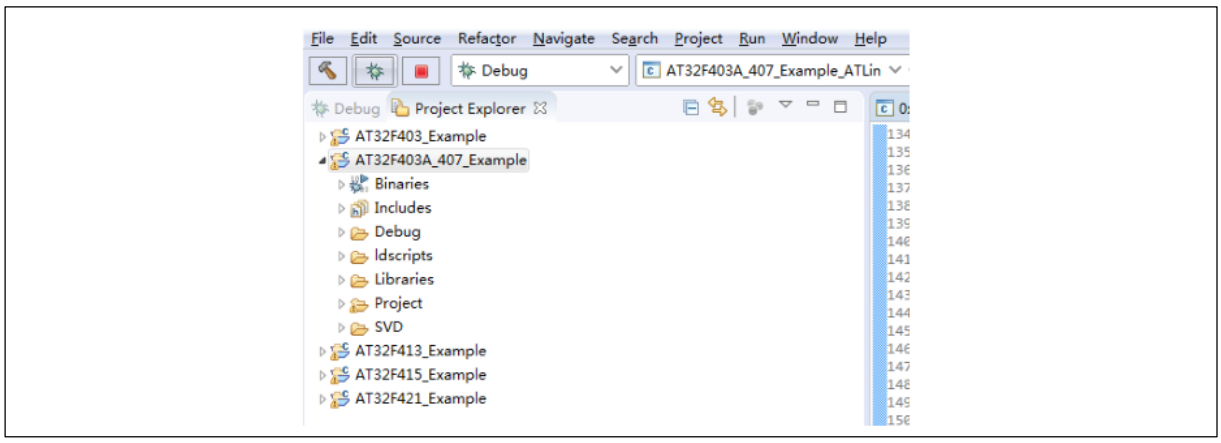
Click ***Debug***, the project will automatically compile, download and enter Debug mode.

Figure 101. Enter Debug mode



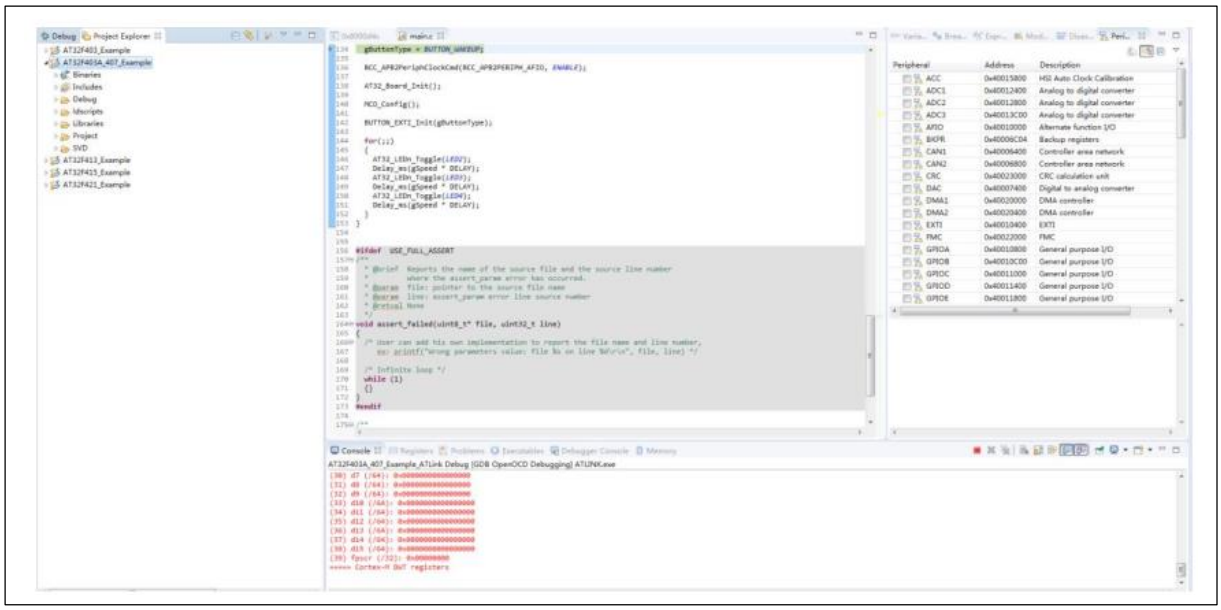
4. ATLink debug (***Debug->AT32F403A_407_Example_ATlink_Debug***)

