

Application note

Eclipse development environment based on Windows Application Note

1 Overview	4
2 Development tools.....	4
2.1 software	4
2.2 hardware	4
3 Development environment setup	5
3.1 Installing the ARM Cross Compiler Toolchain.....	5
3.2 Installing Build Tools	5
3.3 Installing Eclipse IDE.....	6
3.3.1 Installing JDK.....	6
3.3.2 Installation of Eclipse IDE for GNU ARM C/C++ Developers	6
3.4 Installing the JLink Tool	7
3.5 Adding Chip Support	7
3.6 JLink download test.....	7
4 Eclipse Launch and Configuration.....	9
4.1 4.1 Create a New Workspace	9
4.2 Set the Path of Build Tools	10
4.3 Set the Cross-Compilation Toolchain Path	11
4.4 Set the SEGGER J-Link Path	12
5 Project Creation and Configuration.....	13
5.1 Create a New Project	13
5.2 Folder Creation and File Import	15
5.3 Project Configuration	18
5.3.1 Chip Selection	18
5.3.2 Optimization Level Configuration.....	18
5.3.3 GNU Arm Cross C Compiler Macro Configuration	19
5.3.4 GNU Arm Cross C Linker Configuration.....	20
5.3.5 Configure Bin File Generation	21
6 Compilation	22
7 J-Link Download and Debugging.....	23
7.1 Main Tab.....	23
7.2 Debugger Tab	23
7.3 SVD Path Tab	24
7.4 Debug Interface	25
8 Import Existing Projects	27
9 Version history	28
10 Notice	29

1 Overview

Taking N32H760 series MCU as an example, this paper introduces the methods of setting up development environment, It is applicable to all N32 MCU series..

2 Development tools

2.1 software

- 1) Operating System: WIN7 / WIN10 64-bit OS
- 2) IDE: Eclipse IDE for GNU ARM & RISC-V C/C++ Developers
- 3) Cross Compiler Toolchain: arm-none-eabi-gcc
- 4) Build Tools: GNU MCU Eclipse build tools
- 5) GDB Server: J-Link GDB Server
- 6) Debugger: J-Link V9/V10

2.2 hardware

- 1) Development board N32H760xIx7-STB V1.0
- 2) JLink Downloader V9 (need to be no lower than the software support version) or above

3 Development environment setup

3.1 Installing the ARM Cross Compiler Toolchain

➤ **Download the file `xpack-arm-none-eabi-gcc-10.2.1-1.1-win32-x64.zip`:**

Download Address: <https://github.com/xpack-dev-tools/arm-none-eabi-gcc-xpack/tags>

You can visit <https://github.com/xpack-dev-tools/arm-none-eabi-gcc-xpack/releases/> to select and download other different versions of the ARM Cross Compiler Toolchain.

3.2 Installing Build Tools

➤ **Download the file `xpack-windows-build-tools-4.2.1-2-win32-x64.zip`:**

Download Address: <https://github.com/xpack-dev-tools/windows-build-tools-xpack/releases/tag/v4.2.1-2/>

You can visit <https://xpack.github.io/windows-build-tools/releases/> to select and download different versions of the build tools.

3.3 Installing Eclipse IDE

3.3.1 Installing JDK

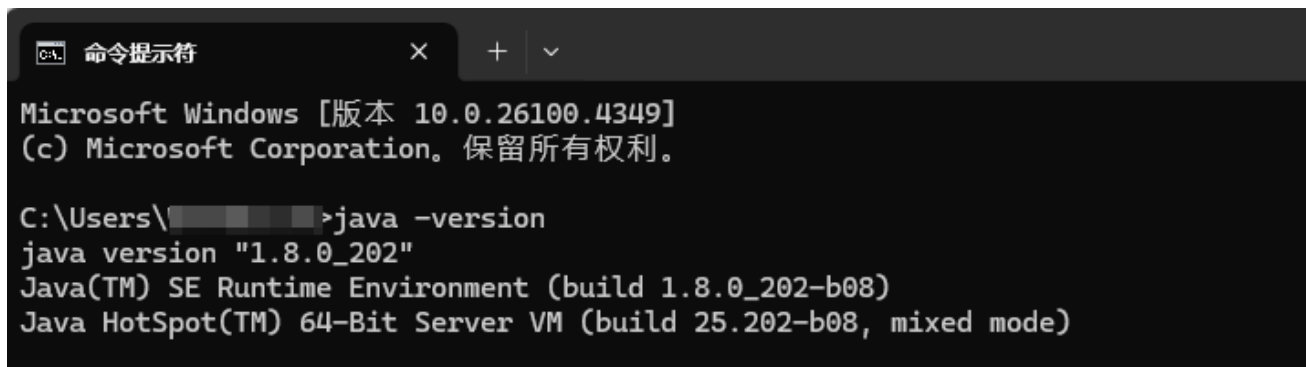
➤ **Download the file `jdk-8u202-windows-x64.exe`:**

Eclipse requires a Java environment to run, so JDK must be installed prior to installing Eclipse.

You can visit <http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html> to select and download different versions of the JDK tools.

In this document, `jdk-8u202-windows-x64.exe` is selected for download and installation. After the installation of the file `jdk-8u202-windows-x64.exe` is completed:

Verify the installation: Open a **DOS** command prompt window and type `java -version` to test if the JDK is installed correctly. If the JDK is installed properly, you will get output similar to the one shown in the figure.



```
Microsoft Windows [版本 10.0.26100.4349]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\>java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)
```

3.3.2 Installation of Eclipse IDE for GNU ARM C/C++ Developers

➤ **Download the file `eclipse-embedcpp-2021-03-R-win32-x86_64.zip`**

Download Address:

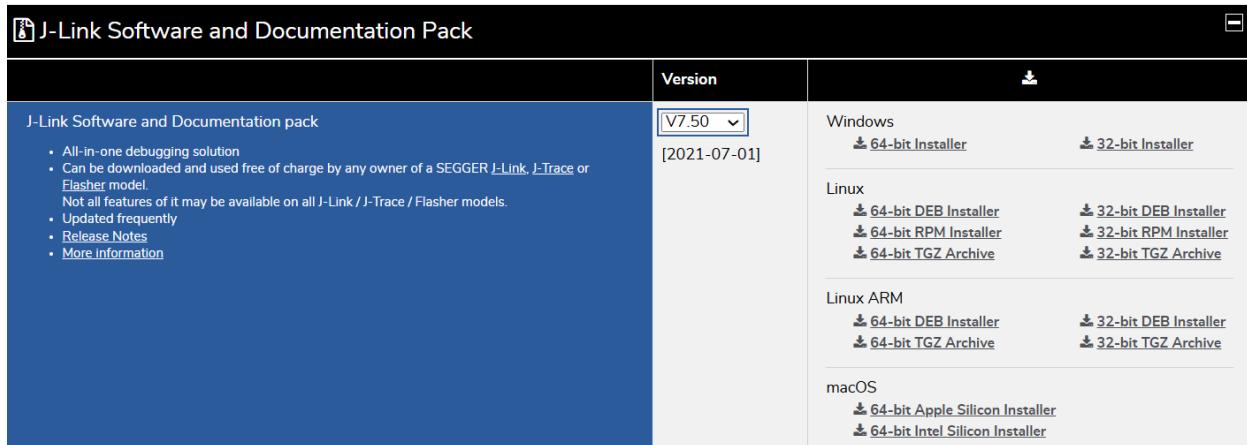
https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2021-03/R/eclipse-embedcpp-2021-03-R-win32-x86_64.zip

You can visit <https://eclipse-embed-cdt.github.io/packages/releases/> to select and download different versions of the Eclipse IDE.

3.4 Installing the JLink Tool

- **Download the JLINK installation package, V7.50 or others version**

Download Address: <https://www.segger.com/downloads/jlink/#-LinkSoftwareAndDocumentationPack>



	Version	Download
J-Link Software and Documentation pack <ul style="list-style-type: none"> All-in-one debugging solution Can be downloaded and used free of charge by any owner of a SEGGER J-Link, J-Trace or Flasher model. Not all features of it may be available on all J-Link / J-Trace / Flasher models. Updated frequently Release Notes More information 	V7.50 [2021-07-01]	Windows <ul style="list-style-type: none"> 64-bit Installer 32-bit Installer Linux <ul style="list-style-type: none"> 64-bit DEB Installer 32-bit DEB Installer 64-bit RPM Installer 32-bit RPM Installer 64-bit TGZ Archive 32-bit TGZ Archive Linux ARM <ul style="list-style-type: none"> 64-bit DEB Installer 32-bit DEB Installer 64-bit TGZ Archive 32-bit TGZ Archive macOS <ul style="list-style-type: none"> 64-bit Apple Silicon Installer 64-bit Intel Silicon Installer

3.5 Adding Chip Support

After installing JLink, we need to add our company's chip patch package to JLink, so that we can get the download algorithm correctly during downloading and debugging.

For details, see <[jlink Tool Adding Nations Chip.7z](#)>.

3.6 JLink download test

- **Test the JLink environment installation**

- 1, Connect the PC and j-Link debugger, connect the development board, and power on;
- 2, Open cmd.exe command line tool, go to JLink installation directory [C:\Program Files \(x86\)\SEGGER\JLink_V640](#), type [jlink.exe](#).

```

SEGGER J-Link Commander V7.50 (Compiled Jul  1 2021 17:43:13)
DLL version V7.50, compiled Jul  1 2021 17:41:53

Connecting to J-Link via USB...O.K.
Firmware: J-Link V9 compiled May  7 2021 16:26:12
Hardware version: V9.40
S/N: 59417452
License(s): RDI, GDB, FlashDL, FlashBP, JFlash
VTref=3.291V

Type "connect" to establish a target connection, '?' for help
J-Link>

```

The image above shows that the PC successfully connected to the JLink debugger.

