



# 基于 Windows 的 ARM Eclipse 开发环境

## 目录

1. 概述 .....	3
2. 开发工具 .....	3
2.1 软件 .....	3
2.2 硬件 .....	3
3. 开发环境搭建 .....	4
3.1 安装 ARM 交叉编译工具链 .....	4
3.2 Build 工具安装 .....	4
3.3 Eclipse IDE 安装 .....	4
3.3.1 JDK 安装 .....	4
3.3.2 Eclipse IDE for GNU ARM C/C++ Developers 安装 .....	4
3.4 安装 JLink 工具 .....	5
3.5 添加芯片支持 .....	5
3.6 JLink 下载测试 .....	5
4. Eclipse 启动及配置 .....	7
4.1 新建 Workspace .....	7
4.2 设置 Build 工具路径 .....	8
4.3 设置交叉编译工具路径 .....	9
4.4 设置 SEGGER J-Link Path .....	10
5. 工程新建及配置 .....	11
5.1 新建工程 .....	11
5.2 文件夹新增以及文件导入 .....	13
5.3 工程配置 .....	15
5.3.1 芯片选择 .....	15
5.3.2 优化等级配置 .....	16
5.3.3 GNU Arm Cross C Compiler 宏配置 .....	17
5.3.4 GNU Arm Cross C Linker 配置 .....	17
5.3.5 配置生成 bin 文件 .....	19
6. 编译 .....	20
7. JLINK 下载及调试 .....	21
7.1 Main 选项卡 .....	21
7.2 Debugger 选项卡 .....	21
7.3 SVD Path 选项卡 .....	22
7.4 Debug 界面 .....	23
8. 导入现有的工程 .....	24
9. 版本历史 .....	26
10. 声明 .....	27

# 1.概述

本文以 N32H760 系列 MCU 为例，本文介绍了如何搭建 N32 Eclipse 开发环境。适用于所有 N32 MCU 系列。

## 2.开发工具

### 2.1 软件

- 1) 操作系统：WIN7 / WIN10 64-bit OS
- 2) IDE: Eclipse IDE for GNU ARM & RISC-V C/C++ Developers
- 3) 交叉编译链：arm-none-eabi-gcc
- 4) 编译工具：GNU MCU Eclipse build tools
- 5) GDB 服务器：J-Link GDB Server
- 6) 调试器：J-Link V9/V10

### 2.2 硬件

- 1) 开发板 N32H760xIx7-STB V1.0
- 2) JLink 下载器 V9 或以上

## 3. 开发环境搭建

### 3.1 安装 ARM 交叉编译工具链

➤ 下载 xpack-arm-none-eabi-gcc-10.2.1-1.1-win32-x64.zip 文件

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

访问 <https://github.com/xpack-dev-tools/arm-none-eabi-gcc-xpack/releases/> 可选择下载其他不同版本的 ARM 交叉编译链。

### 3.2 Build 工具安装

➤ 下载 xPack Windows Build Tools v4.2.1-2-W32-X64.zip 文件

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

访问 <https://xpack.github.io/windows-build-tools/releases/> 可选择下载不同版本的编译工具。

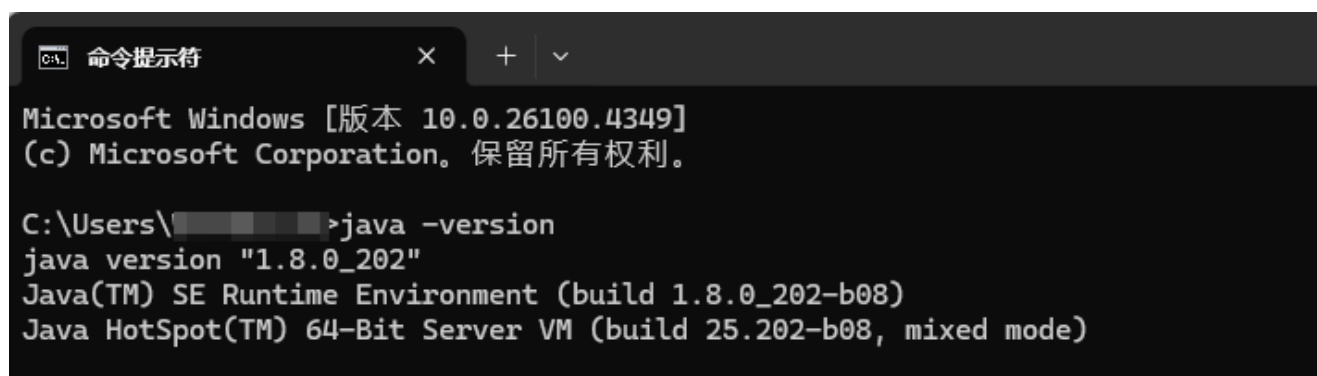
### 3.3 Eclipse IDE 安装

#### 3.3.1 JDK 安装

➤ 下载 the jdk-8u202-windows-x64.exe 文件

Eclipse 需要运行在 Java 环境，所以在安装 Eclipse 之前需要先安装 JDK。访问 <http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html> 可选择下载不同版本的 JDK 工具。本文中選擇 jdk-8u202-windows-x64.exe 下载并安装，安装 the jdk-8u202-windows-x64.exe 文件完成后。

检查是否安装成功：打开 dos 命令行窗口，键入 java -version 测试 JDK 是否正确安装。如果 JDK 已正确安装，可得到如图所示类似输出。



```
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)
```

注意：若不成功，请检查环境变量是否添加好

#### 3.3.2 Eclipse IDE for GNU ARM C/C++ Developers 安装

➤ 下载 Download the eclipse-embedcpp-2021-03-R-win32-x86\_64.zip 文件

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

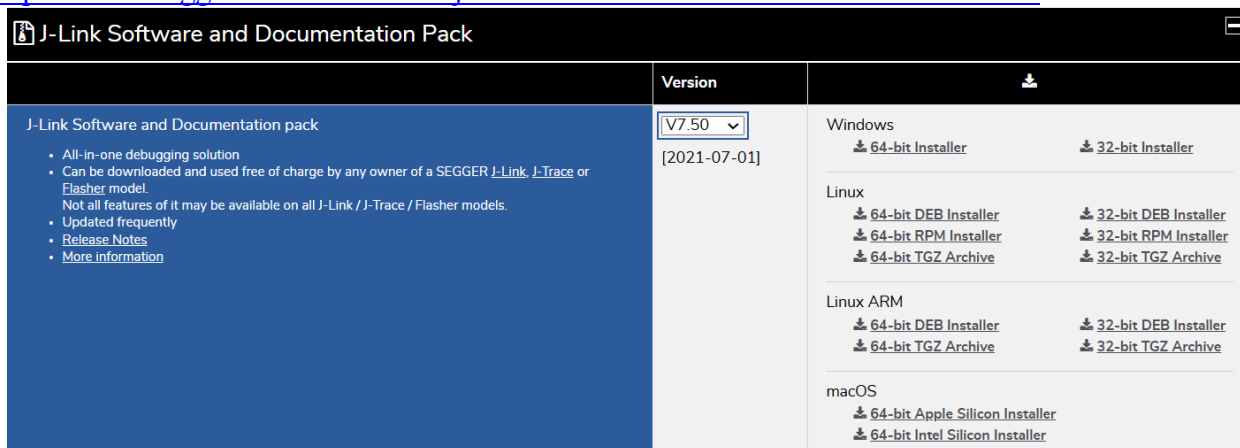
[embedcpp-2021-03-R-win32-x86\\_64.zip](#)。

访问 <https://eclipse-embed-cdt.github.io/packages/releases/> 可选择下载不同版本的 Eclipse IDE。

## 3.4 安装 JLink 工具

➤ 下载 JLINK 安装包, V7.50 或其他版本

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



## 3.5 添加芯片支持

安装好 JLink 之后需要向 JLink 中添加我们公司的芯片补丁包，以便在下载、调试时正确获取到下载算法。具体请参考文档《jlink 工具添加 Nations 芯片.7z》