Figure 102. ATLink debug



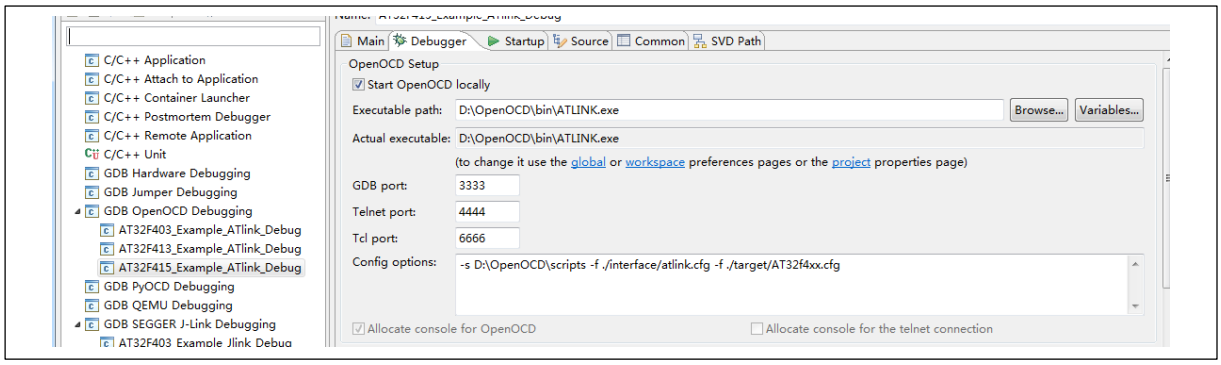
Click ***Debug***, it will automatically compile, download and enter Debug mode.

Figure 103. Enter Debug mode



Note: ATlink default configuration is as follows. The ATLink.exe path must be the same, otherwise, follow section 6 for modification.

