

GigaDevice Semiconductor Inc.

**GD32 RISC-V MCU Windows 系统下 Eclipse
开发教程**

应用笔记

AN067

目录

目录.....	2
图索引.....	4
表索引.....	6
1. 简介.....	7
2. 开发环境.....	8
3. 工程开发.....	9
3.1. 新建工程.....	9
3.2. 新建工程文件夹并添加文件.....	12
3.2.1. 手动建立文件夹及添加文件.....	12
3.2.2. “Refresh”方式建立文件夹及添加文件.....	24
3.3. 工程配置.....	27
3.3.1. Target Processor 配置.....	27
3.3.2. Optimization 配置.....	28
3.3.3. GNU RISC-V Cross Assembler 配置.....	29
3.3.4. GNU RISC-V Cross C Compiler 配置.....	30
3.3.5. GNU RISC-V Cross C Linker 配置.....	32
3.3.6. Build Steps 配置-生成 bin 文件.....	34
3.4. 编译工程.....	35
3.5. 使用 J-Link 下载及调试工程.....	36
3.5.1. Debug 配置界面.....	36
3.5.2. Main 选项卡.....	37
3.5.3. Debugger 选项卡.....	38
3.5.4. Startup 选项卡.....	39
3.5.5. SVD Path 选项卡.....	40
3.6. 使用 GD-Link 下载及调试工程.....	40
3.6.1. Debug 配置界面.....	40
3.6.2. Main 选项卡.....	41
3.6.3. Debugger 选项卡.....	42
3.6.4. SVD Path 选项卡.....	43
3.7. Debug 界面.....	44
3.7.1. 工具栏介绍.....	45
3.7.2. Registers 窗口.....	46
3.7.3. Peripherals 窗口.....	47
3.7.4. Memory 窗口.....	49
3.7.5. Expressions 窗口.....	50

3.7.6.	反汇编窗口	51
3.7.7.	退出 Debug 视图	52
4.	导入现有工程	53
5.	在 RAM 中调试	55
6.	使用 printf 打印	57
6.1.	使用步骤	57
6.2.	打印浮点数据配置	57
7.	版本历史	59

图索引

图 3-1. 新建 RISC-V C 工程	9
图 3-2. 选择 C Managed Build	10
图 3-3. 新建 RISC-V 工程名称及选择工程存放路径	11
图 3-4. 选择 RISC-V 交叉编译工具链	12
图 3-5. 工程视图	12
图 3-6. 新建工程文件夹	13
图 3-7. 建立虚拟子文件夹	13
图 3-8. RISC-V 工程视图	14
图 3-9. 添加文件	15
图 3-10. 选择 Import 文件	15
图 3-11. Import Application 文件夹内文件	16
图 3-12. Import Peripherals 文件夹文件	17
图 3-13. Import Peripheral-Source 文件夹文件	18
图 3-14. Import RISC-V-env_Eclipse 文件夹文件	19
图 3-15. Import RISC-V-drivers 文件夹文件	20
图 3-16. Import RISC-V-stubs 文件夹文件	21
图 3-17. Import Doc 文件夹文件	22
图 3-18. Import Utilities 文件夹文件	23
图 3-19. 最终 RISC-V 工程视图	24
图 3-20. 工程文件夹结构	24
图 3-21. Refresh 工程	25
图 3-22. Eclipse IDE 内工程结构	26
图 3-23. 工程属性配置	27
图 3-24. RISC-V 工程 Target Processor 配置	28
图 3-25. Optimization 配置	29
图 3-26. GNU RISC-V Cross C Assembler -> Includes 配置	30
图 3-27. GNU RISC-V Cross C Compiler -> Preprocessor 配置	31
图 3-28. GNU RISC-V Cross C Compiler -> Includes 配置	32
图 3-29. GNU RISC-V Cross C Linker -> General 配置	33
图 3-30. GNU RISC-V Cross C Linker -> Miscellaneous 配置	34
图 3-31. Build Steps 配置	35
图 3-32. Build 工程	35
图 3-33. Build RISC-V 工程完成	36
图 3-34. 进入 Debug Configurations 界面	37
图 3-35. GDB SEGGER J-Link Debugging-Main 选项卡	38
图 3-36. GDB SEGGER J-Link Debugging-Debugger 选项卡	39
图 3-37. GDB SEGGER J-Link Debugging-Startup 选项卡	40
图 3-38. GDB SEGGER J-Link Debugging-SVD Path 选项卡	40
图 3-39. 进入 Debug Configurations 界面	41
图 3-40. GDB OpenOCD Debugging-Main 选项卡	42