## 3.6 JLink 下载测试

➤ 测试 JLink 环境安装

1. 连接好 PC 和 J-Link 调试器，连接好开发板，上电；
2. 打开 cmd.exe 命令行工具，进入 JLink 安装目录“C:\Program Files (x86)\SEGGER\JLink\_V750”下，输入“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>
```

如上图表示 PC 连接 JLink 调试器成功。

然后根据提示依次输入：“connect”，“N32H760XIX7”，“SWD”，“4000”，如果前面的操作成功，则会看到下面的输出信息，JLink 下载调试环境就可以正常使用了。

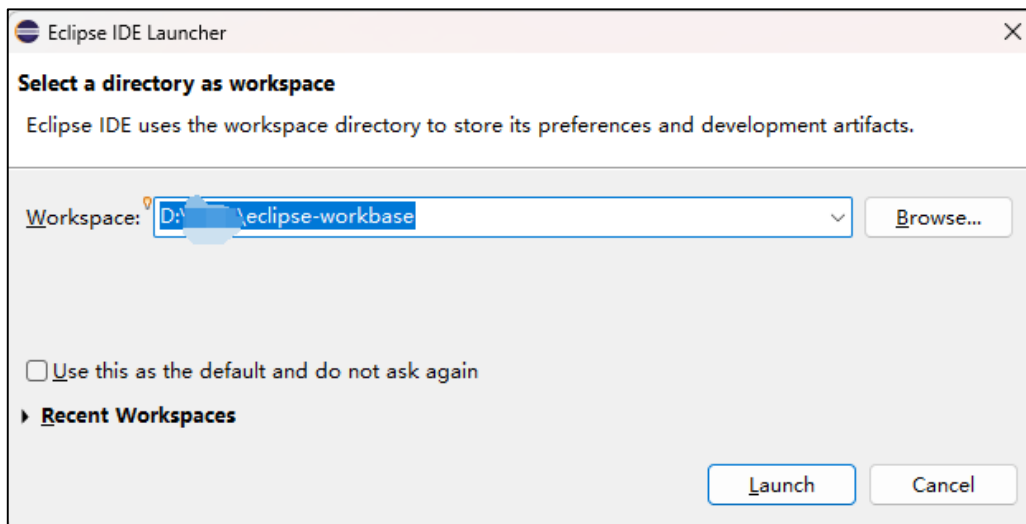
```
Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: N32H760XIX7
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>s
Specify target interface speed [kHz]. <Default>: 4000 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]: E000E000, 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>
```

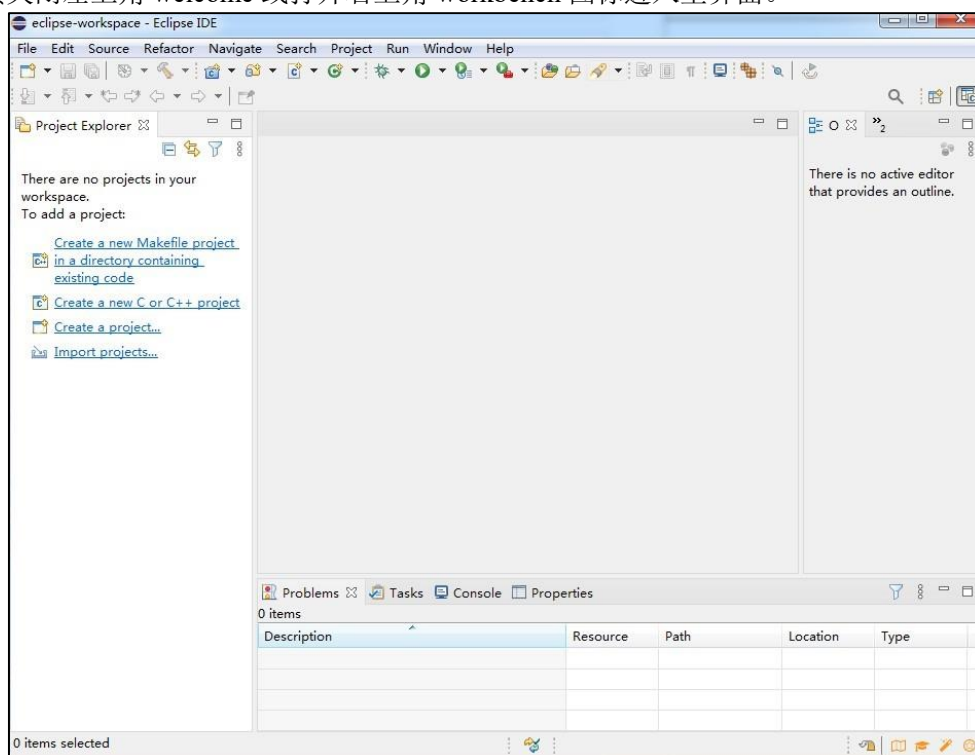
## 4. Eclipse 启动及配置

### 4.1 新建 Workspace

Eclipse 软件本身为绿色软件，无需安装，直接双击 eclipse 文件夹下的 eclipse.exe 来启动 Eclipse，启动之后如下图所示。

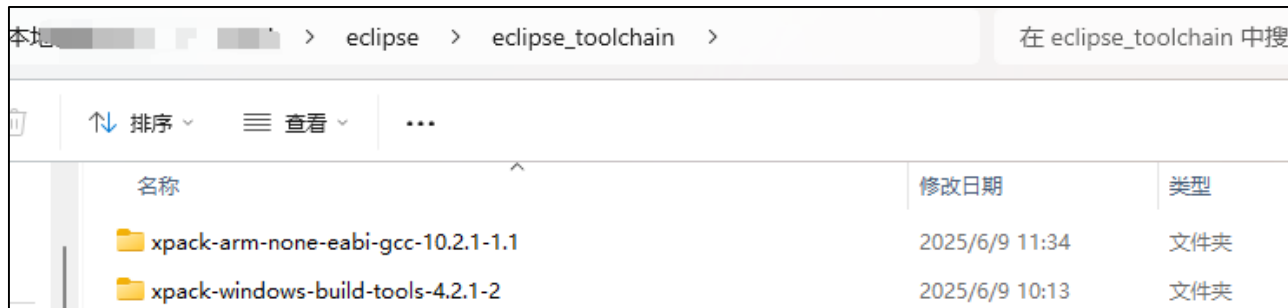


如下图所示，选择本地英文路径，创建 workspace。点击 Launch。注意路径深度不可太深。进入欢迎界面后可以选择直接关闭左上角 welcome 或打开右上角 workbench 图标进入主界面。