- Then according to the prompt input: “connect”, “N32H760XIX7”, “SWD”, “4000”, if the previous operation is successful, you will see the following output information, JLink download debugging environment can be used normally.

```

Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: N32H760XIX7
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>s
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "N32H760XIX7" selected.

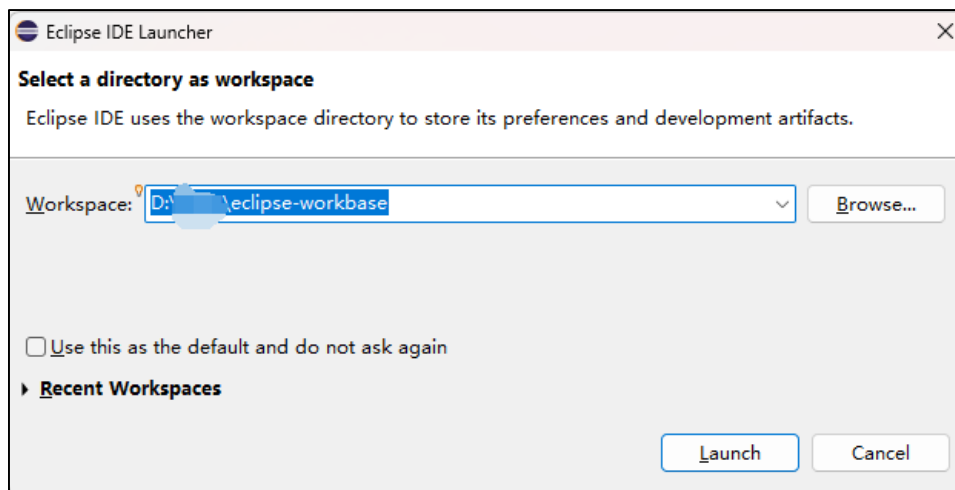
Connecting to target via SWD
InitTarget() start
*****
J-Link script: NationsTech Cortex-M7 J-Link script
*****
InitTarget() end
Found SW-DP with ID 0x2BA01477
DPIDR: 0x2BA01477
AP map detection skipped. Manually configured AP map found.
AP[0]: AHB-AP (IDR: Not set)
AP[1]: AHB-AP (IDR: Not set)
AP[2]: AHB-AP (IDR: Not set)
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x411FC272. Implementer code: 0x41 (ARM)
Found Cortex-M7 r1p2, Little endian.
FPUnit: 8 code (BP) slots and 0 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
ROMTbl[0][0]: E00E0000, CID: B105E00D, PID: 000BB00C SCS-M7
ROMTbl[0][1]: E0001000, CID: B105E00D, PID: 000BB002 DWT
ROMTbl[0][2]: E0002000, CID: B105E00D, PID: 000BB00E FPB-M7
ROMTbl[0][3]: E0000000, CID: B105E00D, PID: 000BB001 ITM
Cache: Separate I- and D-cache.
I-Cache L1: 32 KiB, 512 Sets, 32 Bytes/Line, 2-Way
D-Cache L1: 32 KiB, 256 Sets, 32 Bytes/Line, 4-Way
SetupTarget() start
SetupTarget() end
Cortex-M7 identified.
J-Link>

```


4 Eclipse Launch and Configuration

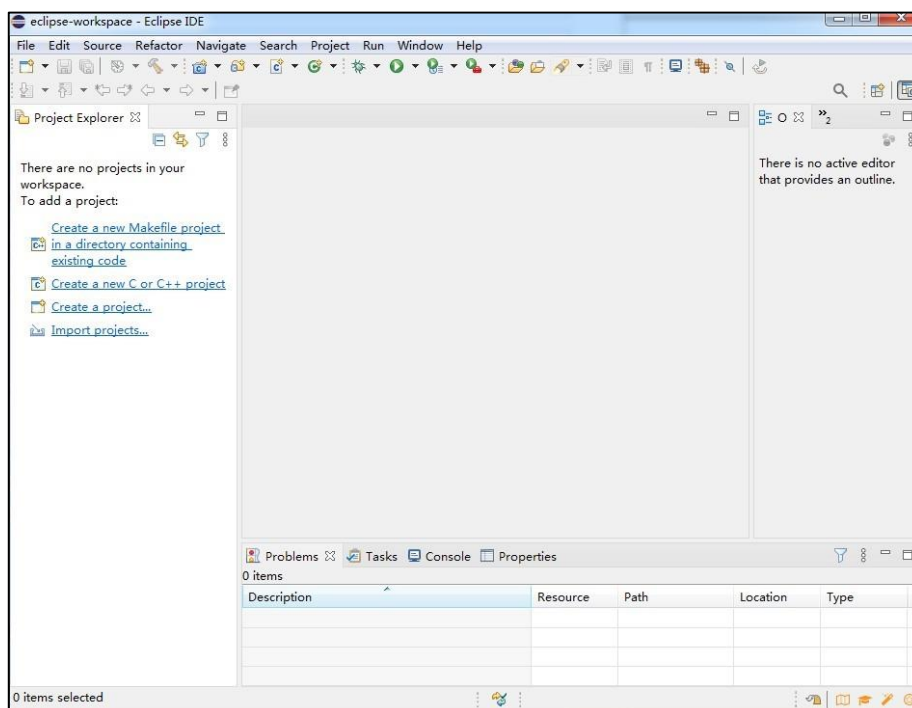
4.1 4.1 Create a New Workspace

Eclipse is a portable application that requires no installation. Simply double-click **eclipse.exe** in the Eclipse folder to launch the program. The interface after launch is shown in the figure below.



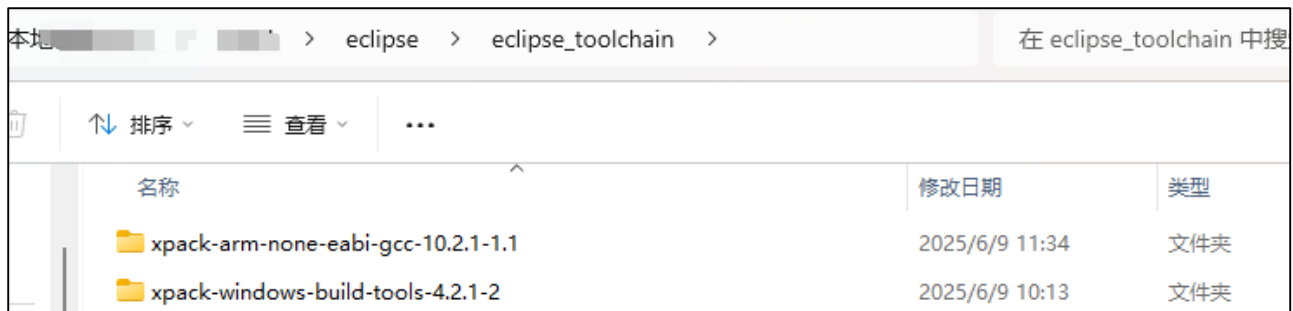
As shown in the figure below, select a local path without Chinese characters, create a workspace, and click **Launch**. Note that the path depth should not be too great.

After entering the welcome page, you can either close the **Welcome** tab in the upper left corner directly or click the **Workbench** icon in the upper right corner to enter the main interface.



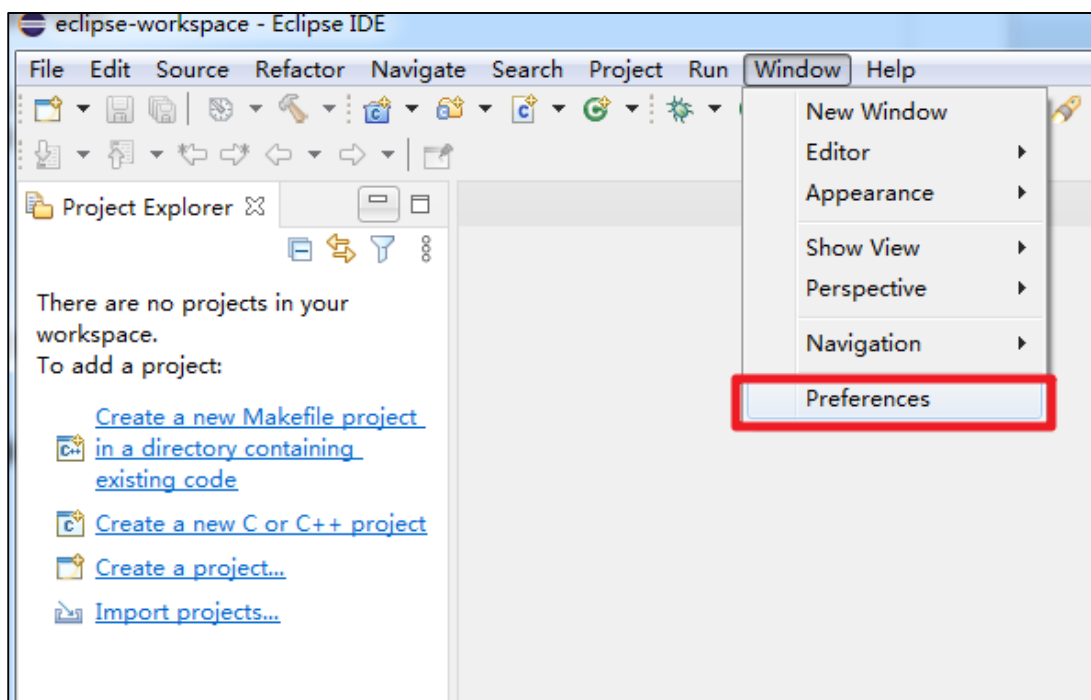
4.2 Set the Path of Build Tools

In the Eclipse installation directory, create a folder named **Eclipse_toolchain**. Decompress both the **ARM cross-compilation toolchain** and **Build tools** downloaded according to the tool installation instructions, and place them into this folder, as shown in the figure below.

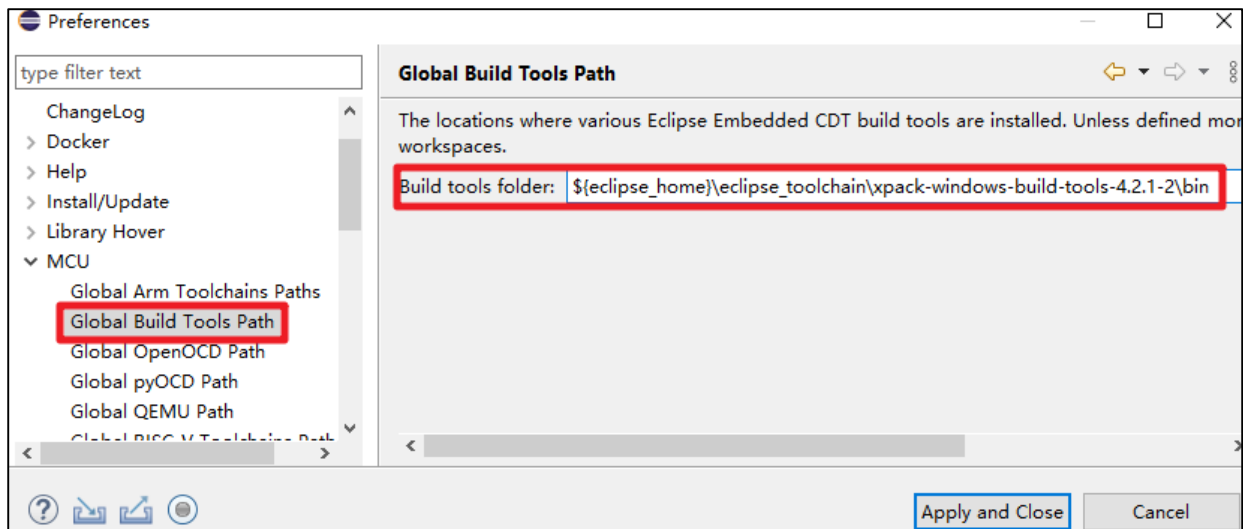


名称	修改日期	类型
xpack-arm-none-eabi-gcc-10.2.1-1.1	2025/6/9 11:34	文件夹
xpack-windows-build-tools-4.2.1-2	2025/6/9 10:13	文件夹