图 3-41. GDB OpenOCD Debugging-Debugger 选项卡	43
图 3-42. GDB OpenOCD Debugging-SVD Path 选项卡.....	43
图 3-43. 进入 Debug 视图-1.....	44
图 3-44. 进入 Debug 视图-2.....	44
图 3-45. Debug 视图.....	45
图 3-46. 打开 Registers 窗口.....	46
图 3-47. Registers 窗口.....	47
图 3-48. 打开 Peripherals 窗口.....	48
图 3-49. Peripherals 窗口.....	49
图 3-50. 打开 Memory 窗口	50
图 3-51. Memory 窗口	50
图 3-52. 打开 Expressions 窗口.....	51
图 3-53. Expressions 窗口	51
图 3-54. 打开反汇编窗口	51
图 3-55. 反汇编窗口	52
图 3-56. 退出 Debug 视图.....	52
图 4-1. 导入现有工程-1.....	53
图 4-2. 导入现有工程-2.....	54
图 5-1. 在 RAM 中调试时 Id 文件 memory map.....	55
图 5-2. 在 RAM 中调试时 Debug Configurations	55
图 5-3. 在 RAM 中调试时 Debug 视图.....	56
图 6-1. 打印浮点数据配置.....	58

表索引

表 7-1. 版本历史	59
-------------------	----

1. 简介

本文介绍了如何使用Eclipse进行GD32 MCU开发。适用于GD32 RISC-V内核系列MCU。

2. 开发环境

- 开发板：GD32 MCU开发板
- 调试器：J-Link V10或GD-Link
- 操作系统：WIN7/WIN10 64-bit OS
- IDE：eclipse-embedcpp-2021-03-R-win32-x86_64
- 交叉编译链：xpack-riscv-none-embed-gcc-10.1.0-1.1-win32-x64
- 编译工具：gnu-mcu-eclipse-windows-build-tools-2.12-20190422-1053-win64
- GDB服务器：OpenOCD / J-Link GDB Server CL V7.11a