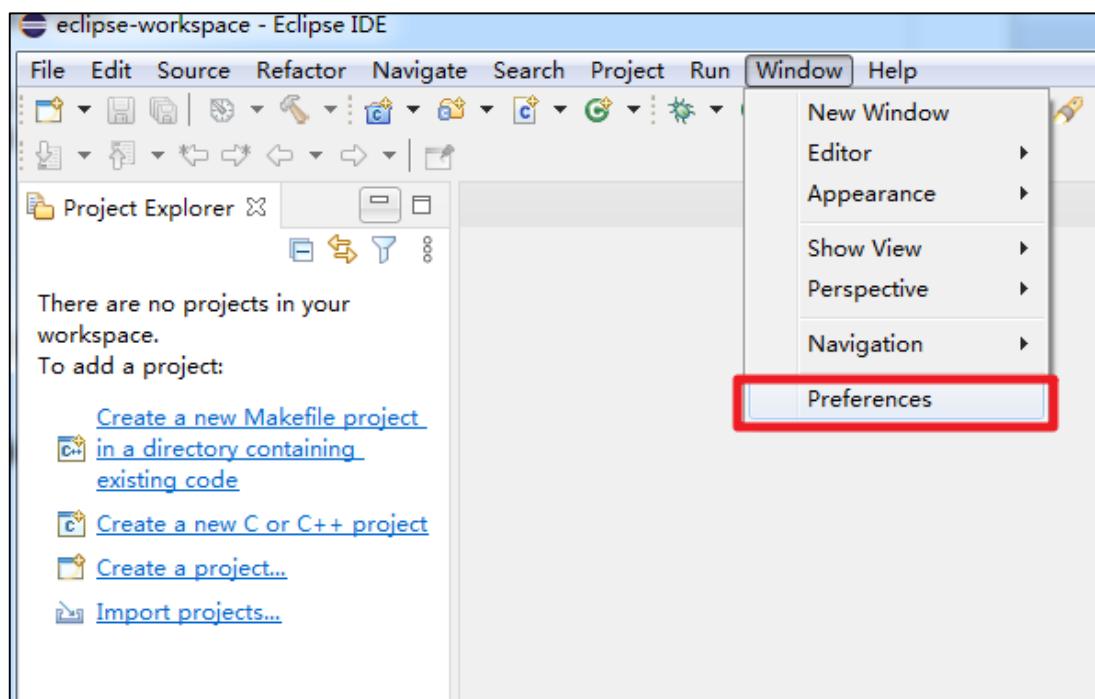


## 4.2 设置 Build 工具路径

在 Eclipse 安装路径下，建立 Eclipse\_toolchain 文件夹。将工具安装说明中下载的 ARM 交叉编译链，Build 工具均解压后放置在该文件夹内，如下图所示：

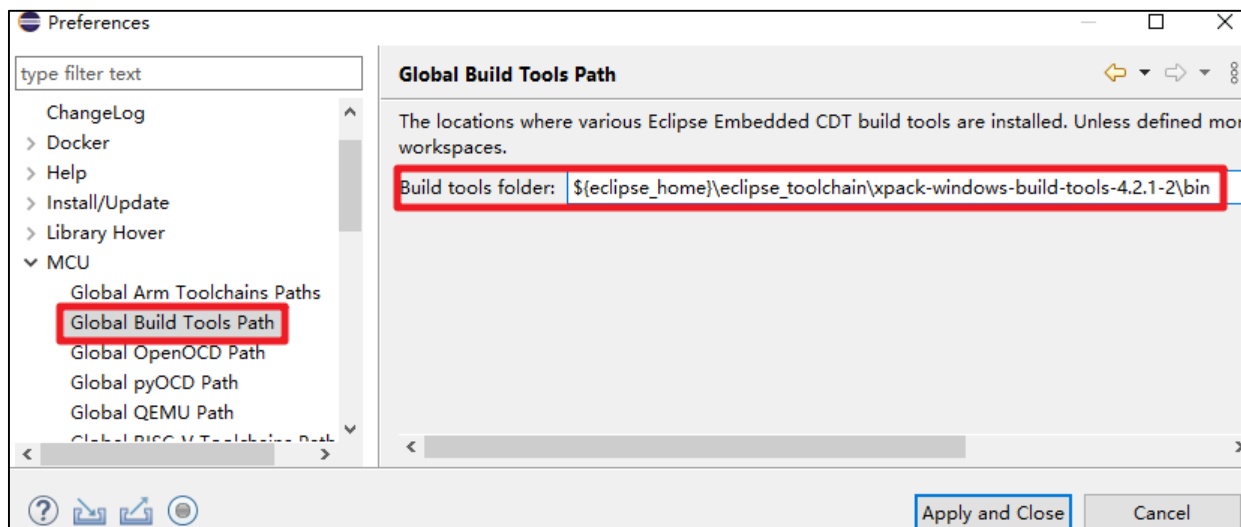


1) 打开 Winodow->Preferences 选项



2) 选择 MCU->Global Build Tools Path，设置全局 Build 工具路径：  
\${eclipse\_home}\eclipse\_toolchain\xpack-windows-build-tools-4.2.1-2\bin，这里需要配置相对路径；

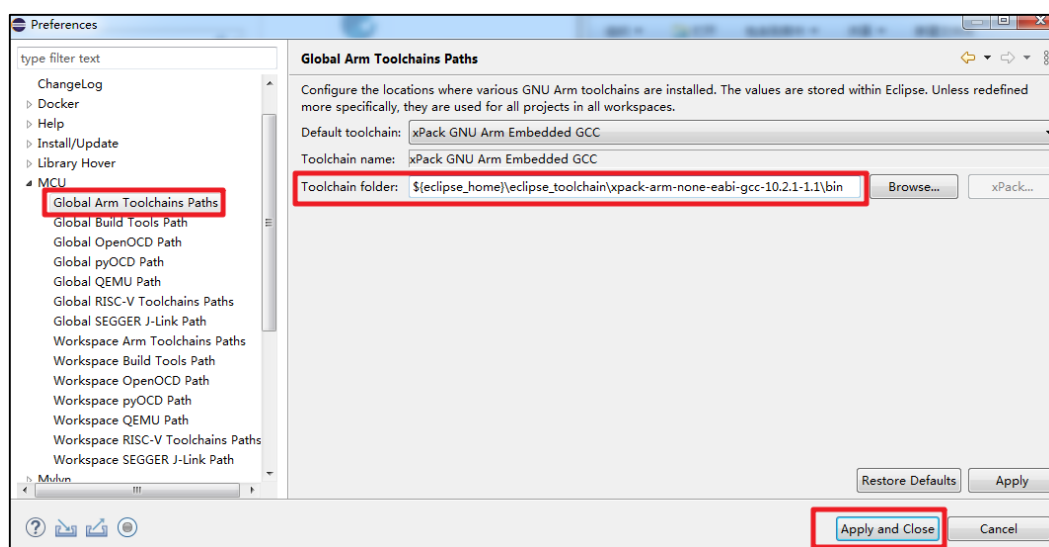




## 4.3 设置交叉编译工具路径

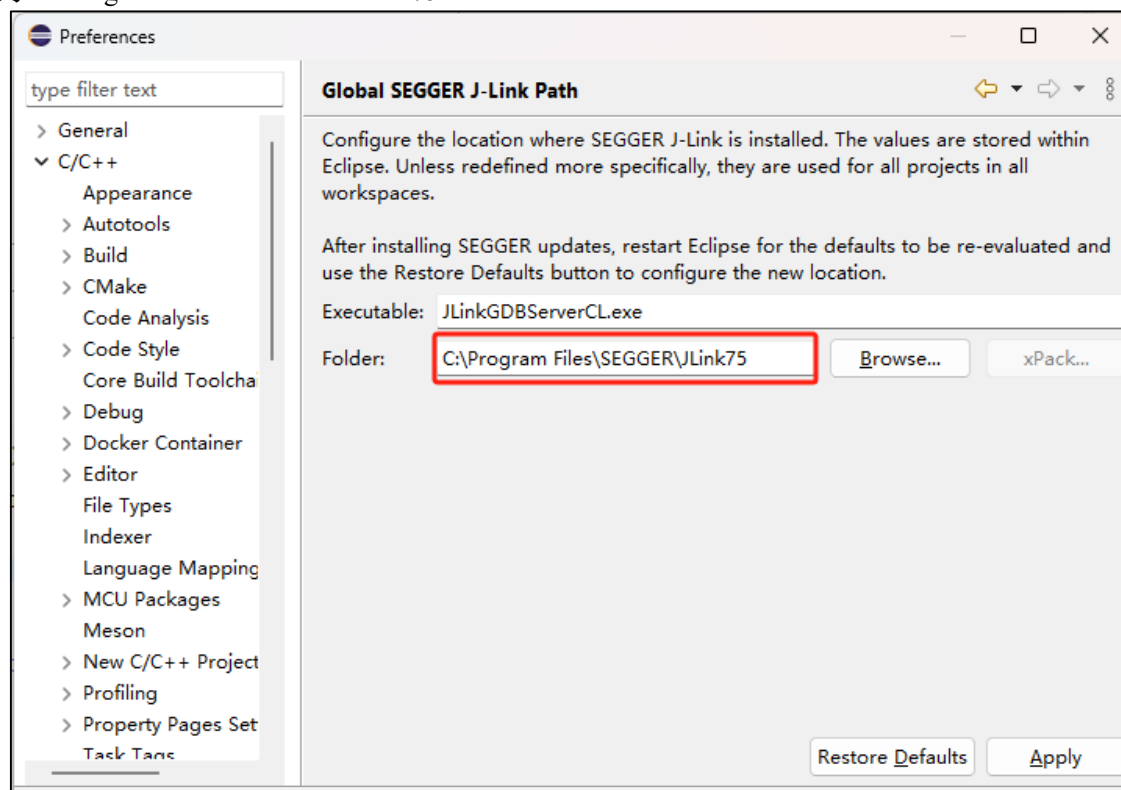
选择 MCU->Global Arm Toolchains Path, 设置全局Arm Toolchains 工具路径。

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



## 4.4 设置 SEGGER J-Link Path

选择 MCU->Global SEGGER J-Link Path，设置全局 SEGGER J-Link 工具路径，这里选择本地绝对路径，本例中路径为 C:\Program Files\SEGGER\JLink75



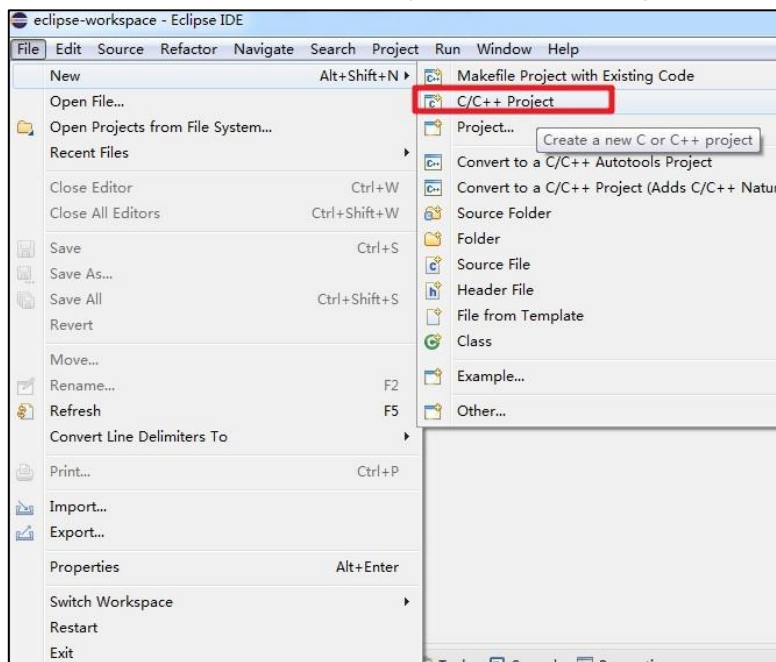
这里就完成 Eclipse IDE 的全部配置。

## 5.工程新建及配置

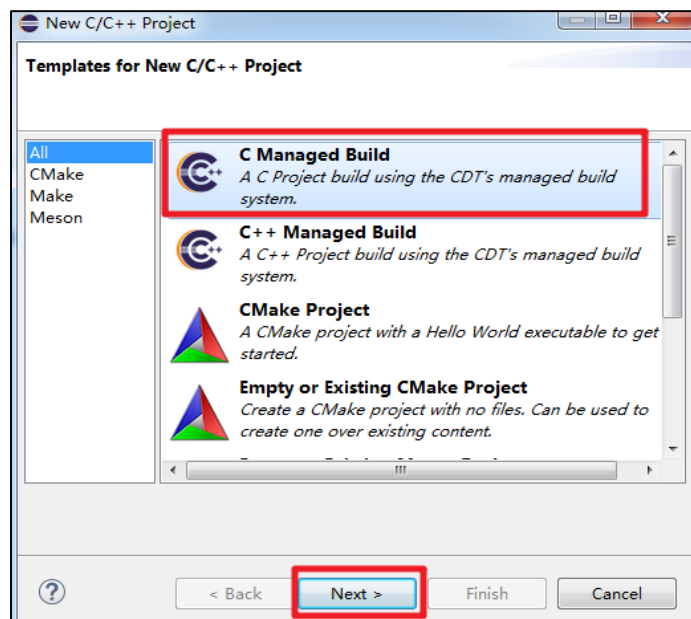
SDK 沿用已发的 SDK 版本，当前使用 v1.0.0，在此基础上做如下修改以适应 Eclipse 开发环境。

### 5.1 新建工程

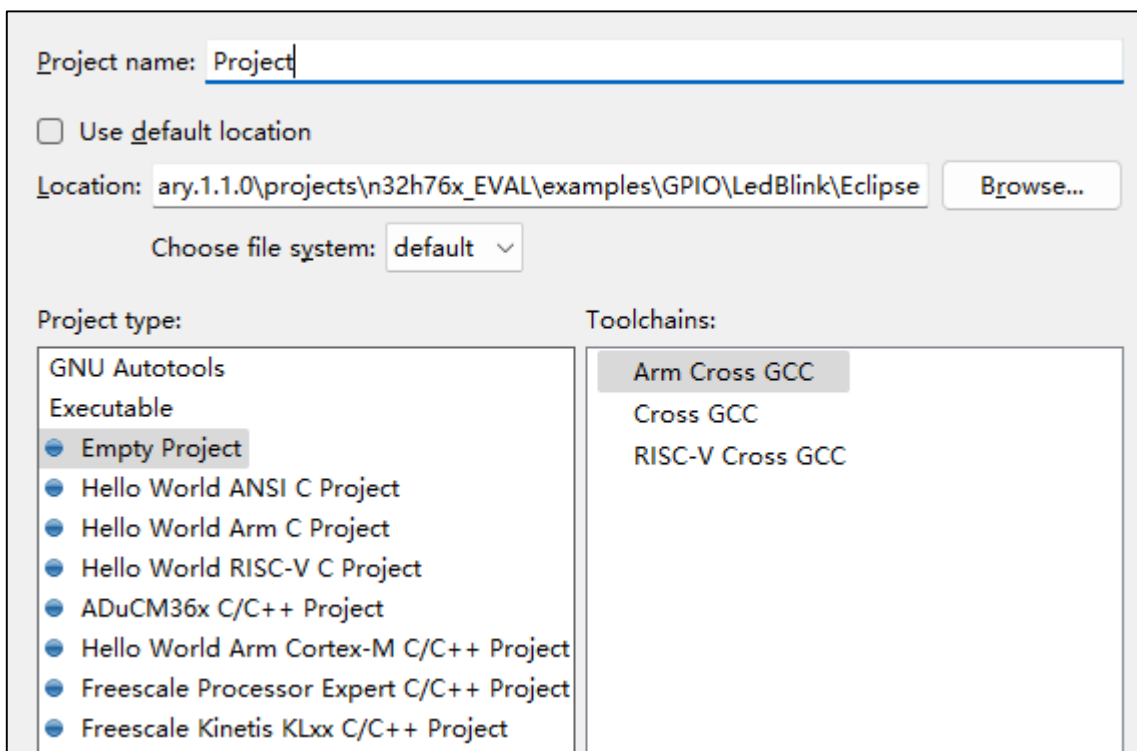
1.打开 Eclipse，在“File->New”下可选择新建 C/C++ Project，选择 C Managed Build。