- 1) Open the **Window > Preferences** option



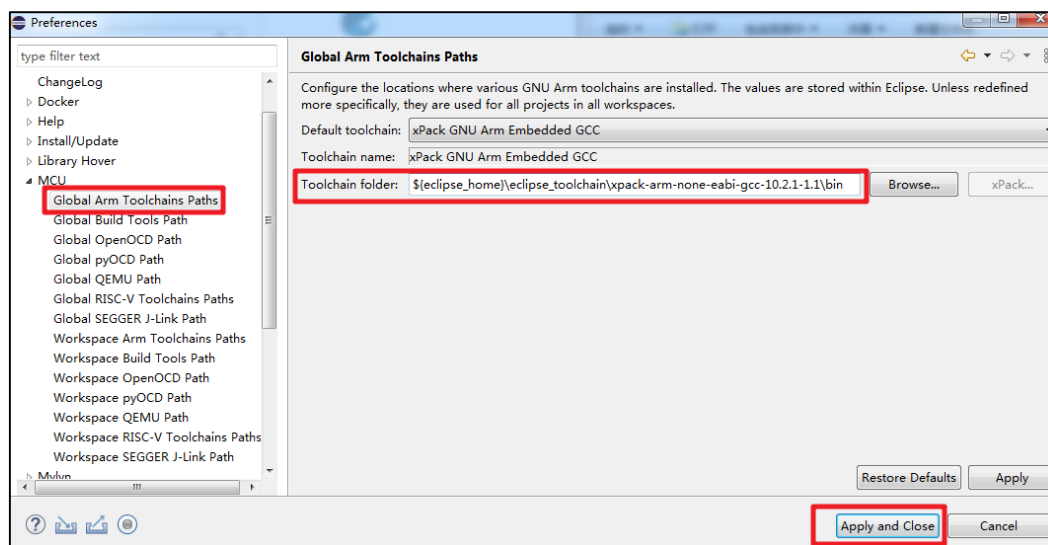
- 2) Select **MCU > Global Build Tools Path**, and set the global Build Tools path:
`${eclipse_home}\eclipse_toolchain\xpack-windows-build-tools-4.2.1-2\bin`. A relative path is required here for configuration.



4.3 Set the Cross-Compilation Toolchain Path

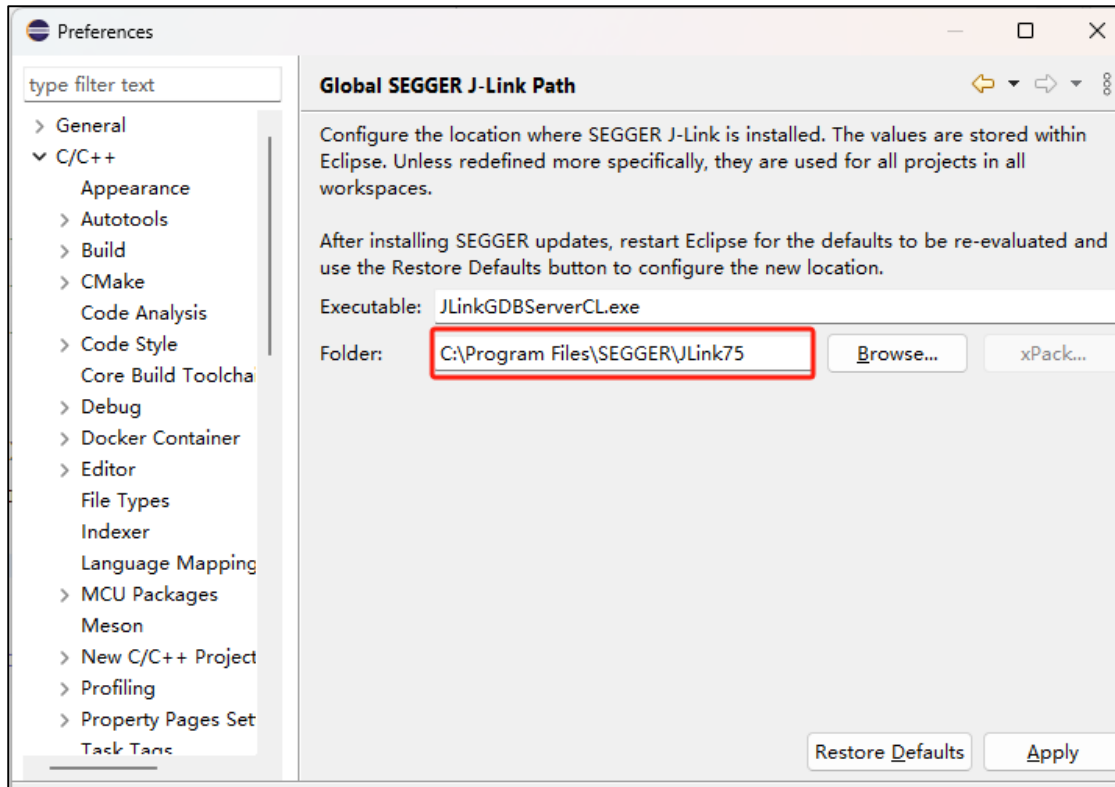
Select **MCU > Global Arm Toolchains Path**, and set the global Arm Toolchains path:

`${eclipse_home}\eclipse_toolchain\xpac-arm-none-eabi-gcc-10.2.1-1.1\bin`



4.4 Set the SEGGER J-Link Path

Select **MCU > Global SEGGER J-Link Path**, and set the global SEGGER J-Link tool path. Select a **local absolute path** here, and the path in this example is C:\Program Files\SEGGER\JLink75.



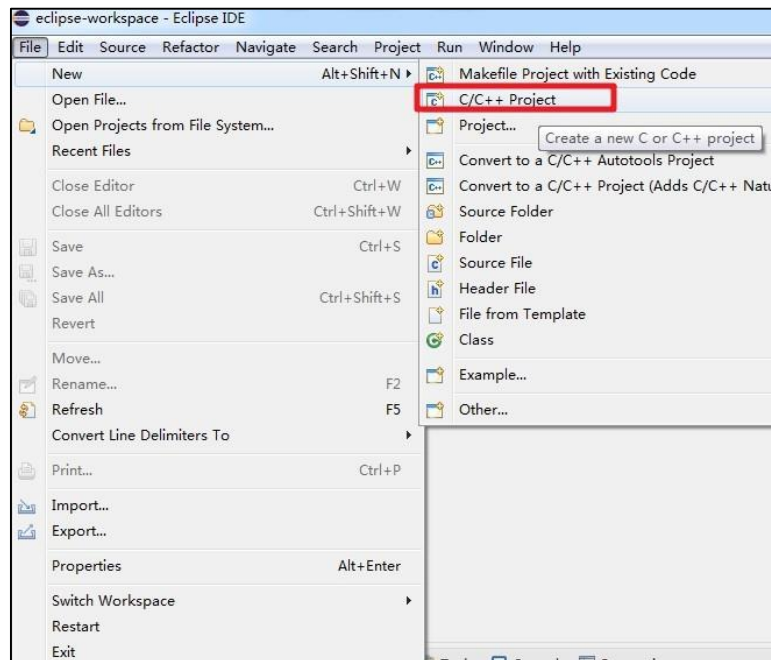
This completes all configurations for the Eclipse IDE.

5 Project Creation and Configuration

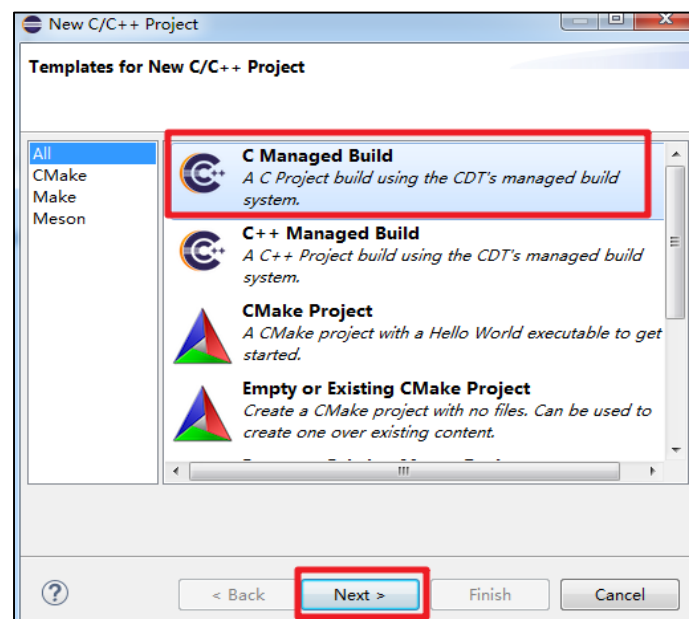
The SDK uses the released newest version, with **v1.0.0** being the current version. Based on this version, make the following modifications to adapt to the Eclipse development environment.

5.1 Create a New Project

- 1) Open Eclipse, navigate to **File > New**, select **C/C++ Project**, and then choose **C Managed Build**.

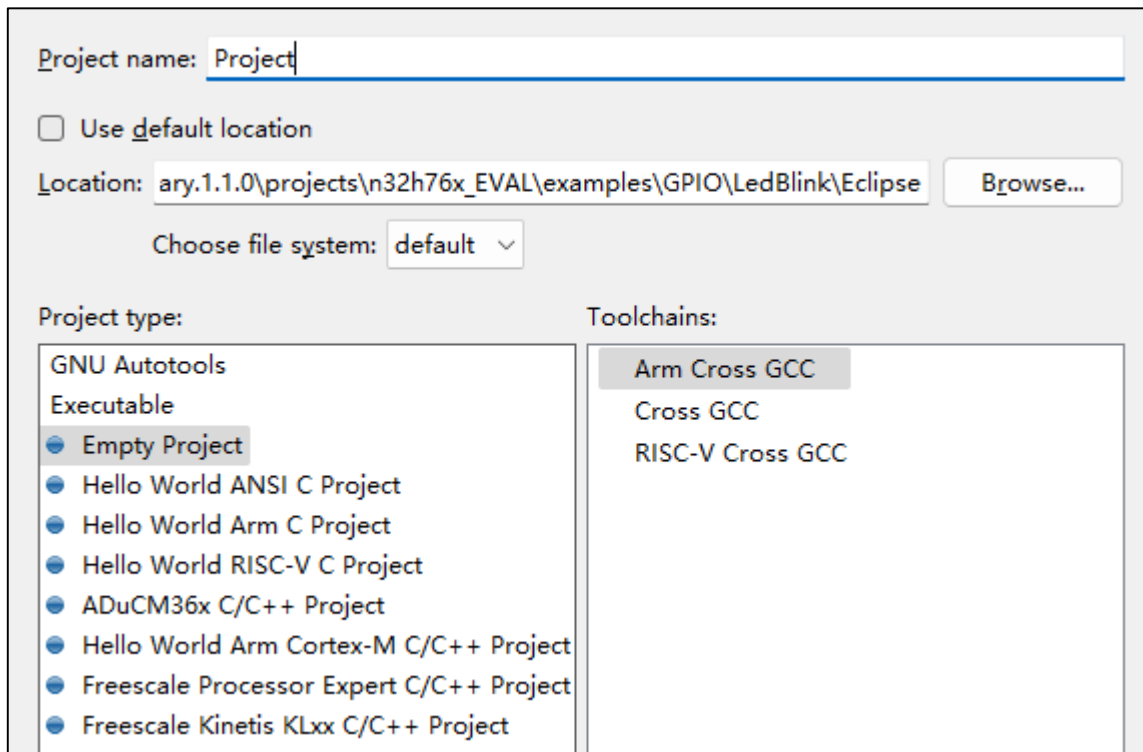


- 2) Select **C Managed Build**



- 3) Enter the Project name, configure the project type, and it is recommended to place the project in the Workspace

directory for convenience. Select **ARM Cross GCC** as the compilation toolchain.