3. 工程开发

3.1. 新建工程

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

图 3-1. 新建 RISC-V C 工程

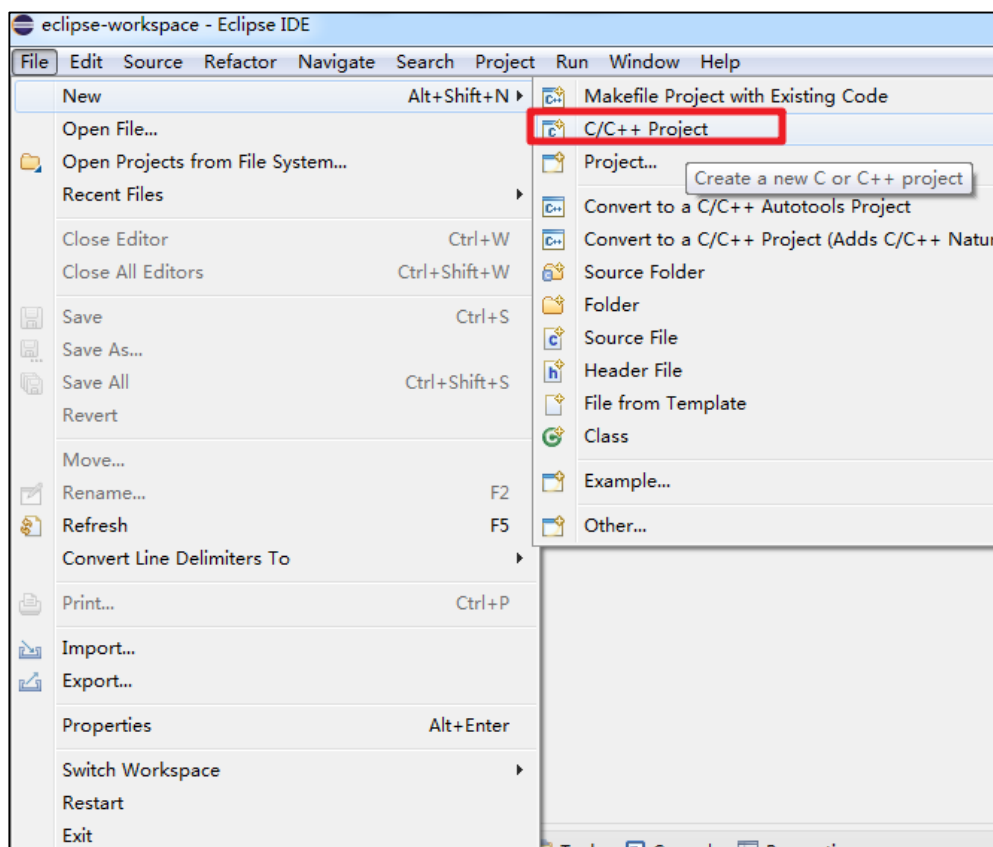
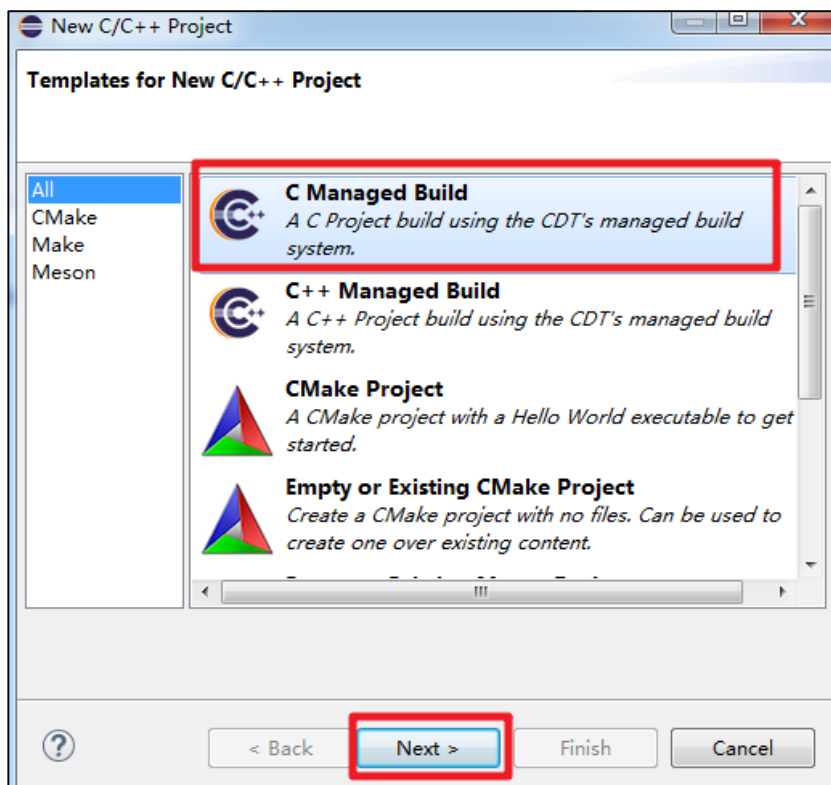
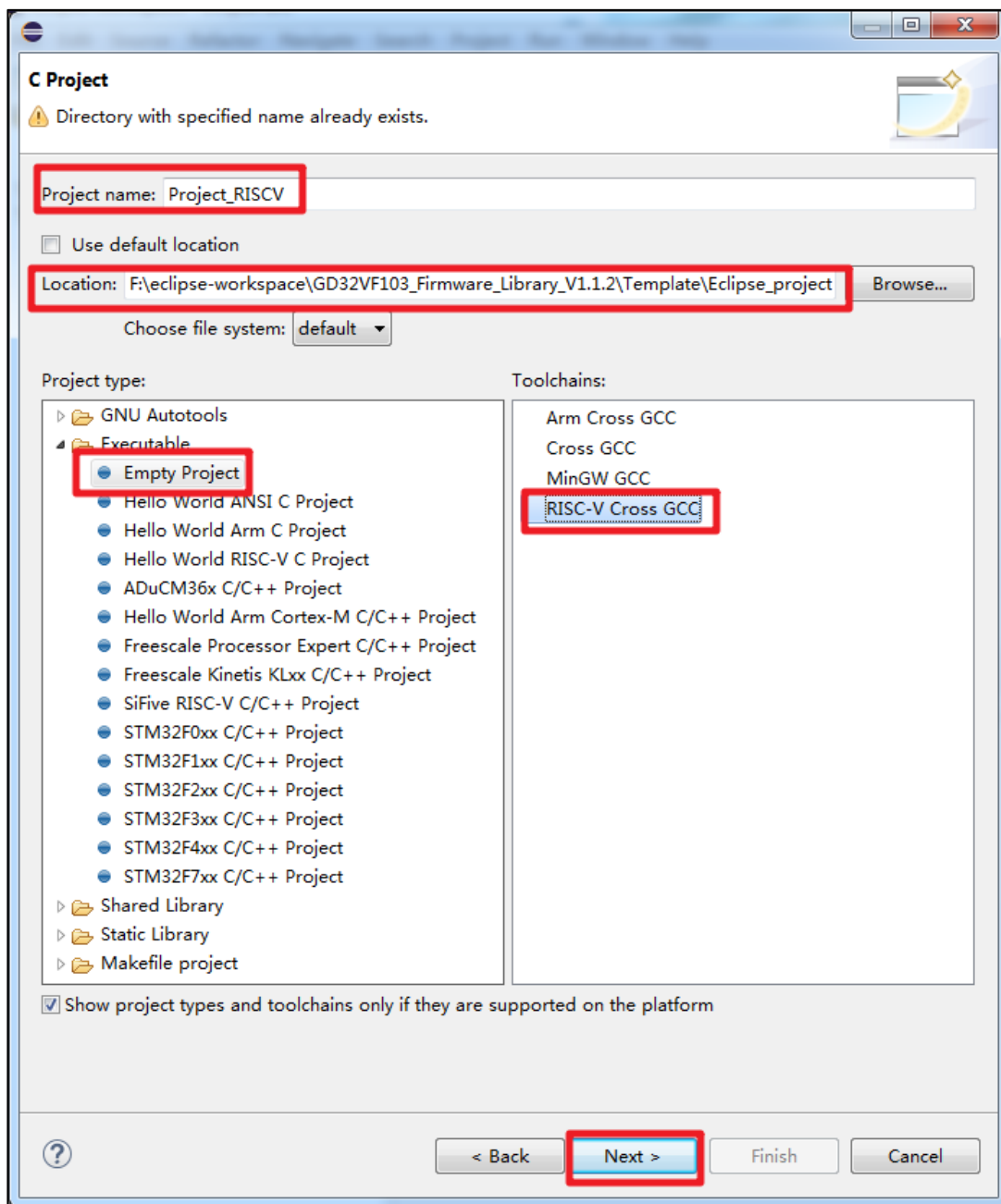