2.选择 C Managed Build



3. 输入 Project name，配置工程类型，为了方便建议将工程放到 Workspace 目录下。编译链选择为 ARM Cross GCC



Project name:

☐ Use default location

Location:

Choose file system:

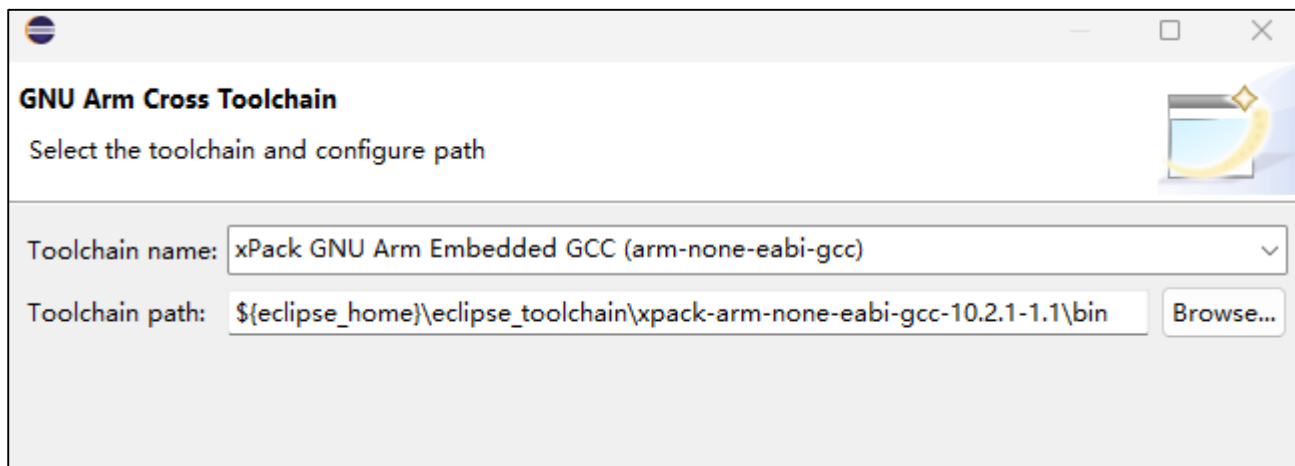
Project type:

- GNU Autotools Executable
- ☒ Empty Project
- ☐ Hello World ANSI C Project
- ☐ Hello World Arm C Project
- ☐ Hello World RISC-V C Project
- ☐ ADuCM36x C/C++ Project
- ☐ Hello World Arm Cortex-M C/C++ Project
- ☐ Freescale Processor Expert C/C++ Project
- ☐ Freescale Kinetis KLxx C/C++ Project

Toolchains:

- ☒ Arm Cross GCC
- ☐ Cross GCC
- ☐ RISC-V Cross GCC

4. 若 Eclipse IDE 已正确设置了 ARM Toolchains Path，这里将会自动选择路径。若 Eclipse IDE 未设置 ARM Toolchains Path，也可在这里选择到 ARM Toolchains 绝对路径。然后点击“Finish”即完成了工程的建立。



**GNU Arm Cross Toolchain**

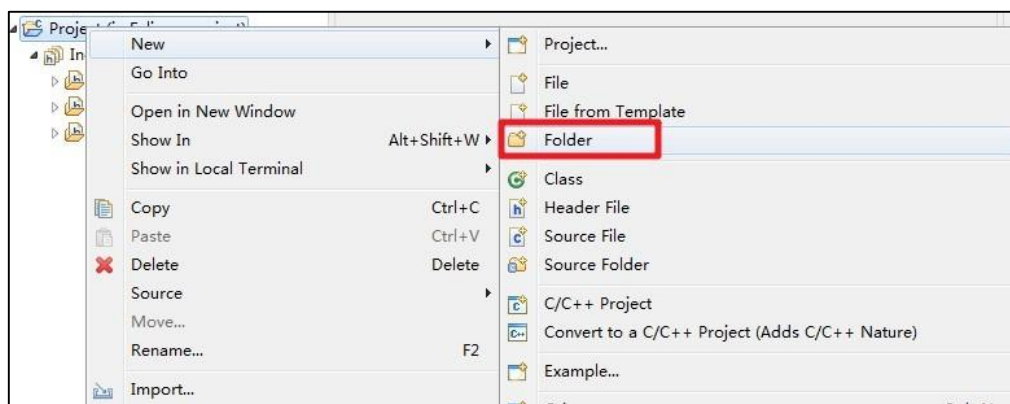
Select the toolchain and configure path

Toolchain name:

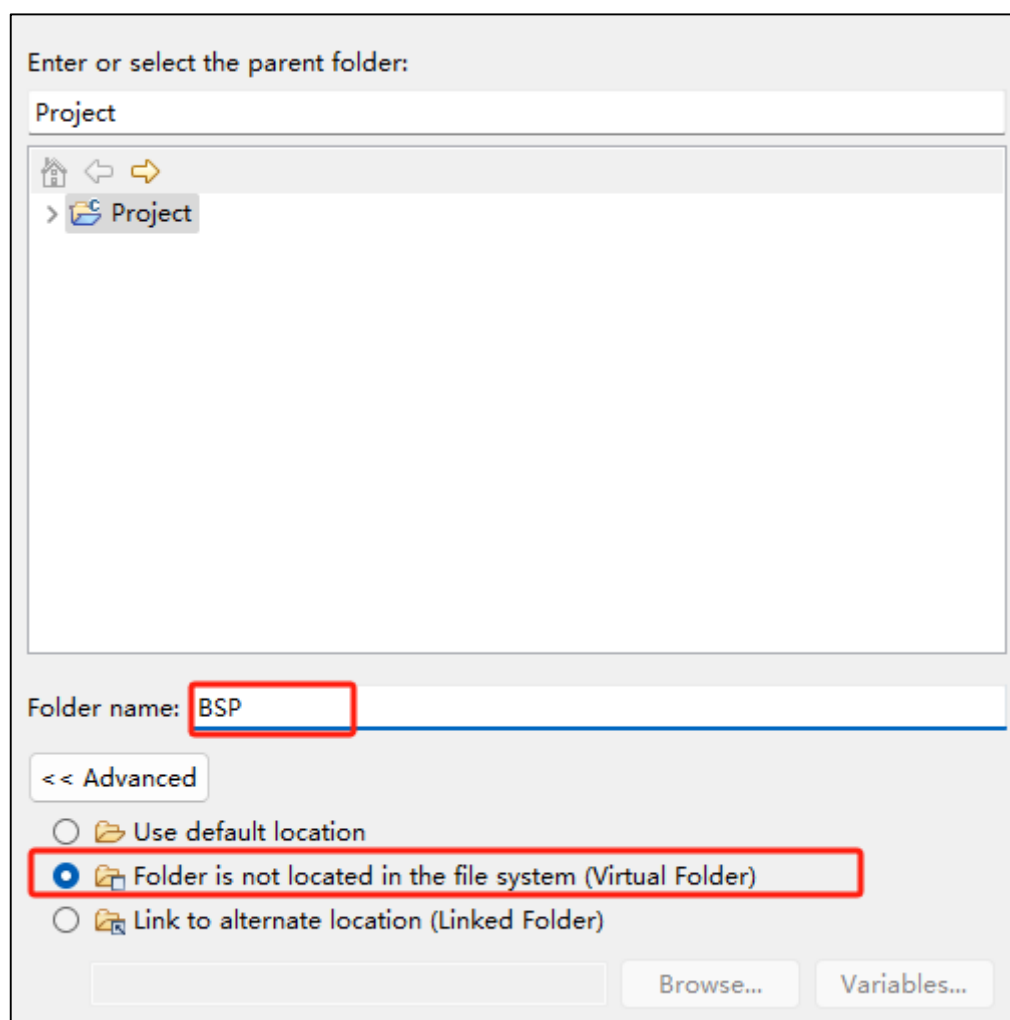
Toolchain path:

## 5.2 文件夹新增以及文件导入

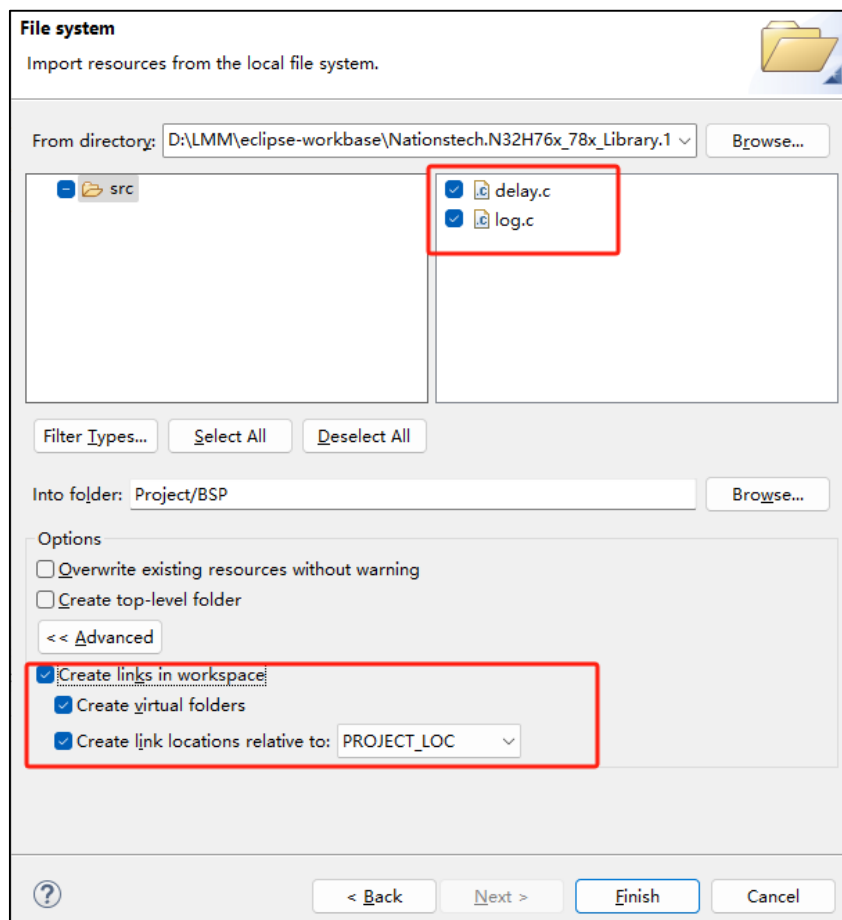