Project name:

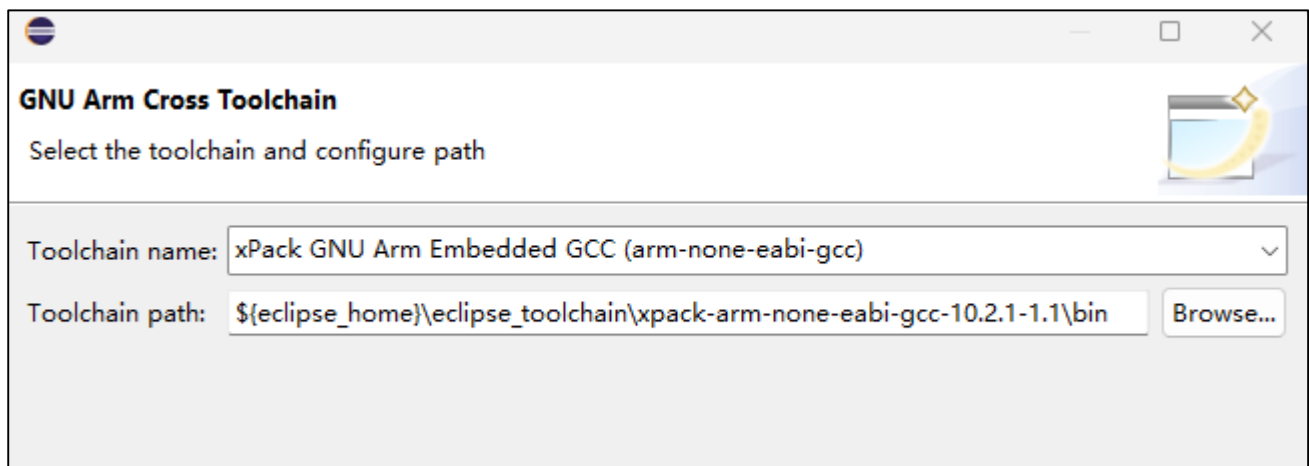
☐ Use default location

Location:

Choose file system:

Project type:	Toolchains:
GNU Autotools	Arm Cross GCC
Executable	Cross GCC
<input checked="" type="radio"/> Empty Project	RISC-V Cross GCC
<input type="radio"/> Hello World ANSI C Project	
<input type="radio"/> Hello World Arm C Project	
<input type="radio"/> Hello World RISC-V C Project	
<input type="radio"/> ADuCM36x C/C++ Project	
<input type="radio"/> Hello World Arm Cortex-M C/C++ Project	
<input type="radio"/> Freescale Processor Expert C/C++ Project	
<input type="radio"/> Freescale Kinetis KLxx C/C++ Project	

- 4) If the **ARM Toolchains Path** has been correctly configured in Eclipse IDE, the path will be selected automatically here. If the **ARM Toolchains Path** has not been configured in Eclipse IDE, you can also select the absolute path of the ARM Toolchains manually here. Then click **Finish** to complete the project creation.



GNU Arm Cross Toolchain

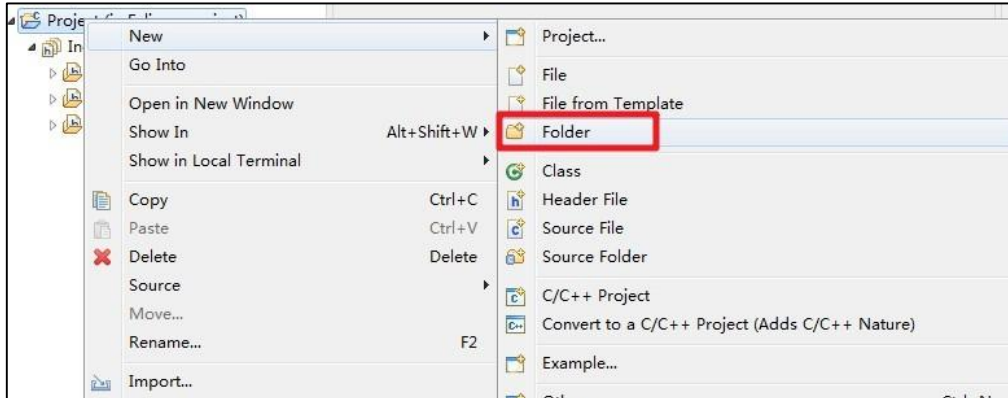
Select the toolchain and configure path

Toolchain name:

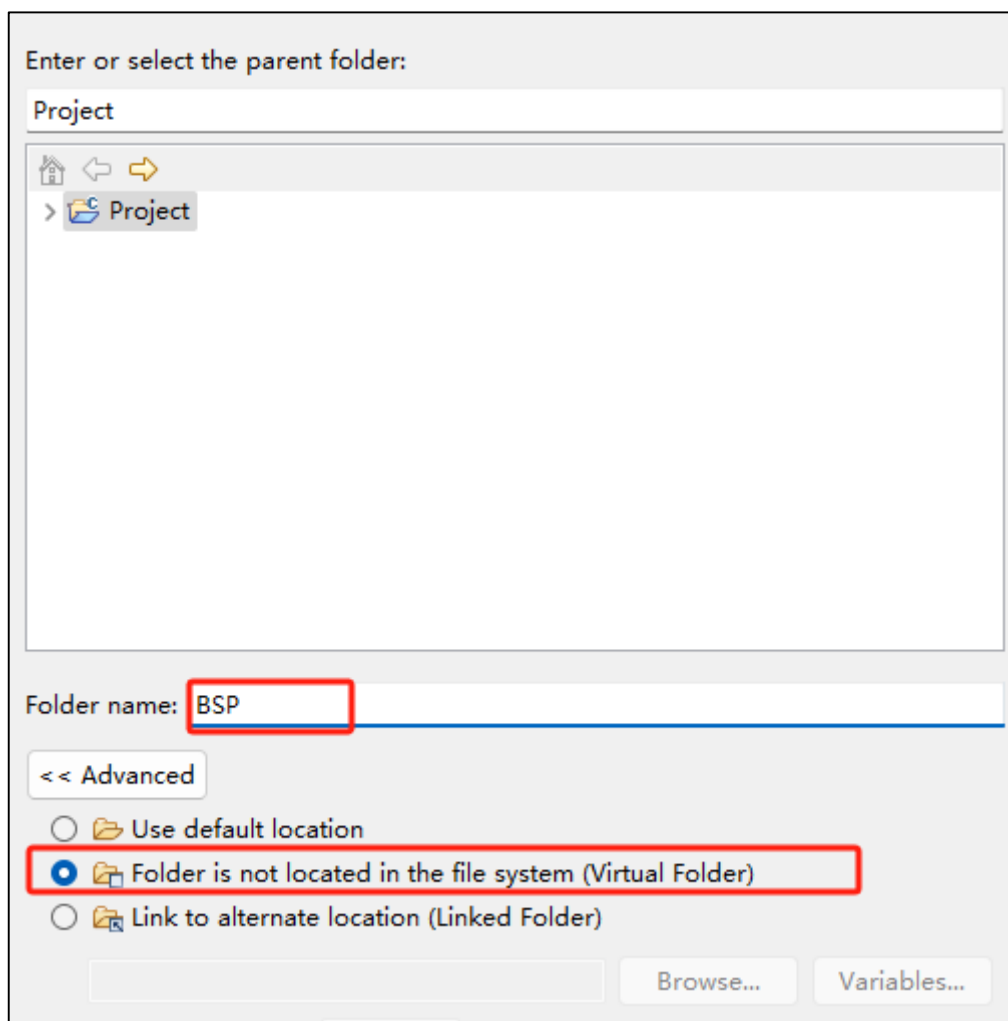
Toolchain path:

5.2 Folder Creation and File Import

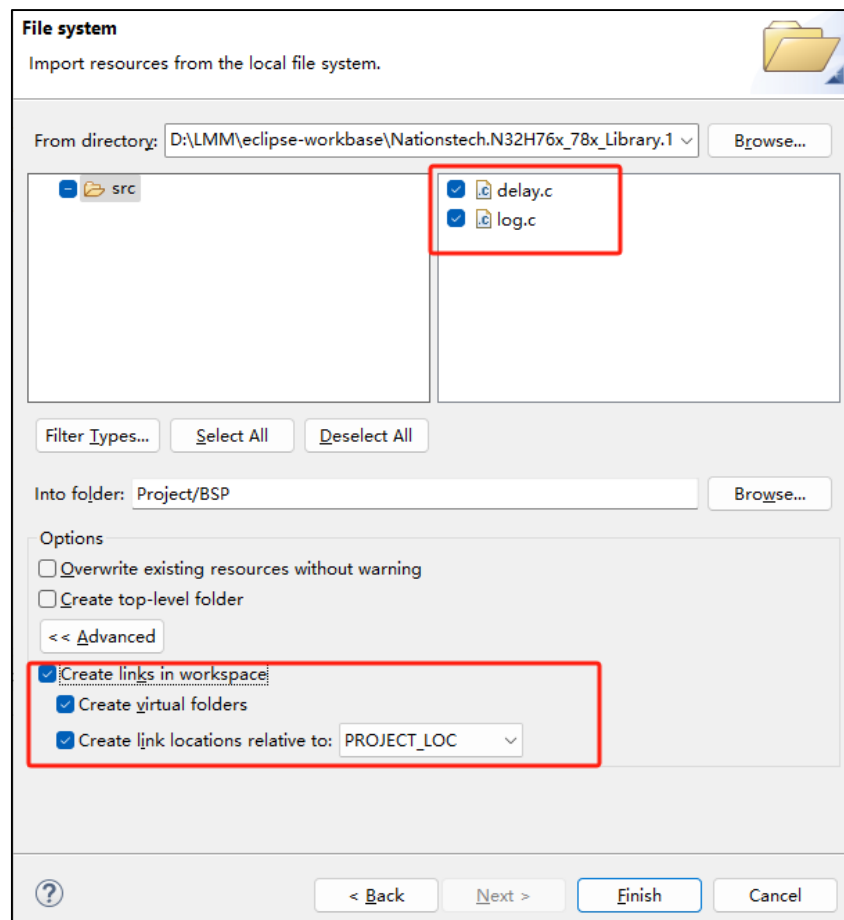
- 1) Right-click the project name and select **New > Folder**.