图 3-2. 选择 C Managed Build



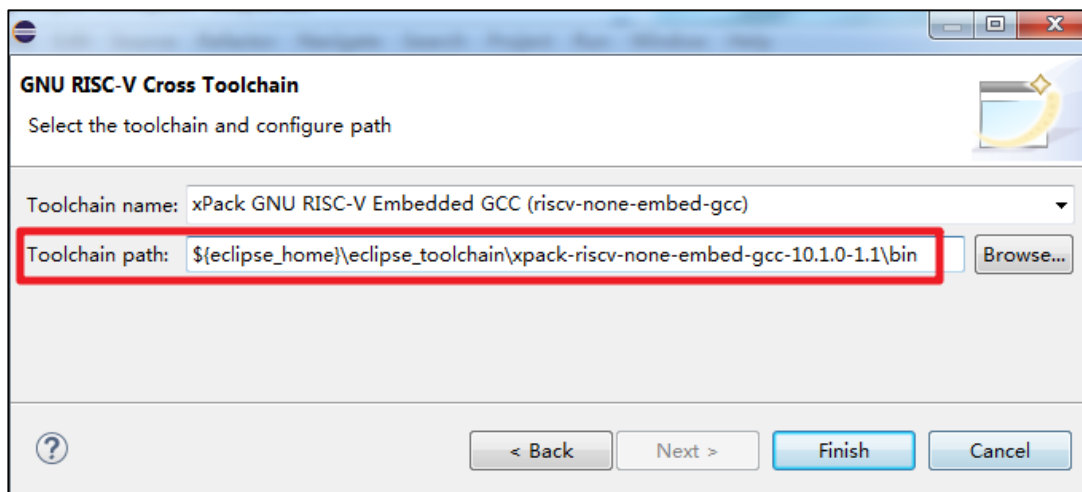
输入 Project name, 配置工程类型, 为了方便建议将工程放到 FW 目录下。编译链选择为 RISC-V Cross GCC。