1. 右击工程名，选择 new->Folder



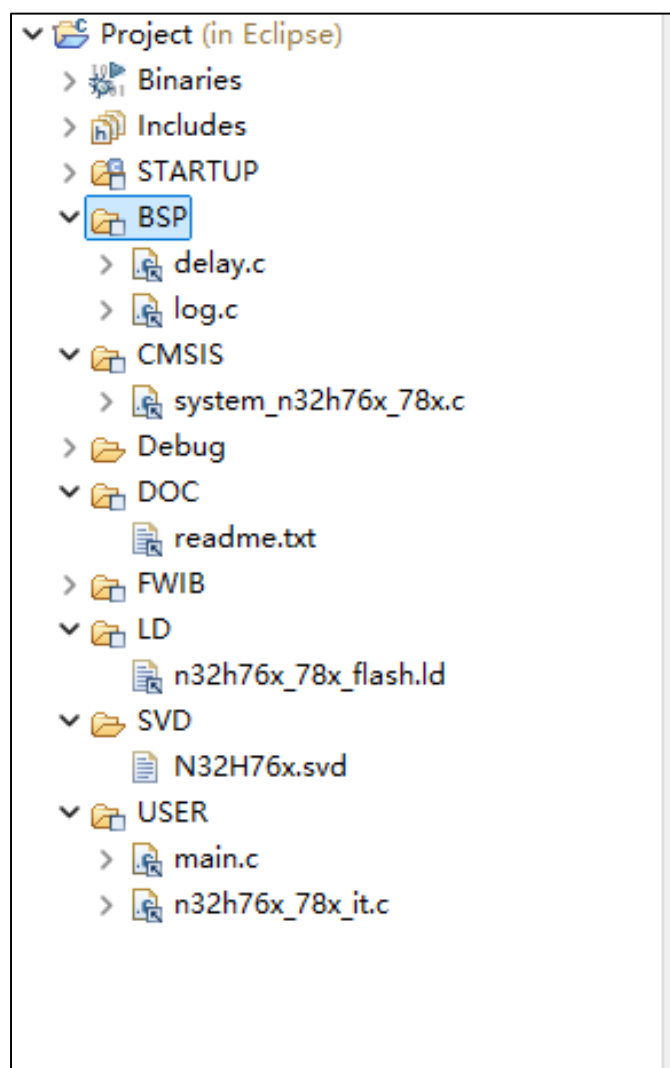
2. 分别建立 STARTUP, BSP, CMSIS, DOC, FWIB, LD, SVD, USER 的子虚拟文件夹；



3. 右击 BSP，选择 Import 选项，可向的 BSP 文件夹导入文件。同理，其他文件夹的导入方式类似，这里就不做重复介绍；



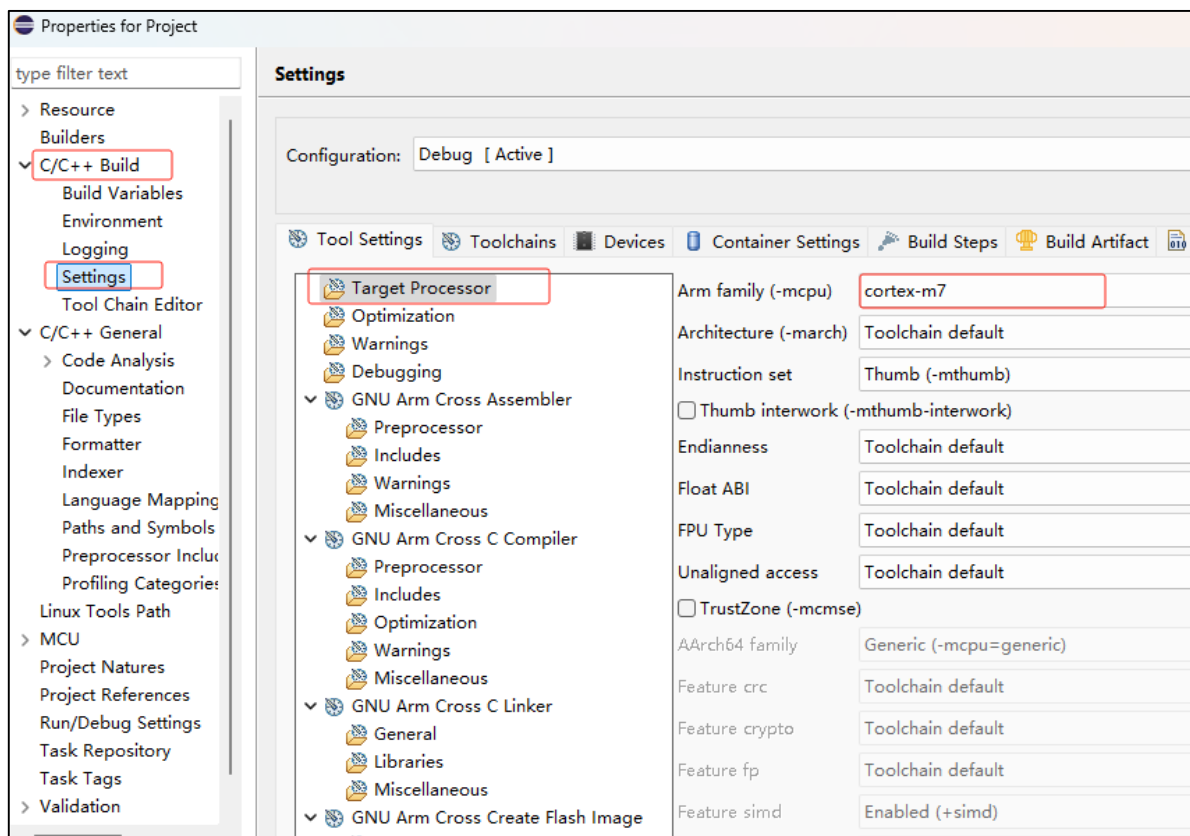
4.导入完成后，文件布局如下图所示；



## 5.3 工程配置

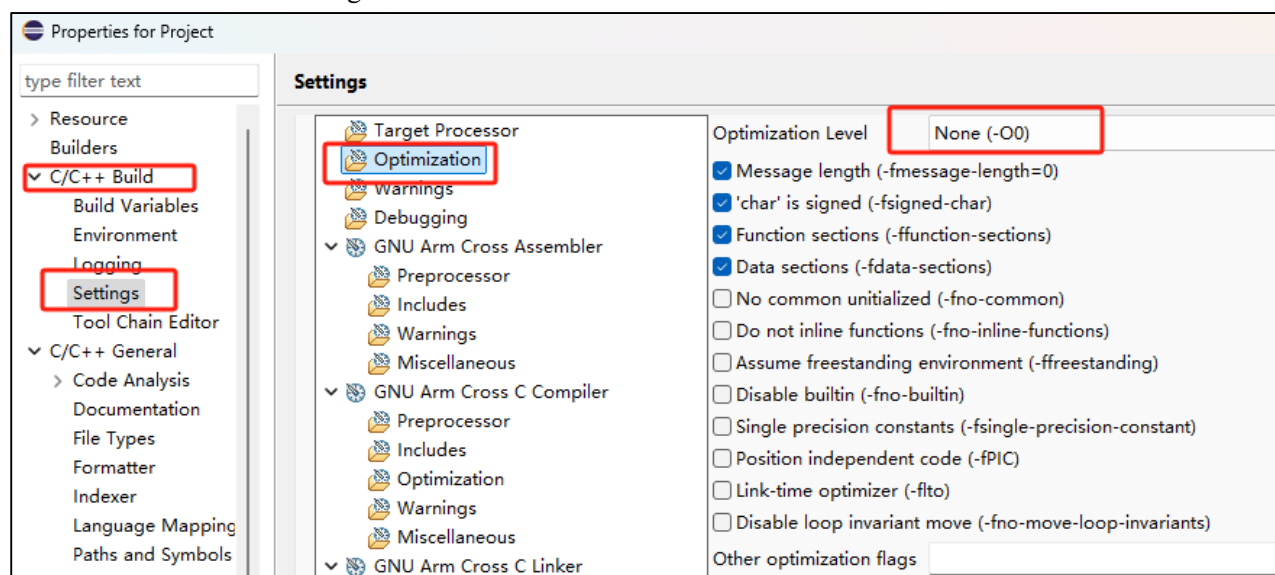
### 5.3.1 芯片选择

右击工程，选择工程属性 Properties 选项打开，在“C/C++ Build->Settings->Tool Settings->Target Processor”下配置如下：根据目标芯片的内核，选择 cortex-m3、cortex-m4、cortex-m0 或 cortex-m7。在本例中，选择 cortex-m7，如下图所示；



## 5.3.2 优化等级配置

在“C/C++ Build->Settings->Tool Settings->Optimization”选项中配置优化等级，可选-O0、-O1、-O2、-O3、-Os、-Ofast、-Og。

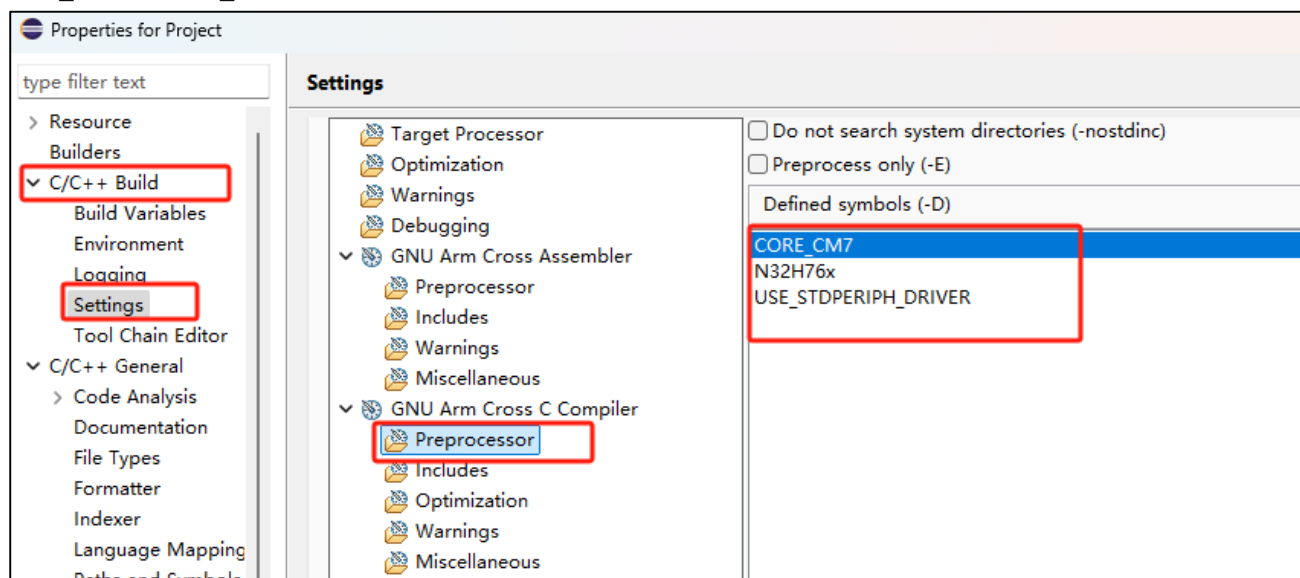




### 5.3.3 GNU Arm Cross C Compiler 宏配置

在“C/C++ Build->Settings->Tool Settings->GNU Arm Cross C Compiler”选项中配置 Cross C 编译选项。

本例中，在“Preprocessor->Defined symbols(-D)”选项中添加 CORE\_CM7，N32H76x 和 USE\_STDPERIPH\_DRIVER 预编译宏



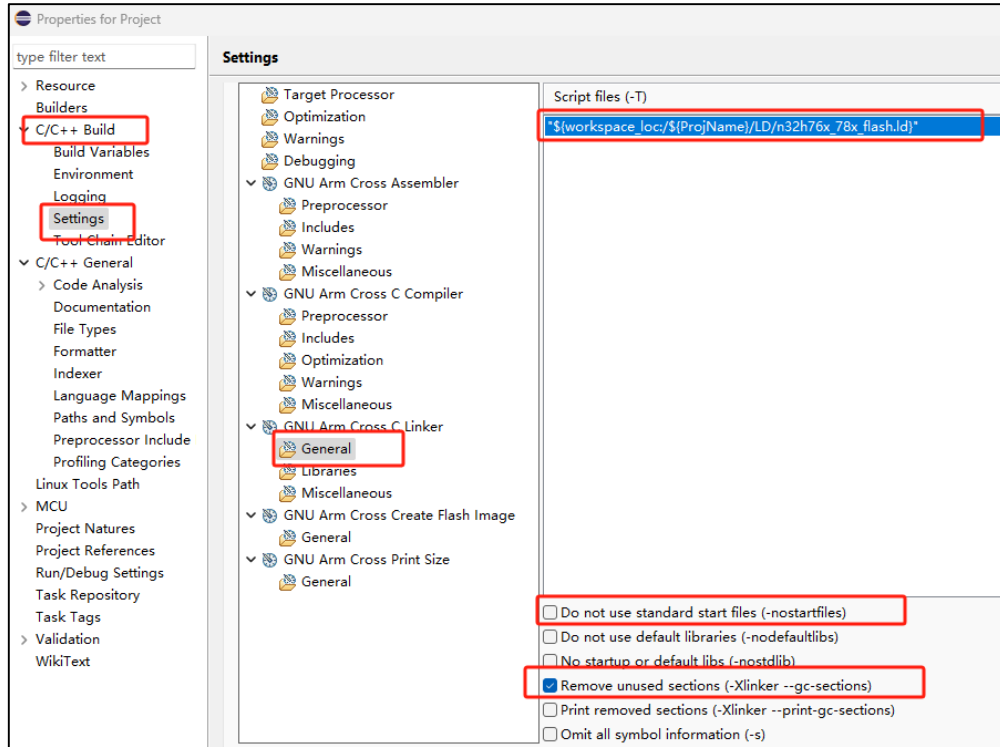
在“includes->Include paths”选项中添加工程所需的头文件路径。在本例中添加：