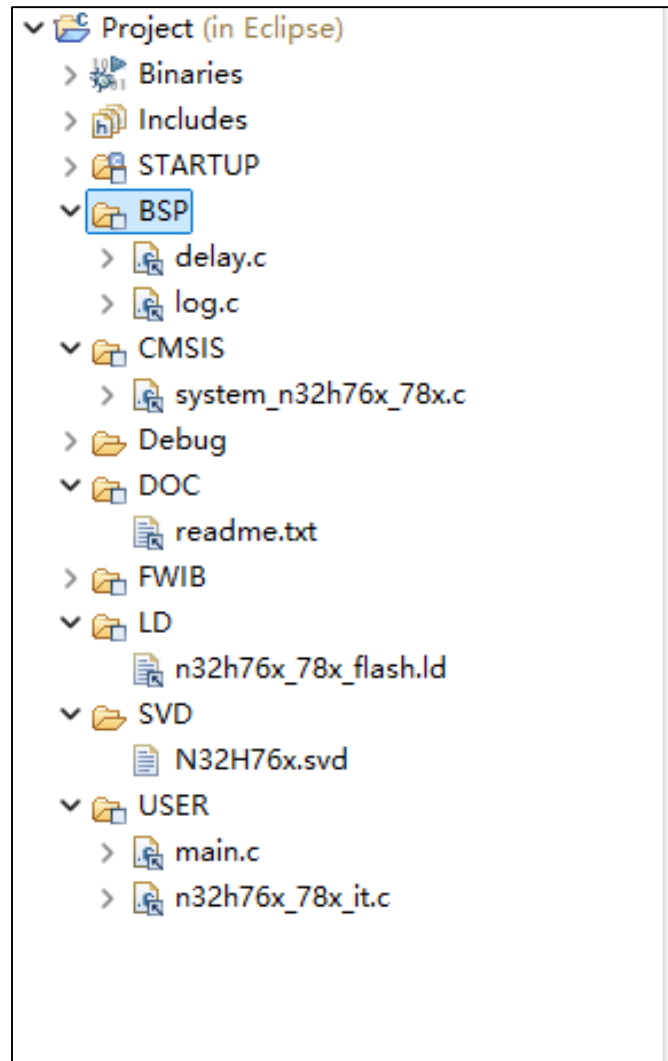
- 2) Create sub-virtual folders named STARTUP, BSP, CMSIS, DOC, FWIB, LD, SVD, and USER respectively.



- 3) Right-click BSP and select the **Import** option to import files into the BSP folder. Similarly, the import method for other folders is the same, and redundant descriptions are omitted here.



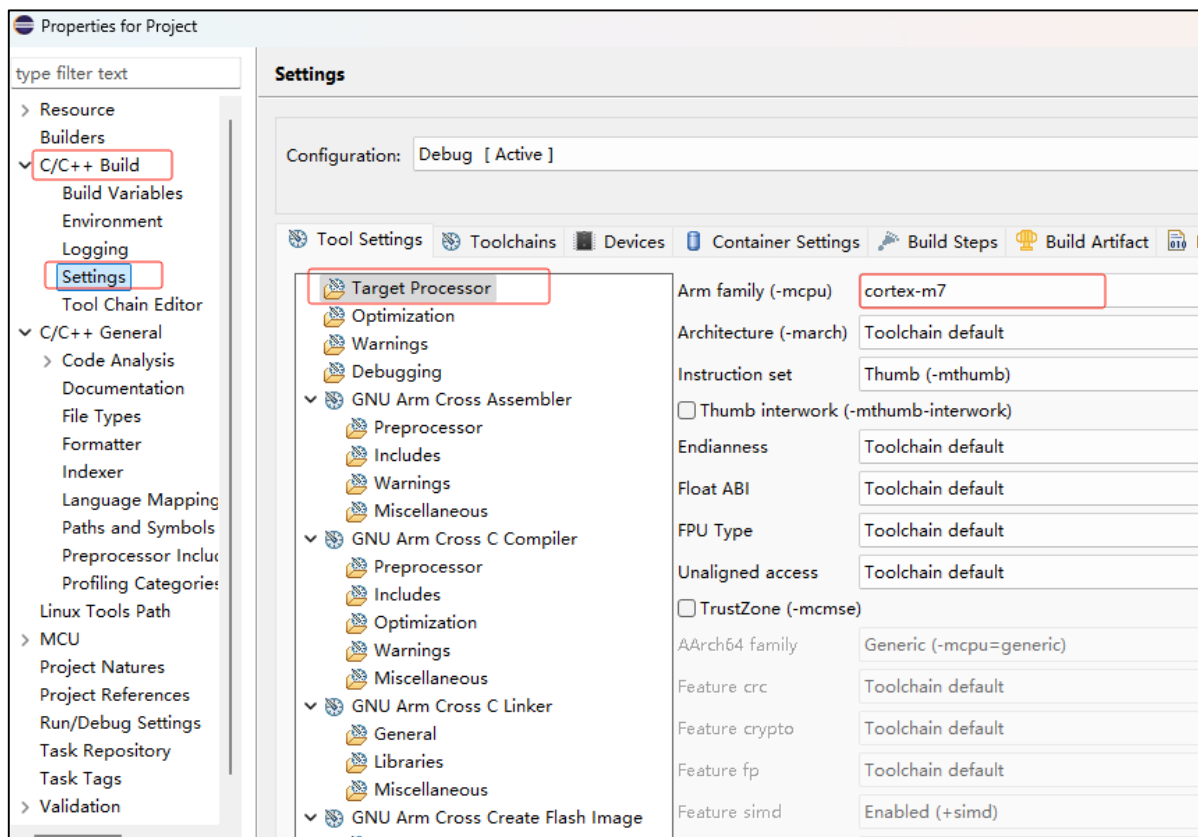
4) After the import is completed, the file structure is shown in the figure below.



5.3 Project Configuration

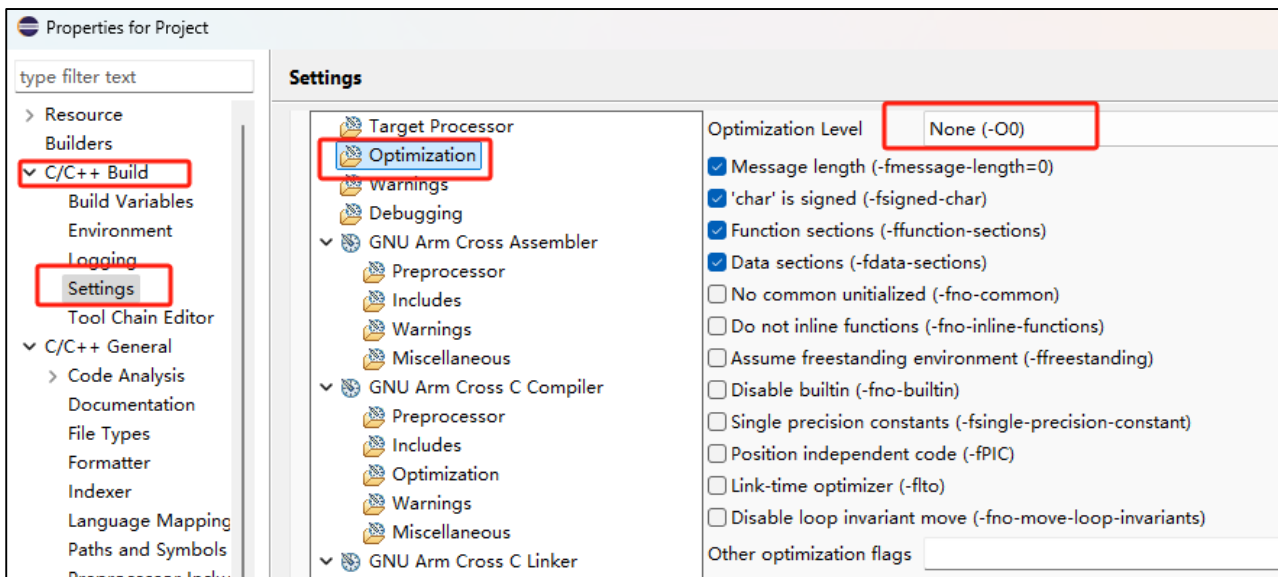
5.3.1 Chip Selection

Right-click the project and select the **Properties** option to open the project properties. Under **C/C++ Build > Settings > Tool Settings > Target Processor**, configure as follows: select **cortex-m3**, **cortex-m4**, **cortex-m0** or **cortex-m7** according to the core of the target chip. In this example, **cortex-m7** is selected, as shown in the figure below.



5.3.2 Optimization Level Configuration

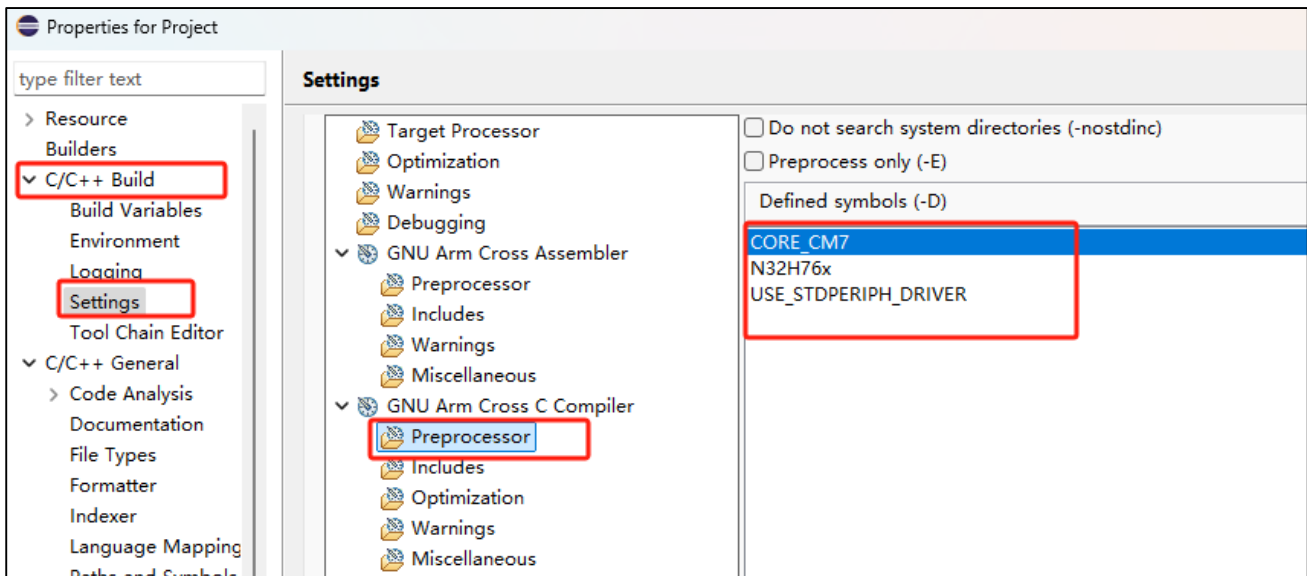
In the **C/C++ Build > Settings > Tool Settings > Optimization** option, configure the optimization level. The available options are **-O0**, **-O1**, **-O2**, **-O3**, **-Os**, **-Ofast**, and **-Og**.



5.3.3 GNU Arm Cross C Compiler Macro Configuration

In the C/C++ Build > Settings > Tool Settings > GNU Arm Cross C Compiler option, configure the Cross C Compiler options.

In this example, add the precompilation macros CORE_CM7, N32H76x and USE_STDPERIPH_DRIVER in the Preprocessor > Defined symbols (-D) option.