图 3-3. 新建 RISC-V 工程名称及选择工程存放路径



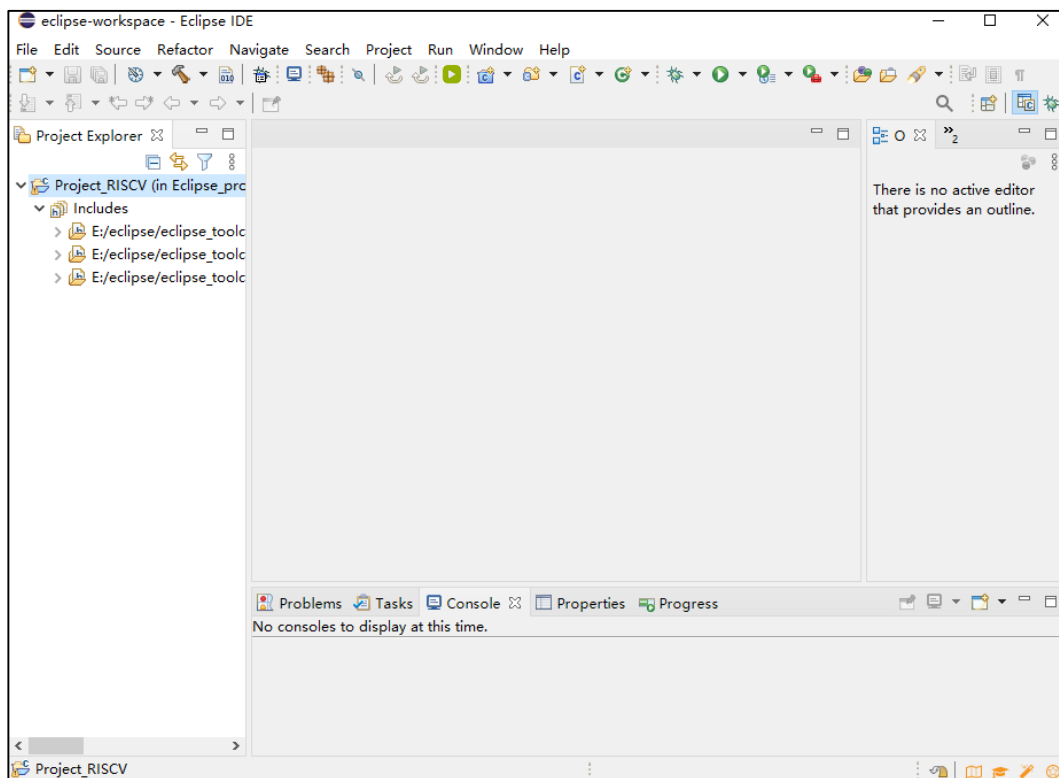
若Eclipse IDE已正确设置了RISC-V Toolchains Path，这里将会自动选择路径。若Eclipse IDE未设置RISC-V Toolchains Path，也可在这里选择到RISC-V Toolchains绝对路径。