```

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

### 5.3.4 GNU Arm Cross C Linker 配置

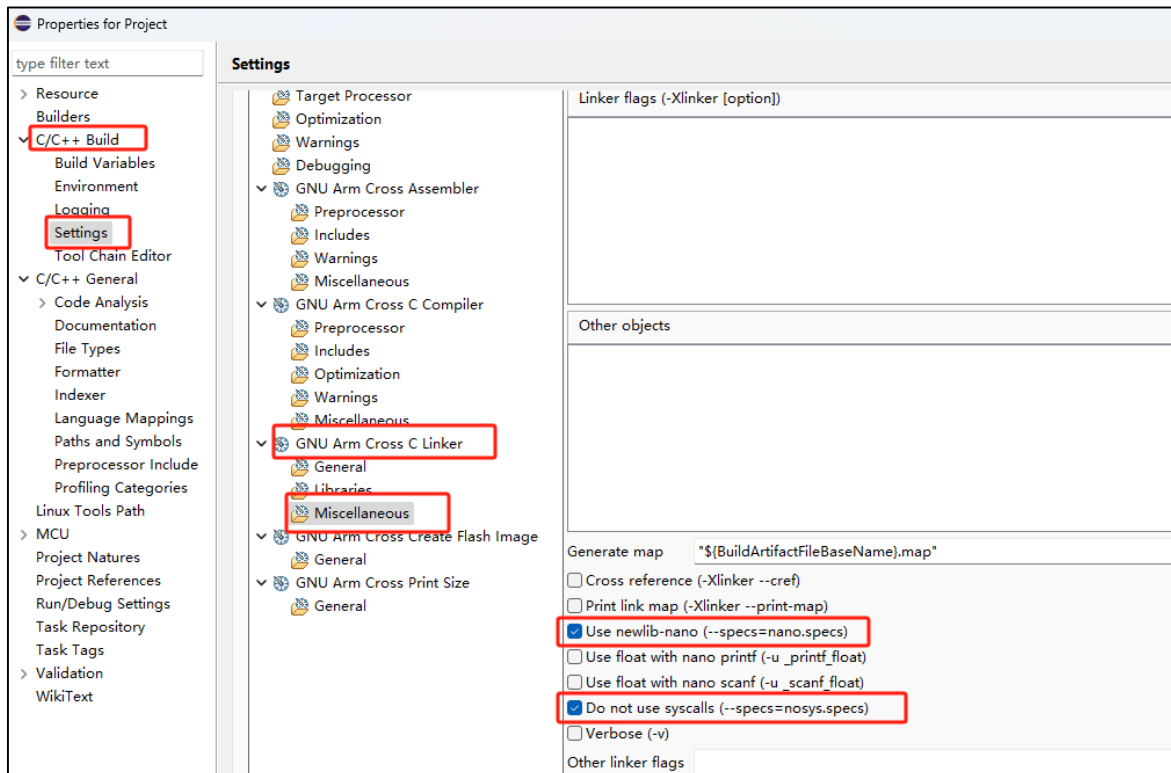
在“C/C++ Build->Settings->Tool Settings->GNU Arm Cross C Linker”配置 Cross C 链接选项。在“General->Script files”选项中添加： "\${workspace\_loc:\${ProjName}}/LD/n32h76x\_78x\_flash.ld" "



ld 脚本负责告诉链接器，编译完成的可执行文件如何配置内存。使用的 ld 脚本应该符合目标芯片的 FLASH 及 SRAM 大小及客户所需的内存配置。

注：需要取消勾选“Do not use standard start files”。

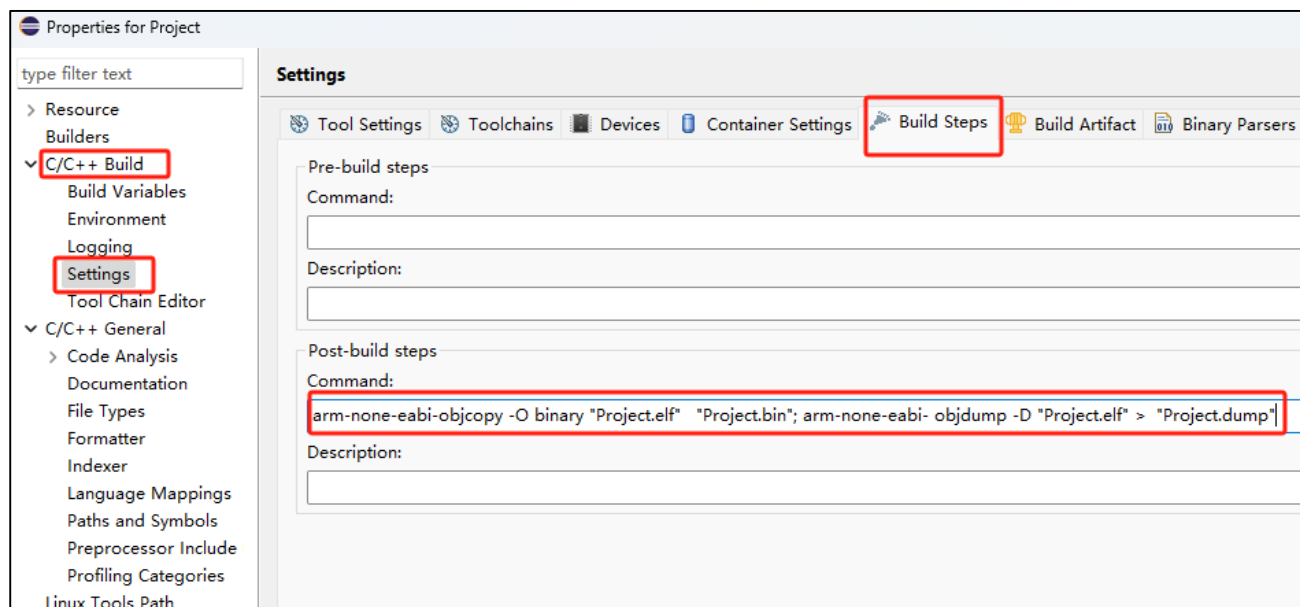
在 Miscellaneous 选项中，勾选 Use newlib-nano 及 Do not use syscalls。（可优化代码大小）



### 5.3.5 配置生成 bin 文件

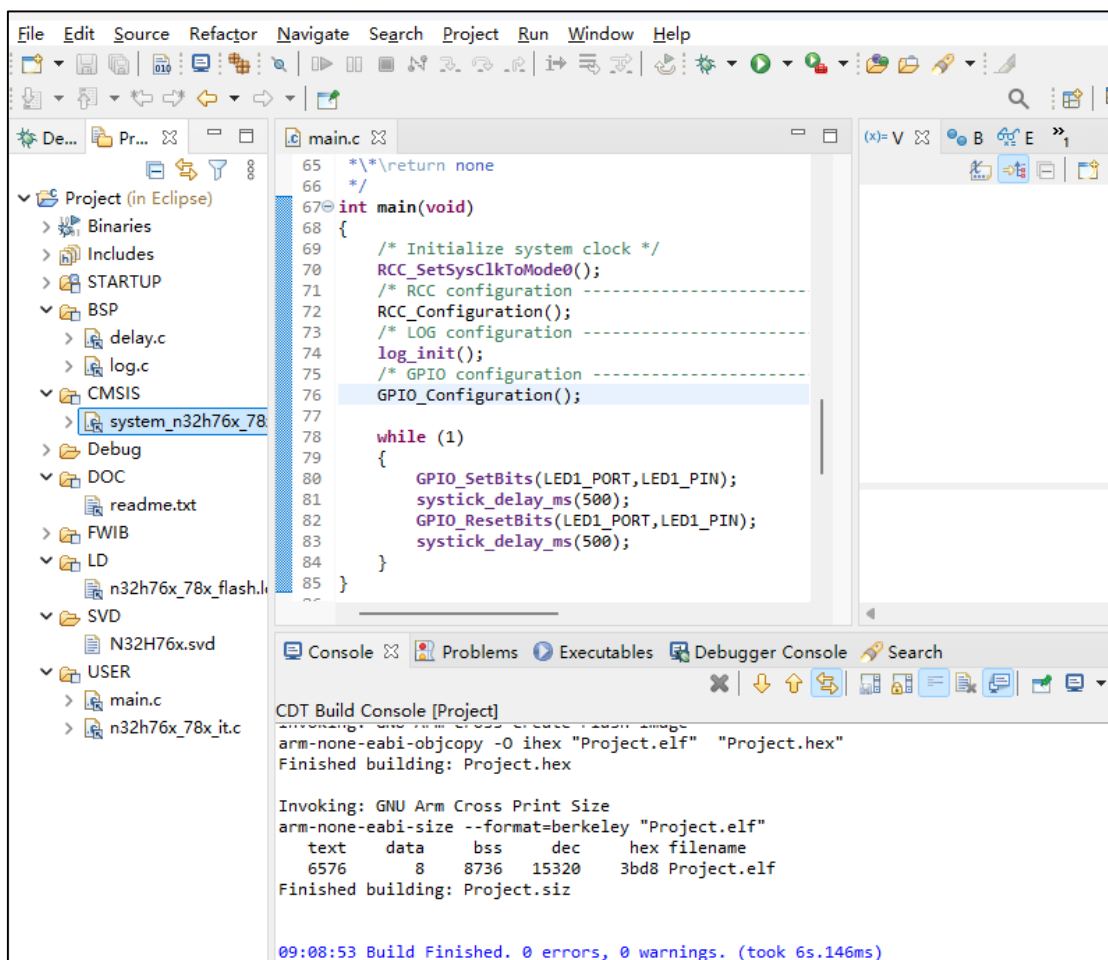
在“C/C++ Build->Settings-> Build Steps”可添加命令，生成 bin/hex 文件。

本例中添加：arm-none-eabi-objcopy -O binary "Project.elf" "Project.bin"; arm-none-eabi-objdump -D "Project.elf" > "Project.dump"