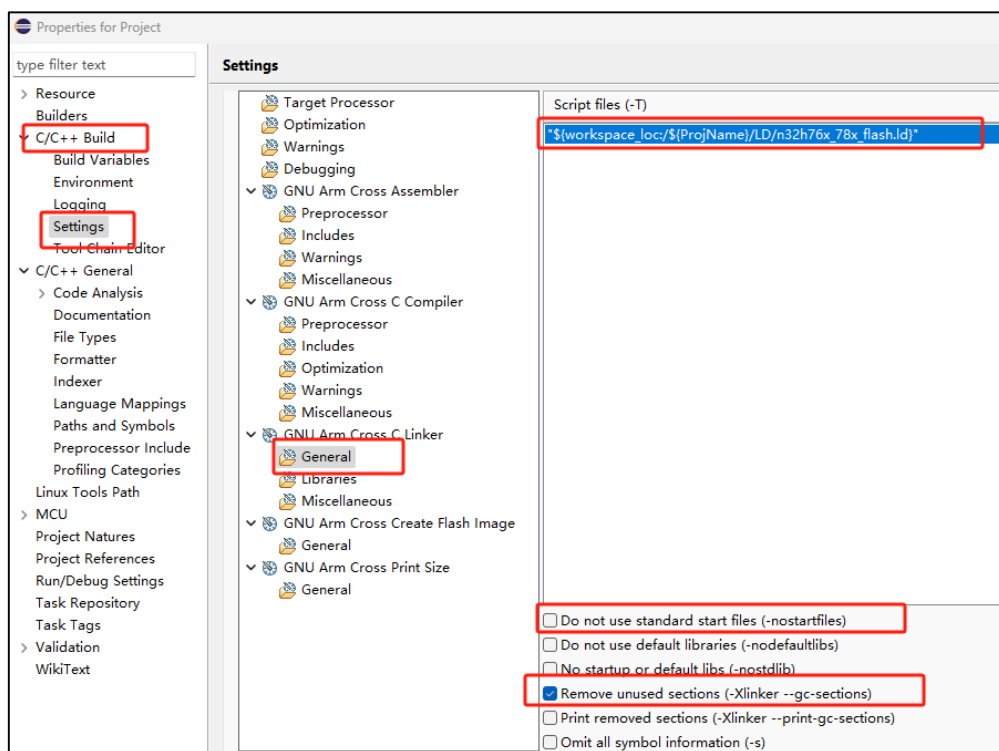
在”includes->Include paths”选项中添加工程所需的头文件路径。在本例中添加：

```
"${ProjDirPath}/../../../../firmware/n32h76x_78x_std_periph_driver/inc"
"${ProjDirPath}/../../../../firmware/CMSIS/core"
"${ProjDirPath}/../../../../firmware/CMSIS/device"
"${ProjDirPath}/../../../../bsp/inc"
"${ProjDirPath}/../inc"
```

5.3.4 GNU Arm Cross C Linker Configuration

Configure the Cross C Linker options in **C/C++ Build > Settings > Tool Settings > GNU Arm Cross C Linker**. Add the following content in the **General > Script files** option:

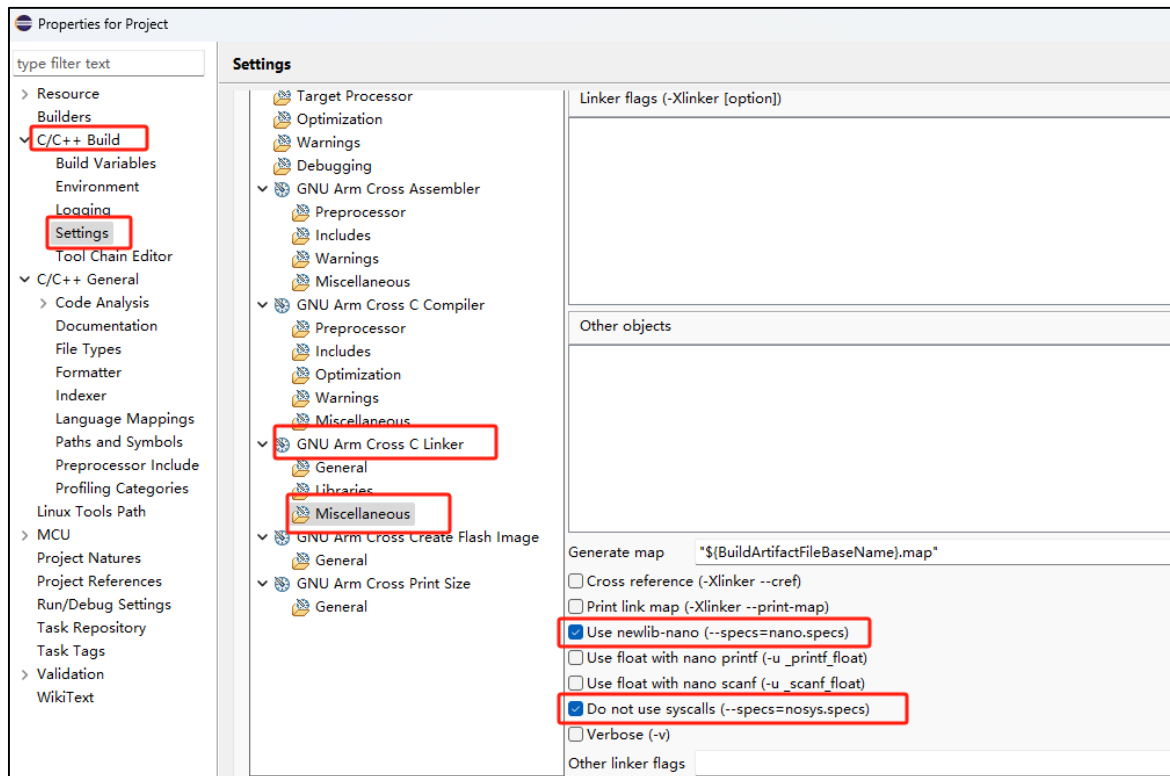
```
"${workspace_loc}/${ProjName}/LD/n32h76x_78x_flash.ld"
```



The linker script is responsible for telling the linker how to configure the memory for the compiled executable file. The linker script used shall match the FLASH and SRAM sizes of the target chip as well as the memory configuration required by the customer.

Note: Uncheck the option "Do not use standard start files".

In the **Miscellaneous** option, check the boxes for **Use newlib-nano** and **Do not use syscalls**. (This optimizes the code size.)

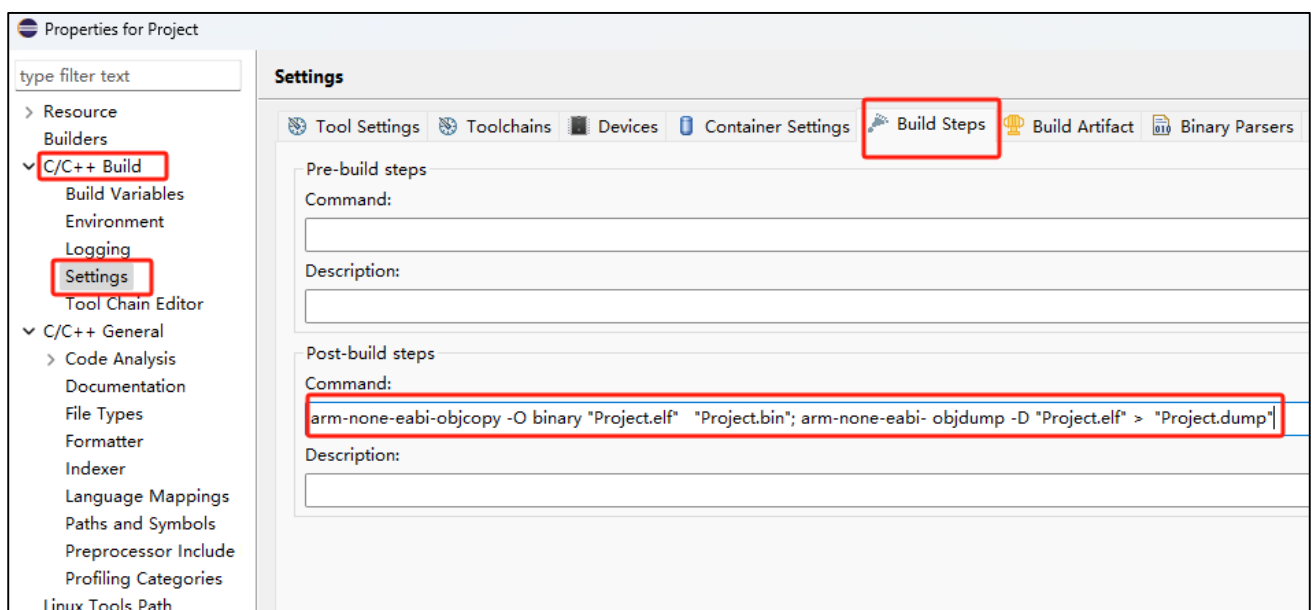


5.3.5 Configure Bin File Generation

Commands can be added in **C/C++ Build > Settings > Build Steps** to generate bin/hex files.

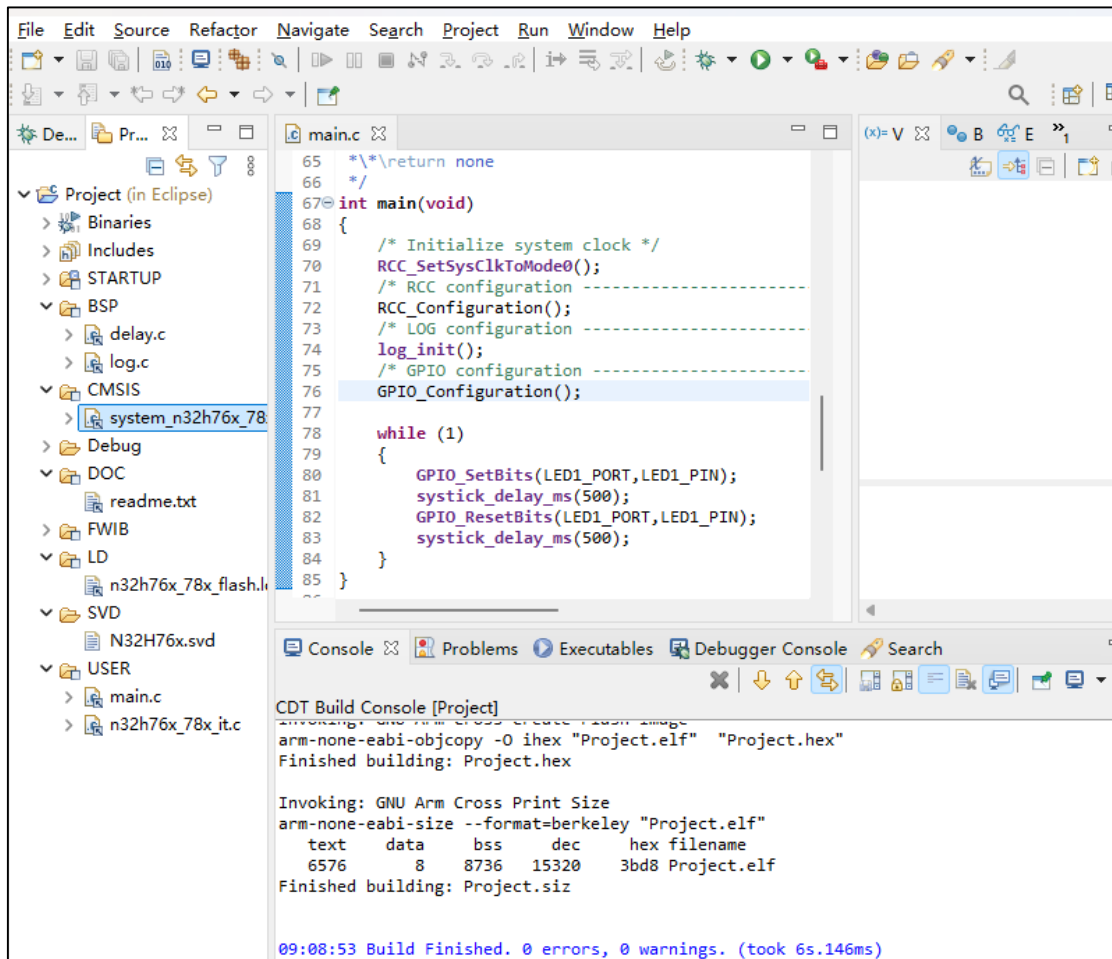
In this example, add the following commands:

```
arm-none-eabi-objcopy -O binary "Project.elf" "Project.bin"; arm-none-eabi-objdump -D "Project.elf" > "Project.dump"
```