图 3-4. 选择 RISC-V 交叉编译工具链



所示在跳出的界面选择“Next”最后选择“Finish”，直到显示界面如[图3-5. 工程视图](#)所示。至此，完成Project的建立。

图 3-5. 工程视图

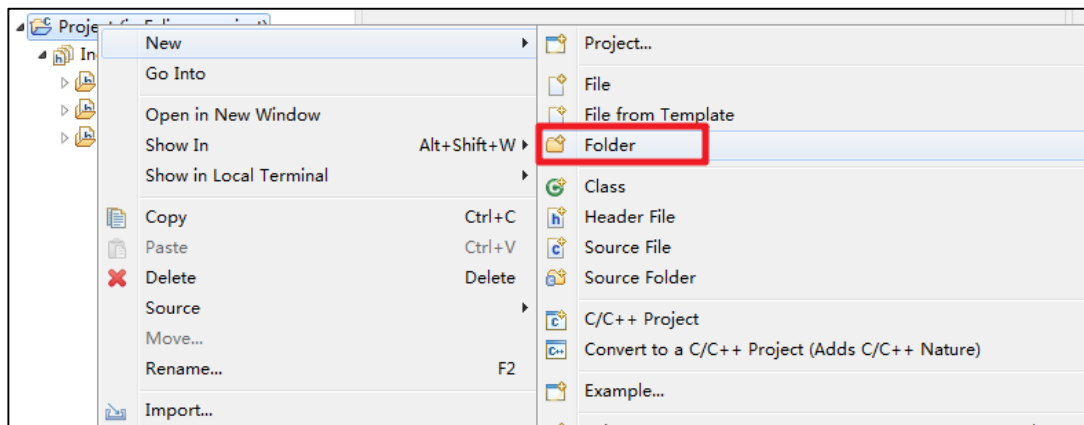


3.2. 新建工程文件夹并添加文件

3.2.1. 手动建立文件夹及添加文件

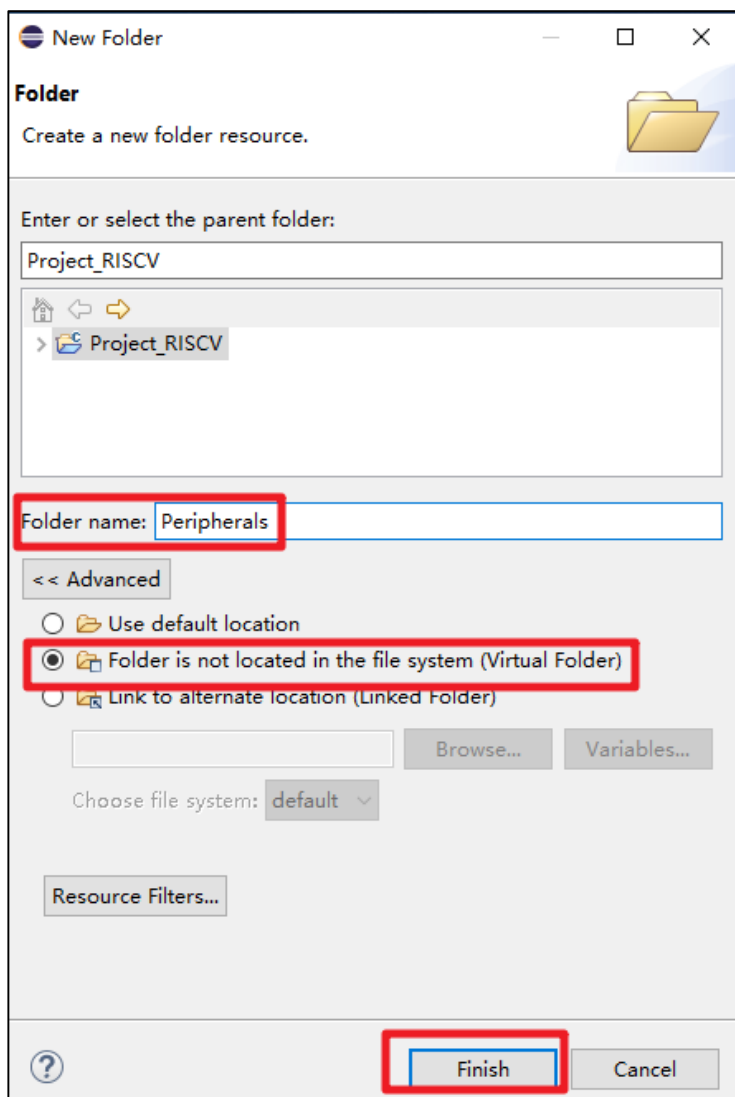
右击工程名，选择new->Folder。