Figure 104. ATlink default configuration

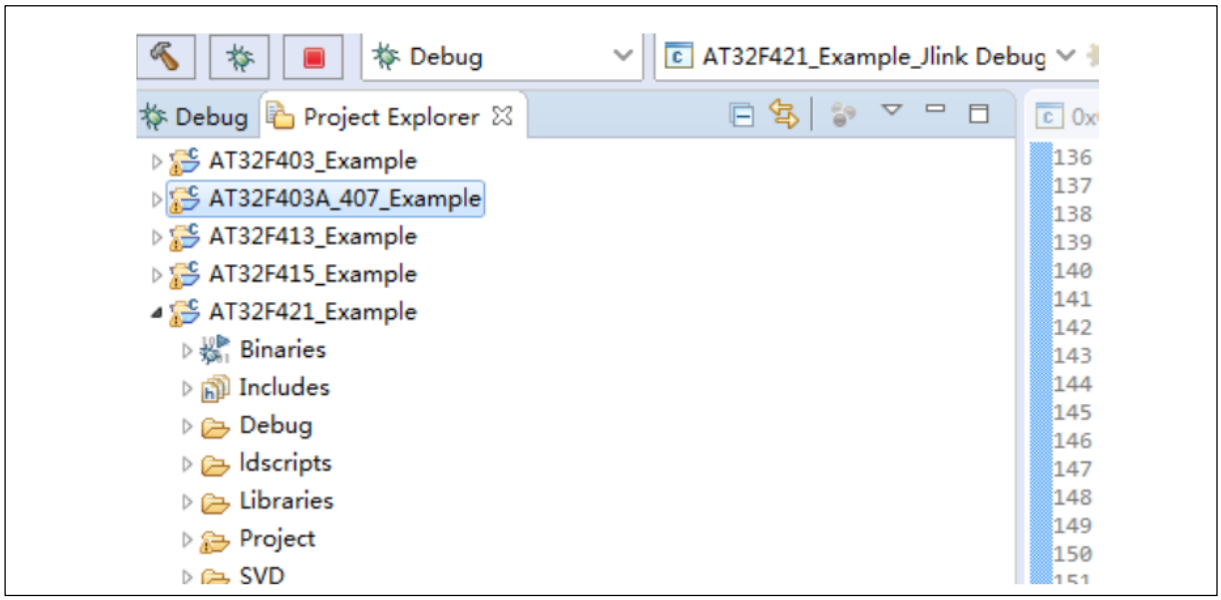


7.6 AT32F421_Example compile and debug

In AT32F421_Example, the chip we use is AT32F421RCT7, and the board AT-START-F421-V1.0.

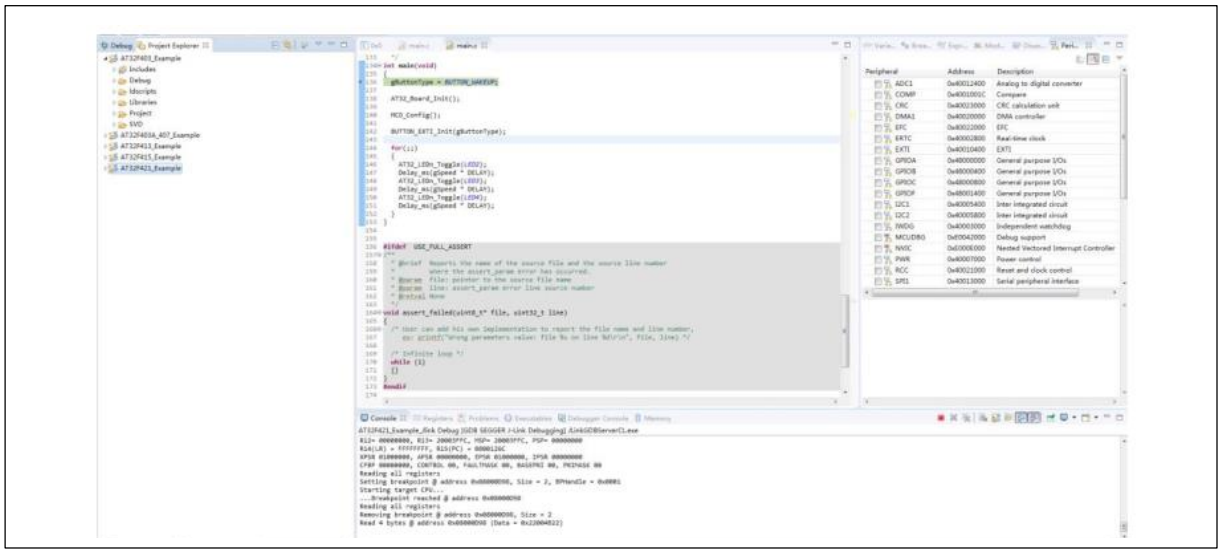
3. JLink debug (***Debug->AT32F421_Example_Jlink_Debug***)