6 Compilation

Select **Project > Build Project** to compile the current project. Save the current project before each compilation; otherwise, the previous version of the project will be compiled instead. After making modifications, to ensure correctness, please **clean** the project first before building it. Upon compilation completion, the corresponding **elf**, **hex**, and **bin** files will be generated.



```

65  /*\*/return none
66  */
67  int main(void)
68  {
69      /* Initialize system clock */
70      RCC_SetSysClkToMode0();
71      /* RCC configuration -----
72      RCC_Configuration();
73      /* LOG configuration -----
74      log_init();
75      /* GPIO configuration -----
76      GPIO_Configuration();
77
78      while (1)
79      {
80          GPIO_SetBits(LED1_PORT, LED1_PIN);
81          systick_delay_ms(500);
82          GPIO_ResetBits(LED1_PORT, LED1_PIN);
83          systick_delay_ms(500);
84      }
85  }

```

```

CDT Build Console [Project]
Invoking: GNU Arm Cross Create Flash Image
arm-none-eabi-objcopy -O ihex "Project.elf" "Project.hex"
Finished building: Project.hex

Invoking: GNU Arm Cross Print Size
arm-none-eabi-size --format=berkeley "Project.elf"
   text  data   bss   dec   hex filename
  6576     8   8736  15320  3bd8 Project.elf
Finished building: Project.siz

09:08:53 Build Finished. 0 errors, 0 warnings. (took 6s.146ms)

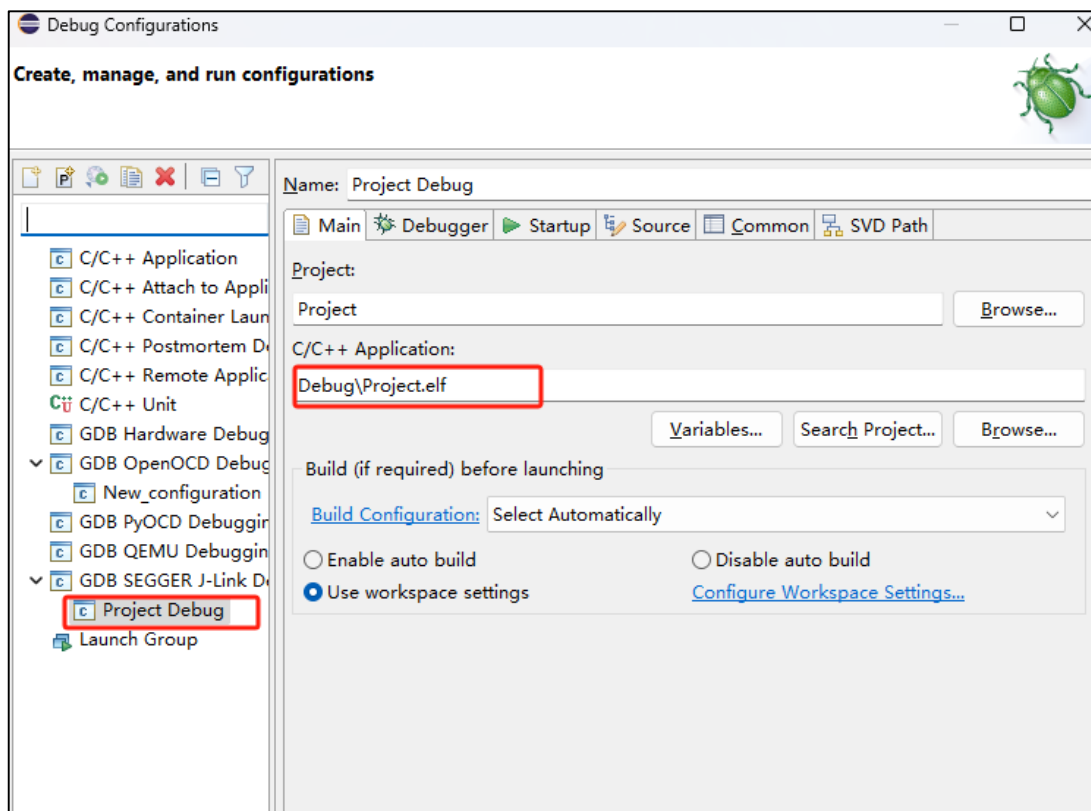
```

7 J-Link Download and Debugging

Click **Run > Debug Configurations** in the menu bar to open the Debug configuration interface. Select **J-Link GDB Server CL** as the GDB Server, and use the GDB tool in the GCC toolchain as the GDB Client. Double-click **GDB SEGGER J-Link Debugging** to create a new set of J-Link configuration options.

7.1 Main Tab

In the **Main** tab, select the current project. Normally, the elf file under the current project will be added automatically. If not, click **Browse** to manually add the elf file.

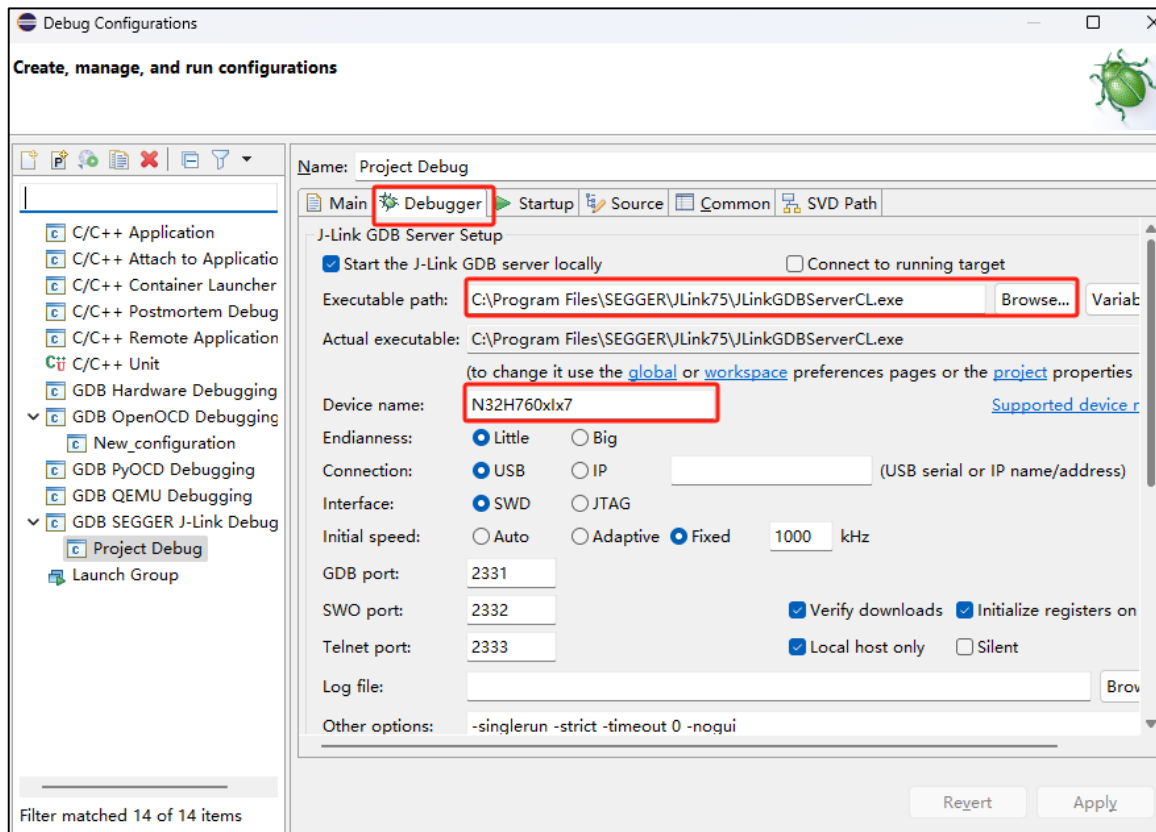


7.2 Debugger Tab

In the **Debugger** tab, fill in the Device name of the target chip, which is N32H760xIx7 in this example.

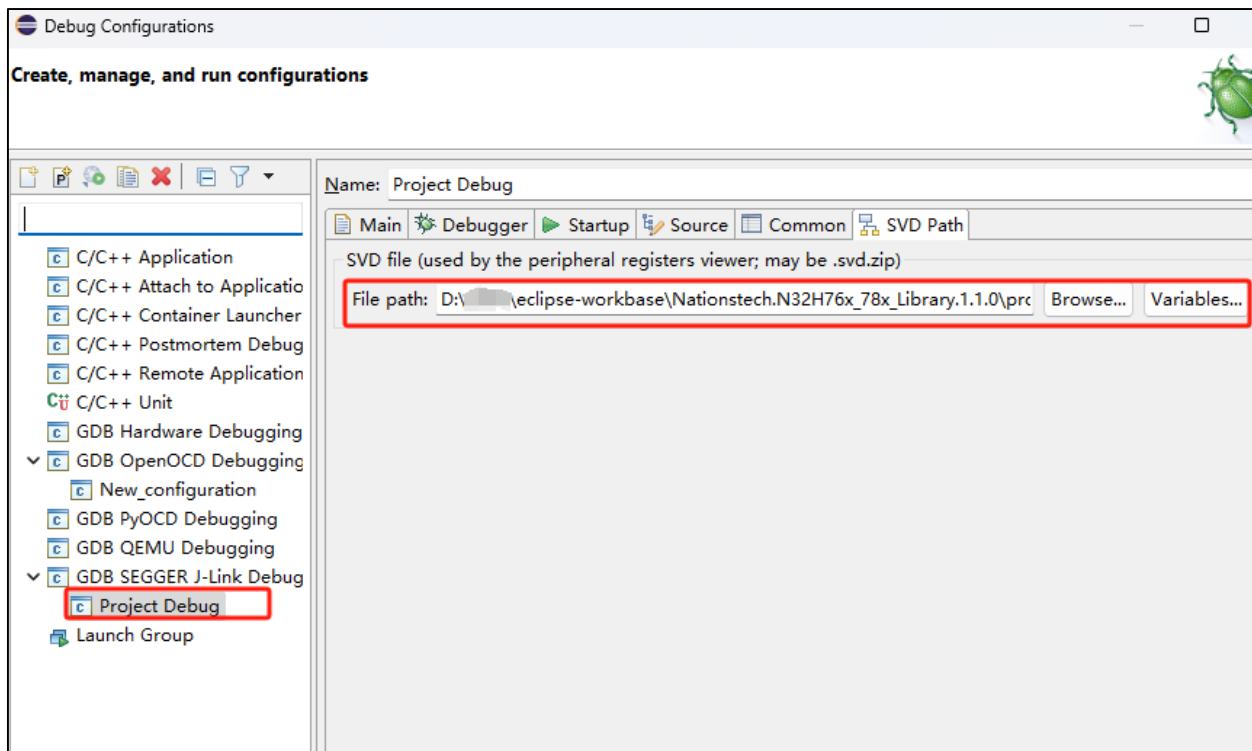
If the J-Link path was configured correctly during the Eclipse environment setup, it will be recognized automatically here. If it was not configured correctly previously, you can also select the absolute path of **J-Link GDB Server CL** in the **Executable path** field.