## 6. 编译

选择 Project->Build Project 可编译当前工程，每次编译之前需先保存当前工程再编译，否则编译的是上一次的工程。修改后,为了确保正确,请先 clean 工程后再 build。编译完成后，可见已生成对应的 elf、hex 及 bin 文件。



```

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 Compile Hex 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)

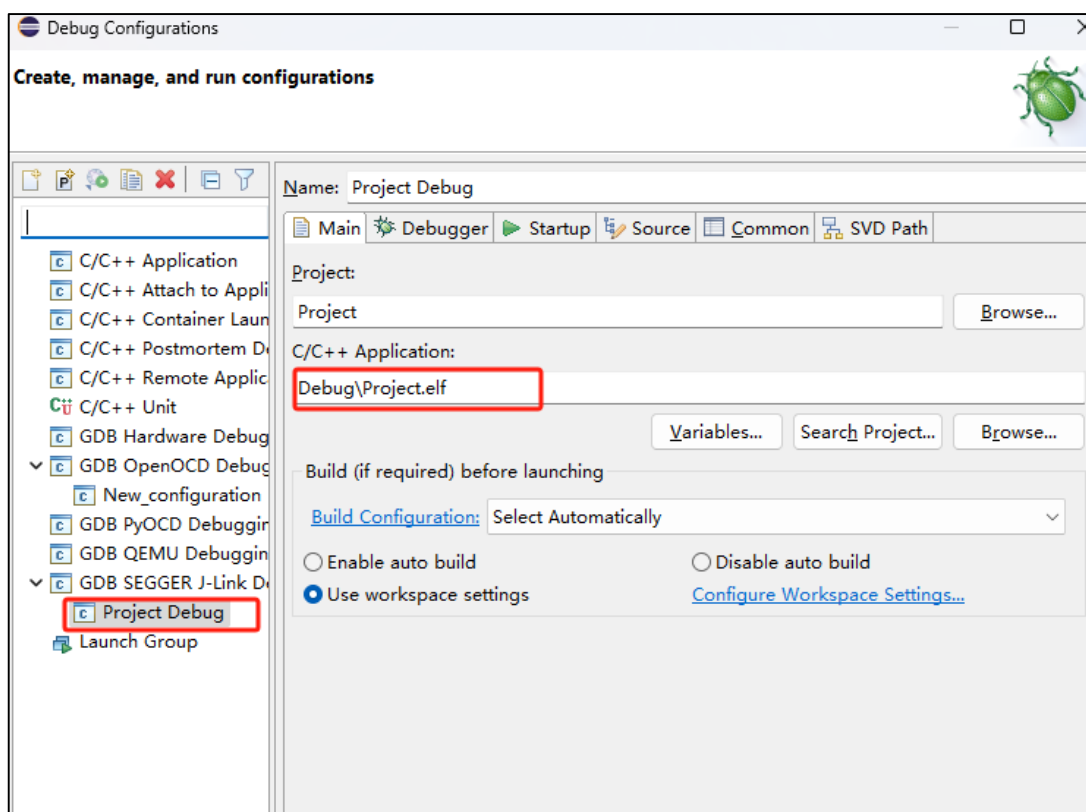
```

## 7.JLINK 下载及调试

在菜单栏中，点击 Run->Debug Configurations，进入 Debug 配置界面。使用 J-Link GDBServerCL 作为 GDB Server，使用 GCC 工具链中的 GDB 工具作为 GDB Client。双击 GDB SEGGER J-Link Debugging，新建一套 J-Link 的配置选项。

### 7.1 Main 选项卡

在 Main 选项卡中，选择当前工程，一般会添加当前工程下的 elf 文件。如果没有，可以点击 Browse，手动添加 elf 文件

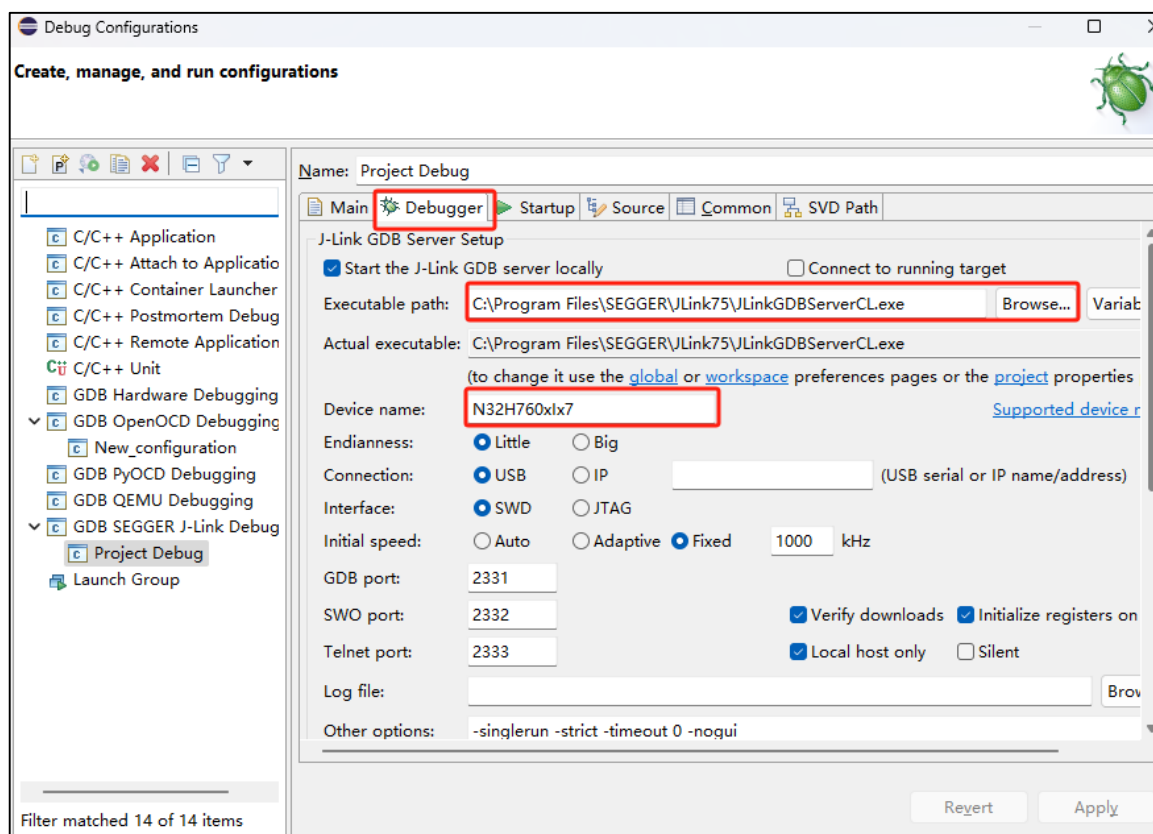


### 7.2 Debugger 选项卡

在 Debugger 选项卡中，填写目标芯片型号 Device name，本例中为 N32H760xIx7。

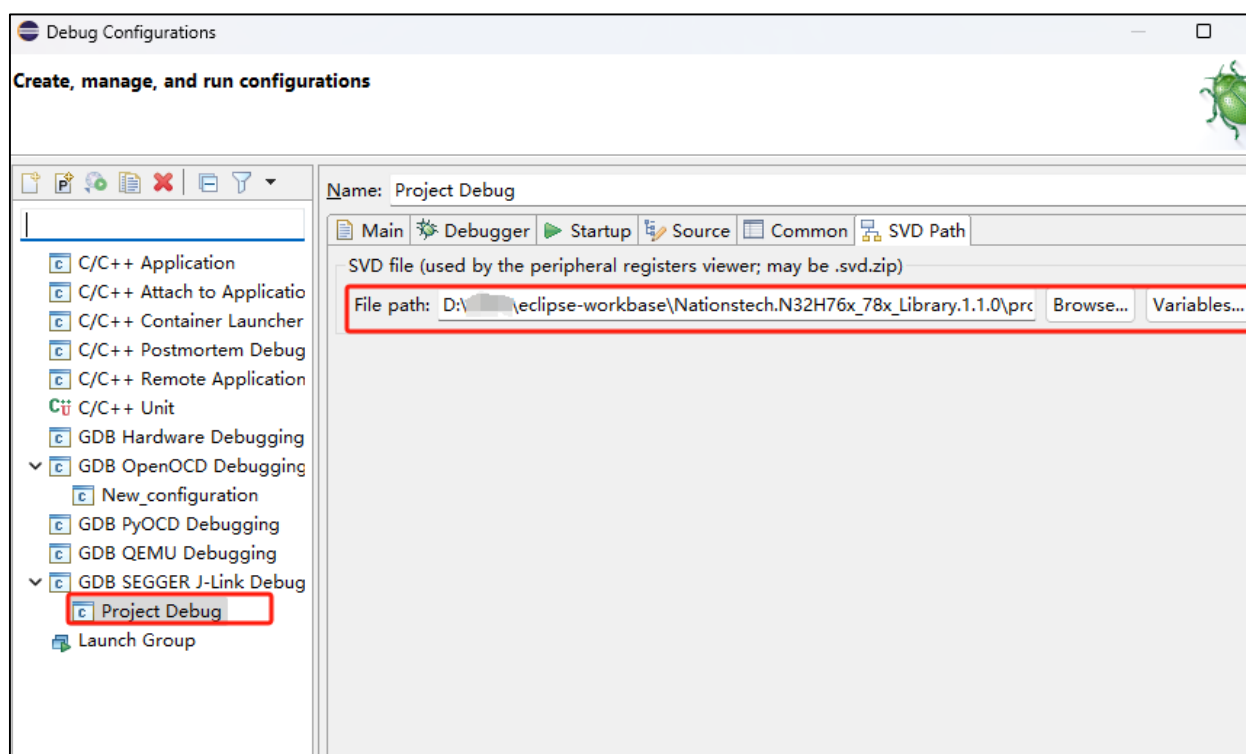
若在搭建 Eclipse 环境时已正确配置 J-Link 路径，这里将自动识别。如果之前没有正确配置，也可在 Executable path 栏，选中 J-Link GDBServerCL 的绝对路径。