Figure 105. JLink debug



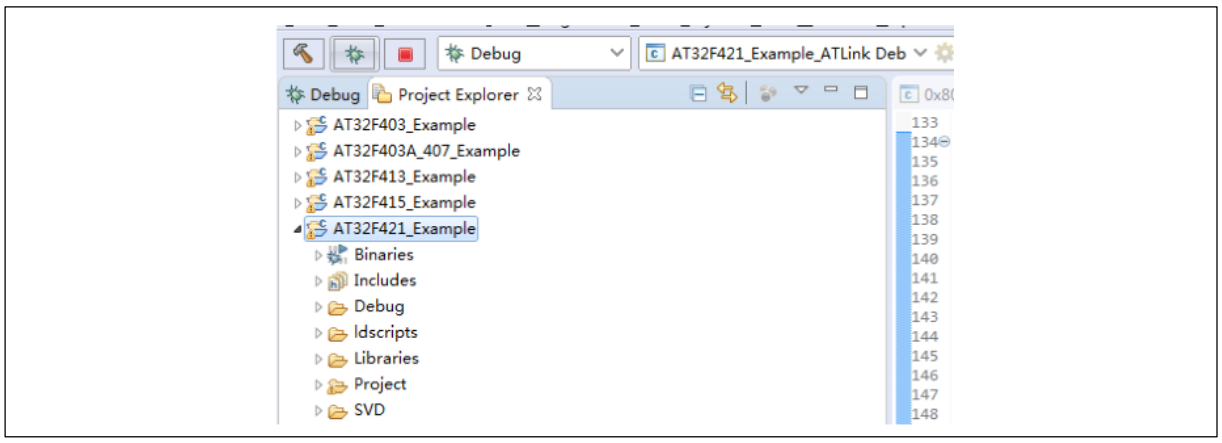
Click ***Debug***, the project will automatically compile, download and enter Debug mode.

Figure 106. Enter Debug mode



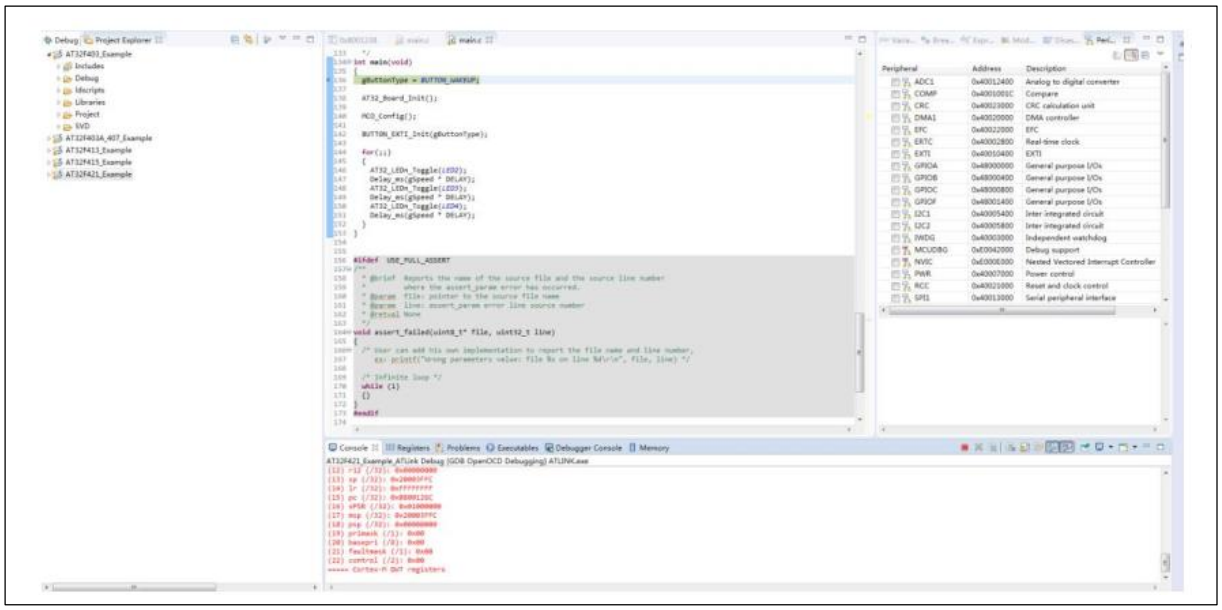
5. ATLink debug (***Debug->AT32F421_Example_ATlink_Debug***)