图 3-6. 新建工程文件夹



建立虚拟文件夹Peripherals。

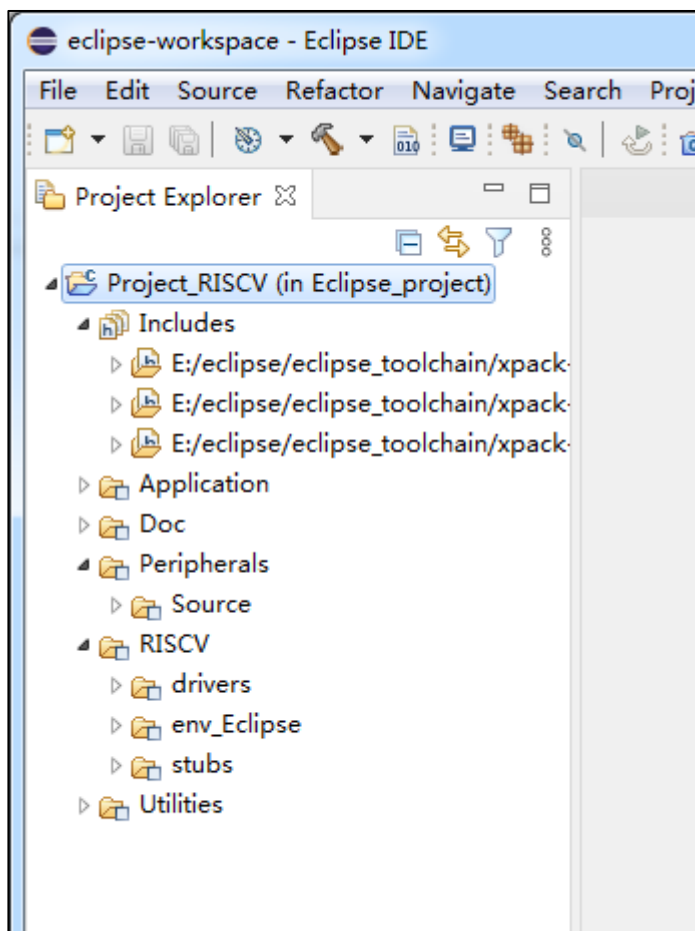
图 3-7. 建立虚拟子文件夹



同样方法建立Application、Doc和Utilities文件夹。并在Peripherals文件夹内建立Source子文件

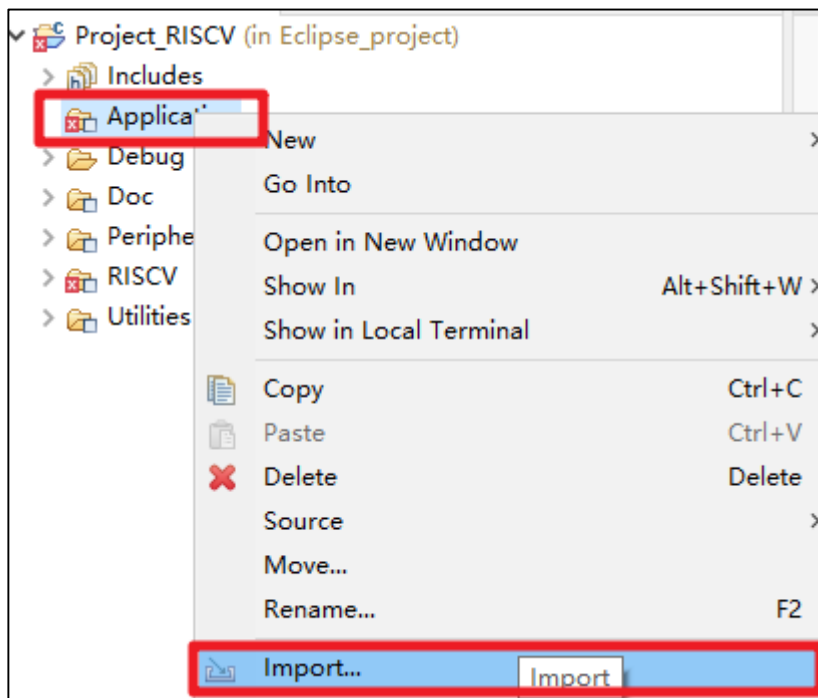
夹。在RISCV文件夹内建立drivers、env_Eclipse和stubs子文件夹。