Note: The filled-in chip model must be supported by the configured J-Link driver.



7.3 SVD Path Tab

In the **SVD Path** tab, select the SVD file required by the target chip.



7.4 Debug Interface

After the J-Link is connected to the device and the **Debug Configurations** are configured, click **Debug** to enter the Debug perspective, then switch to the **DEBUG** perspective to start normal debugging.

The screenshot shows the Eclipse IDE interface for a C project. The main editor window displays the source code of 'main.c'. The code includes headers for 'main.h', 'delay.h', and 'log.h'. It defines two functions: 'GPIO_Configuration(void)' and 'RCC_Configuration(void)'. The 'main()' function initializes the system clock, configures the GPIO, and enters a loop that sets bits, delays, and resets bits. The left sidebar shows the project structure with 'Project Debug [GDB SEGGER J-Link Debugging]' selected. The bottom status bar shows the current address as '70 : 1 : 3344'.

```

55 #include "main.h"
56 #include "delay.h"
57 #include "log.h"
58
59 void GPIO_Configuration(void);
60 void RCC_Configuration(void);
61 /**
62  * \name main.
63  * \fun Main program.
64  * \param none
65  * \return none
66  */
67 int main(void)
68 {
69     /* Initialize system clock */
70     RCC_SetSysClkToMode0();
71     /* RCC configuration -----
72     RCC_Configuration();
73     /* LOG configuration -----
74     log_init();
75     /* GPIO configuration -----
76     GPIO_Configuration();
77
78     while (1)
79     {
80         GPIO_SetBits(LED1_PORT, LED1_PIN);
81         systick_delay_ms(500);
82         GPIO_ResetBits(LED1_PORT, LED1_PIN);
83         systick_delay_ms(500);
84     }
85 }

```

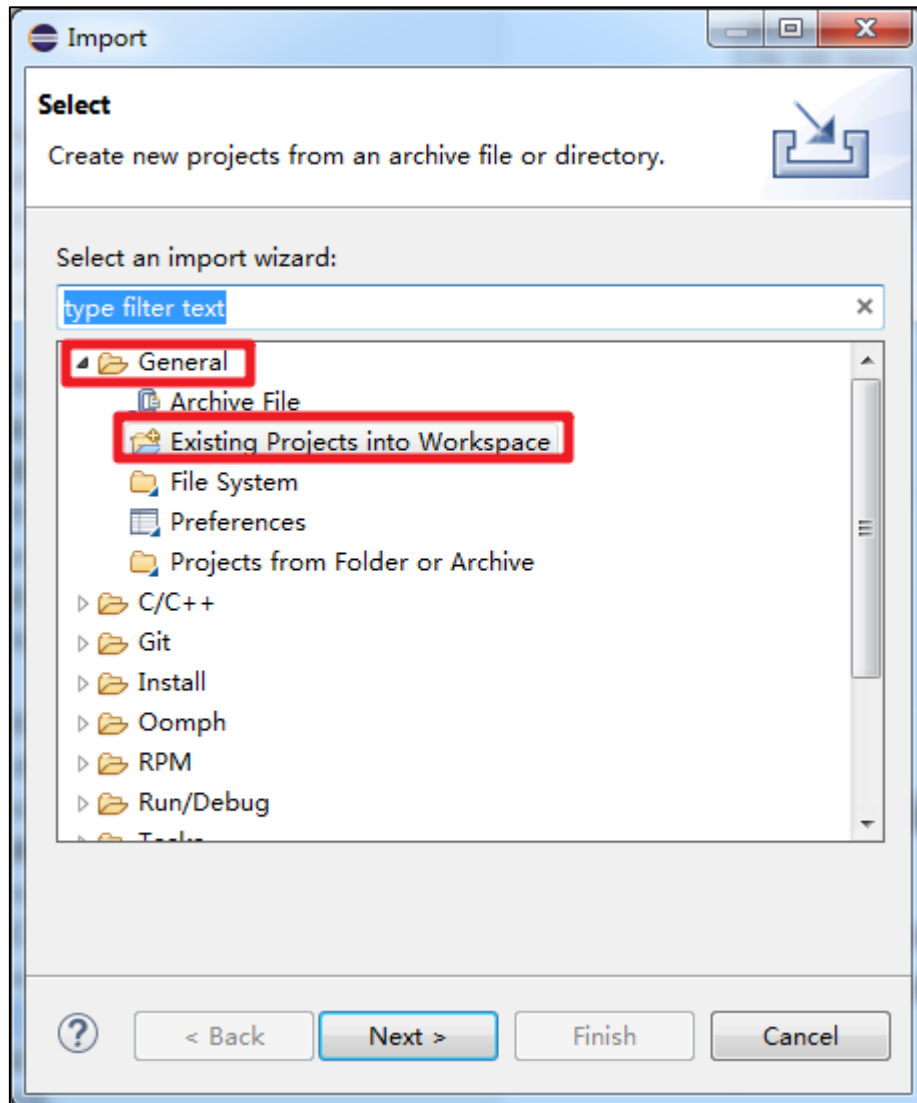
Project Debug [GDB SEGGER J-Link Debugging]

egfpcr" itize=32 r2" tpui" t/> egso bttze=32 r3" tpflt/> egsl bttze=32 r4" tpflt/> e
egfpcr" itize=32 r2" tpui" t/> egso bttze=32 r3" tpflt/> egsl bttze=32 r4" tpflt/> e
egfpcr" itize=32 r2" tpui" t/> egso bttze=32 r3" tpflt/> egsl bttze=32 r4" tpflt/> e

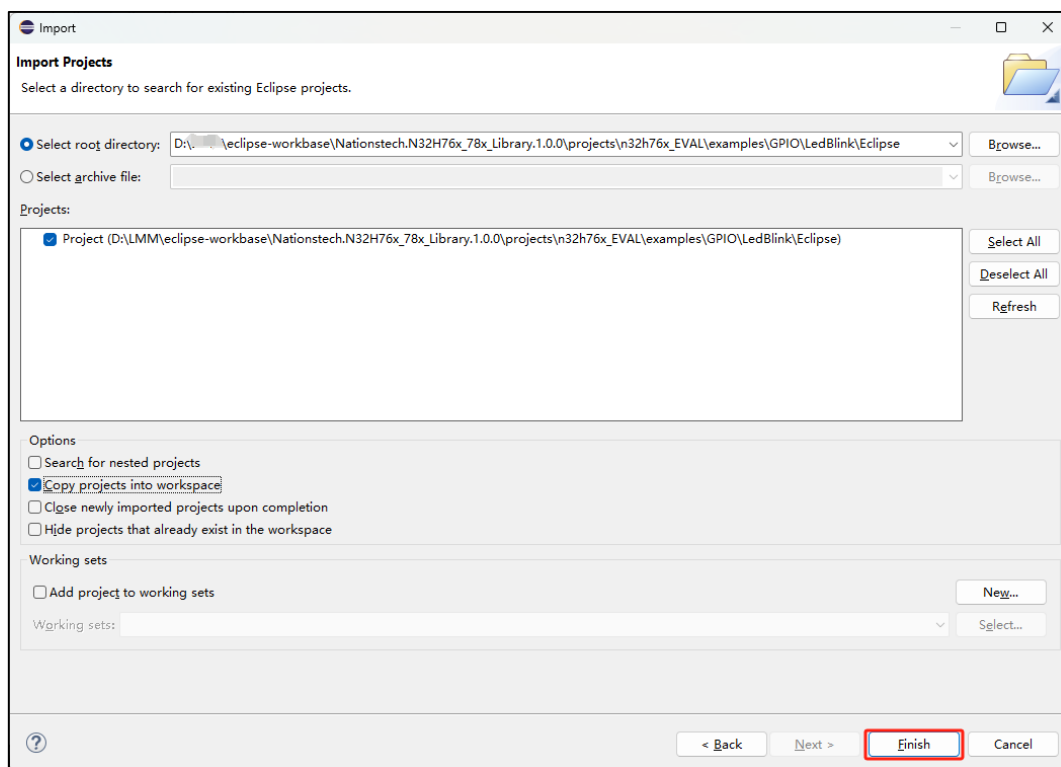
Writable Smart Insert 70 : 1 : 3344

8 Import Existing Projects

- 1) Besides creating a new project, you can also directly import an existing Eclipse project. In the menu bar, click **File > Import** and select **General > Existing Projects into Workspace** to import the existing project, then click **Next**.



- 2) Select the path of the existing project files. Eclipse will recognize all projects under this path. Select the corresponding project and click **Finish** to import the existing project.



9 Version history

Date	Version	Modify
2025/10/21	V1.0.0	The initial release

10 Notice

This document is the exclusive property of NSING TECHNOLOGIES PTE. LTD. (Hereinafter referred to as NSING). This document, and the product of NSING described herein (Hereinafter referred to as the Product) are owned by NSING under the laws and treaties of Republic of Singapore and other applicable jurisdictions worldwide. The intellectual properties of the product belong to NSING Technologies Inc. and NSING Technologies Inc. does not grant any third party any license under its patents, copyrights, trademarks, or other intellectual property rights. Names and brands of third party may be mentioned or referred thereto (if any) for identification purposes only. NSING reserves the right to make changes, corrections, enhancements, modifications, and improvements to this document at any time without notice. Please contact NSING and obtain the latest version of this document before placing orders. Although NSING has attempted to provide accurate and reliable information, NSING assumes no responsibility for the accuracy and reliability of this document. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. In no event shall NSING be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising in any way out of the use of this document or the Product. NSING Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, Insecure Usage'. Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, all types of safety devices, and other applications intended to supporter sustain life. All Insecure Usage shall be made at user's risk. User shall indemnify NSING and hold NSING harmless from and against all claims, costs, damages, and other liabilities, arising from or related to any customer's Insecure Usage Any express or implied warranty with regard to this document or the Product, including, but not limited to. The warranties of merchantability, fitness for a particular purpose and non-infringement are disclaimed to the fullest extent permitted by law. Unless otherwise explicitly permitted by NSING, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.