Figure 107. ATLink debug



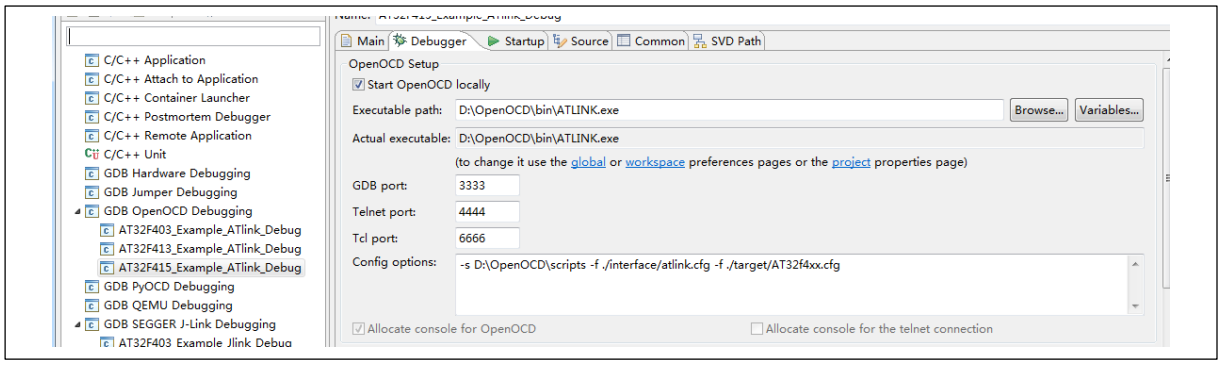
Click ***Debug***, it will automatically compile, download and enter Debug mode.

Figure 108. Enter Debug mode



Note: ATlink default configuration is as follows. The ATLink.exe path must be the same, otherwise, follow section 6 for modification.

Figure 109. ATlink default configuration



8 Revision history

Table 1. Document revision history

Date	Revision	Changes
2019.08.13	1.0.0	Initial release.
2019.08.16	1.0.1	1. Added the description of GNU ARM Eclipse Build Tools 2. Modified the description of adding script file “AT32F403xG_FLASH.ld” in new project.
2019.09.05	1.0.2	1. Added a note for not using Chinese path when creating new projects. 2. Added OpenOCD.zip in the AT32_Eclipse_Packet 3. Added .S and .ld files of more products. 4. Added the description of Eclipse+ATLink debug
2019.09.20	1.0.3	1. Modified the description of section 1. 2. Modified the section 3. 3. Modified Section 4, and added “AT32F403 configuration and compile”, “AT32F413 configuration and compile” and “AT32F415 configuration and compile”. 4. Added Section 7.
2020.09.07	1.0.4	1. Added AT32F403A_407 Example. 2. Added AT32F421 Example.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers understand and agree that purchasers are solely responsible for the selection and use of Artery's products and services.

Artery's products and services are provided "AS IS" and Artery provides no warranties express, implied or statutory, including, without limitation, any implied warranties of merchantability, satisfactory quality, non-infringement, or fitness for a particular purpose with respect to the Artery's products and services.

Notwithstanding anything to the contrary, purchasers acquires no right, title or interest in any Artery's products and services or any intellectual property rights embodied therein. In no event shall Artery's products and services provided be construed as (a) granting purchasers, expressly or by implication, estoppel or otherwise, a license to use third party's products and services; or (b) licensing the third parties' intellectual property rights; or (c) warranting the third party's products and services and its intellectual property rights.

Purchasers hereby agrees that Artery's products are not authorized for use as, and purchasers shall not integrate, promote, sell or otherwise transfer any Artery's product to any customer or end user for use as critical components in (a) any medical, life saving or life support device or system, or (b) any safety device or system in any automotive application and mechanism (including but not limited to automotive brake or airbag systems), or (c) any nuclear facilities, or (d) any air traffic control device, application or system, or (e) any weapons device, application or system, or (f) any other device, application or system where it is reasonably foreseeable that failure of the Artery's products as used in such device, application or system would lead to death, bodily injury or catastrophic property damage.