**注意：**填写的芯片型号，在所配置的 J-Link 驱动中必须支持。



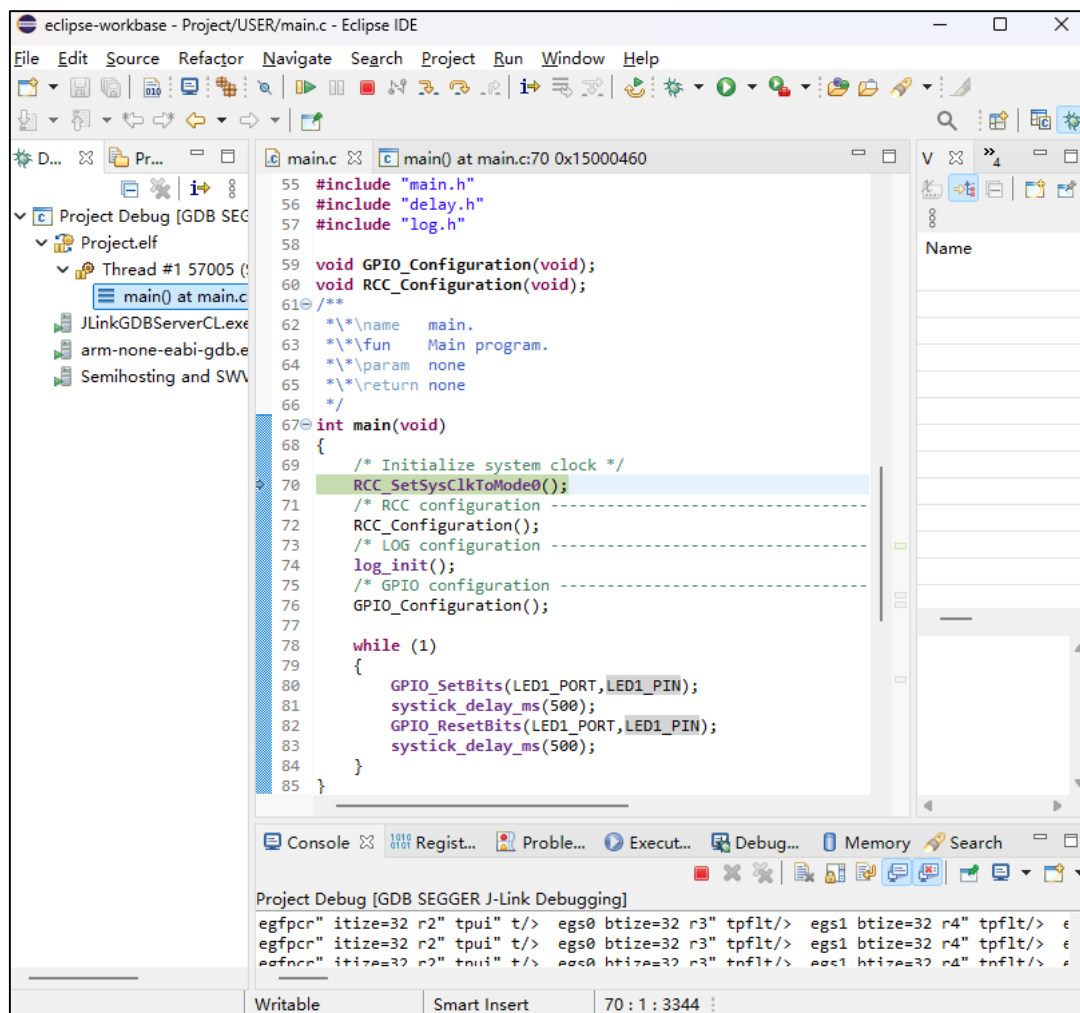
## 7.3 SVD Path 选项卡

在 SVD Path 选项卡，选择目标芯片所需的 SVD 文件。



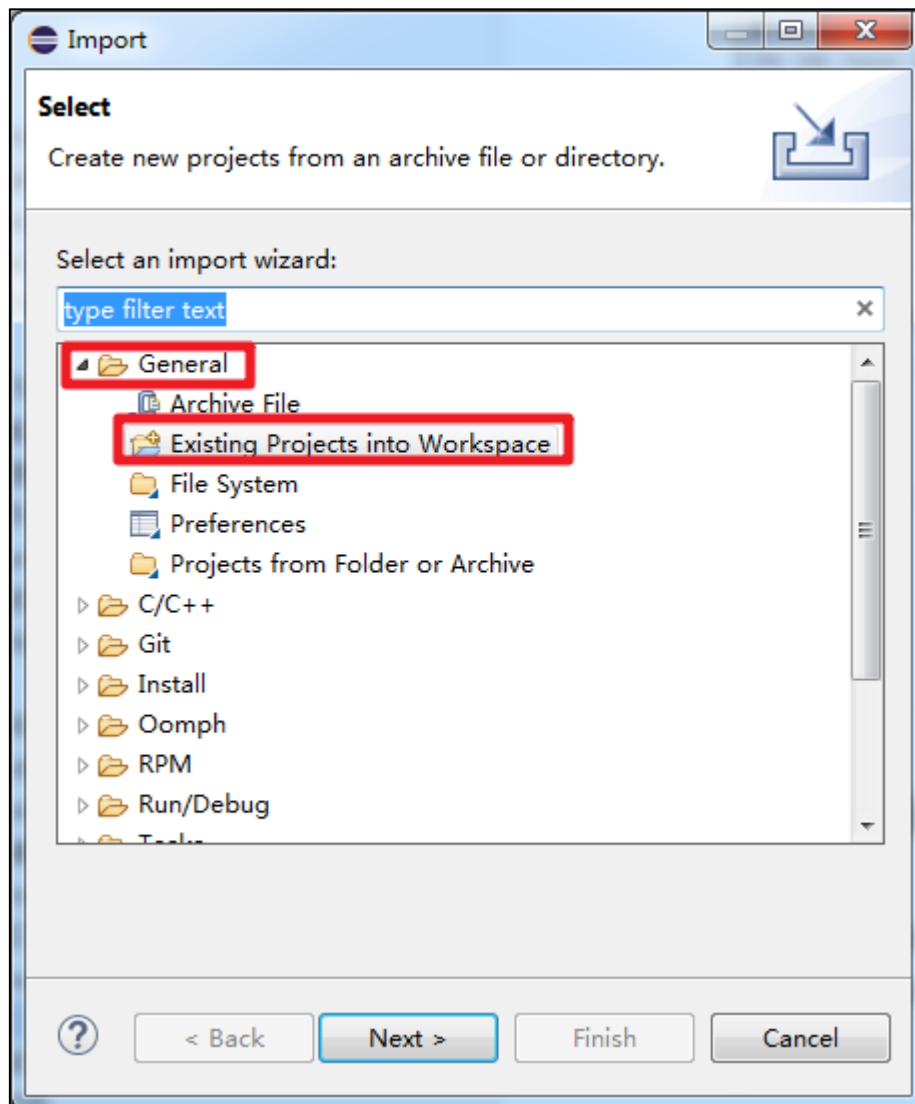
## 7.4 Debug 界面

Jlink 连接号设备，Debug Configurations 配置完成后，点击 Debug，进入 Debug 视图，然后切换到 DEBUG 视图，就可以正常调试了；



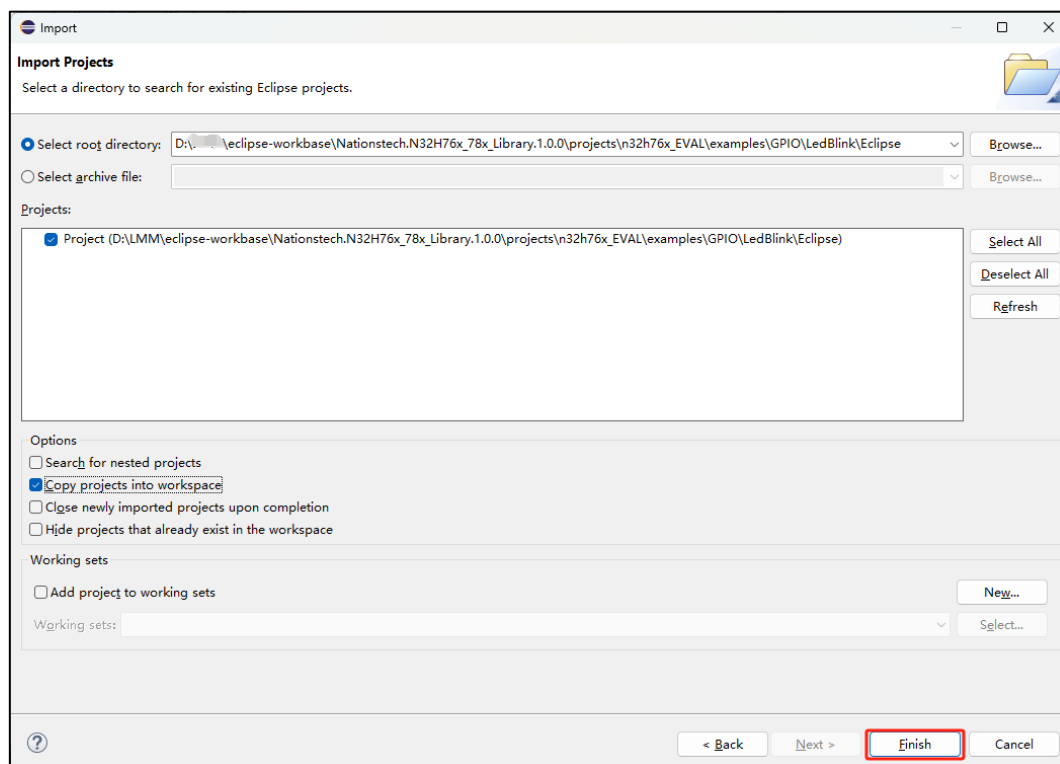
## 8. 导入现有的工程

在除新建工程外，也可直接导入已有 Eclipse 工程。在菜单栏中，点击 File->Import ，选择 General->Existing Projects into Workspace 导入已有工程，点击“Next”：





选择已有工程文件的路径，Eclipse 会识别出该路径下的所有工程，选择相应的工程，点击“Finish”，即可导入现有工程。



## 9.版本历史

日期	版本	修改
2025/08/18	V1.0.0	初始版本

## 10. 声明

国民技术股份有限公司（下称“国民技术”）对此文档拥有专属产权。依据中华人民共和国的法律、条约以及世界其他法域相适用的管辖，此文档及其中描述的国民技术产品（下称“产品”）为公司所有。

国民技术在此并未授予专利权、著作权、商标权或其他任何知识产权许可。所提到或引用的第三方名称或品牌（如有）仅用作区别之目的。

国民技术保留随时变更、订正、增强、修改和改良此文档的权利，恕不另行通知。请使用人在下单购买前联系国民技术获取此文档的最新版本。

国民技术竭力提供准确可信的资讯，但即便如此，并不推定国民技术对此文档准确性和可靠性承担责任。

使用此文档信息以及生成产品时，使用者应当进行合理的设计、编程并测试其功能性和安全性，国民技术不对任何因使用此文档或本产品而产生的任何直接、间接、意外、特殊、惩罚性或衍生性损害结果承担责任。

国民技术对于产品在系统或设备中的应用效果没有任何故意或保证，如有任何应用在其发生操作不当或故障情况下，有可能致使人员伤亡、人身伤害或严重财产损失，则此类应用被视为“不安全使用”。

不安全使用包括但不限于：外科手术设备、原子能控制仪器、飞机或宇宙飞船仪器、所有类型的安全装置以及其他旨在支持或维持生命的应用。

所有不安全使用的风险应由使用人承担，同时使用人应使国民技术免于因为这类不安全使用而导致被诉、支付费用、发生损害或承担责任时的赔偿。

对于此文档和产品的任何明示、默示之保证，包括但不限于适销性、特定用途适用性和不侵权的保证责任，国民技术可在法律允许范围内进行免责。

未经明确许可，任何人不得以任何理由对此文档的全部或部分进行使用、复制、修改、抄录和传播。