图 3-8. RISC-V 工程视图



右击Application，选择Import选项，可导入文件。

图 3-9. 添加文件



Import选择File System。选择需要导入的文件的的路径，勾选需要导入的文件。

图 3-10. 选择 Import 文件

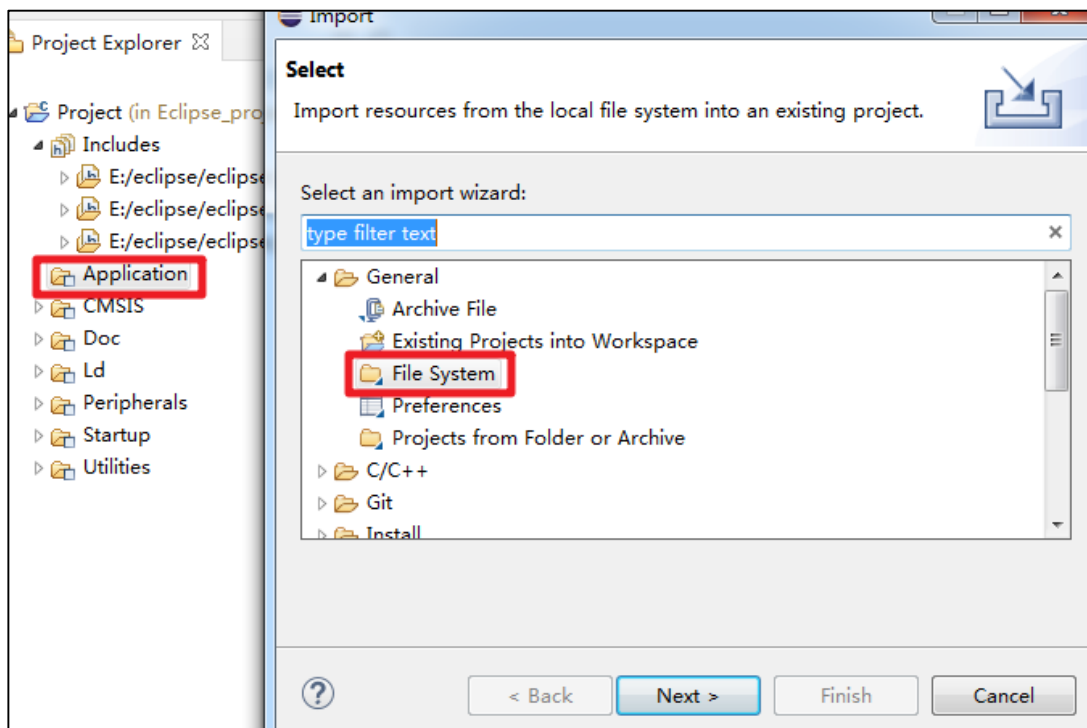
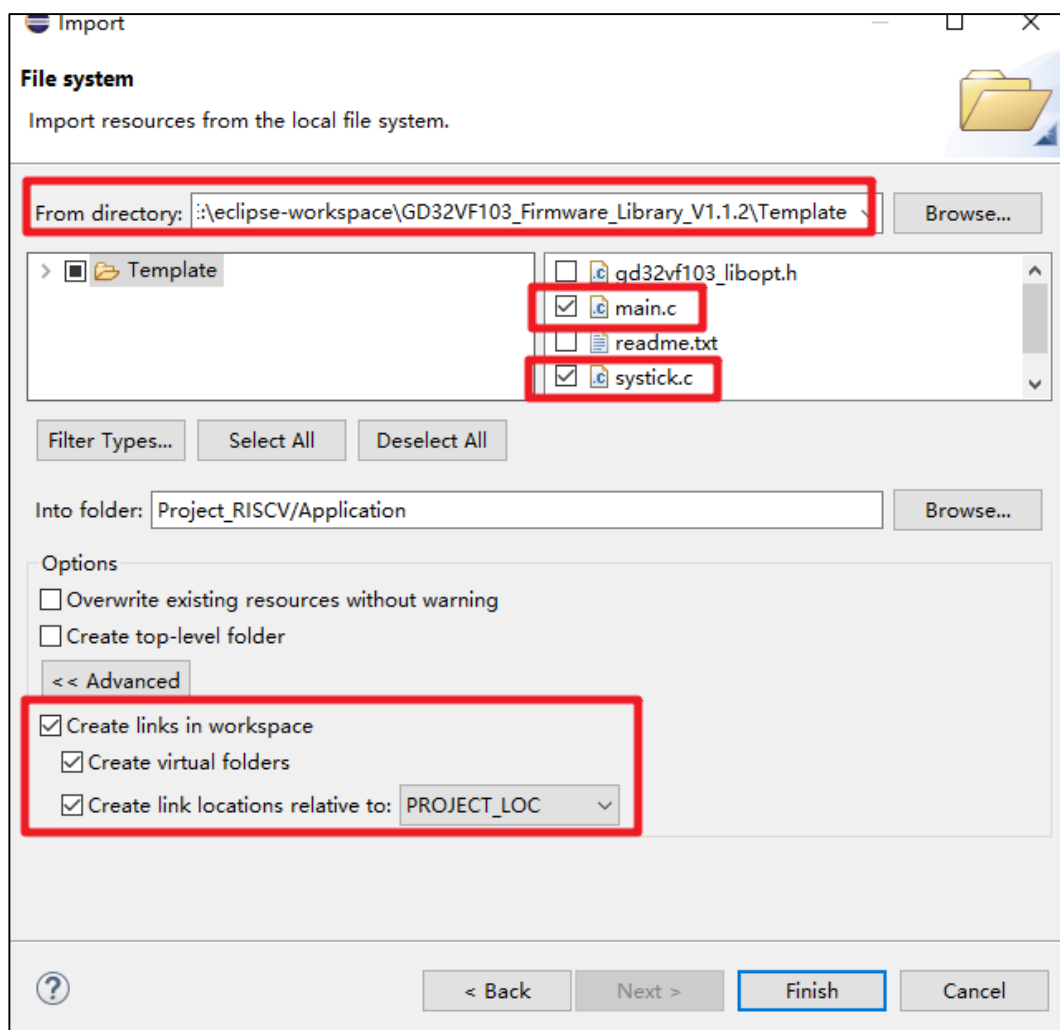


图 3-11. Import Application 文件夹内文件



同理，将所需文件导入Peripherals、RISCV、Doc和Utilities文件夹。

图 3-12. Import Peripherals 文件夹文件

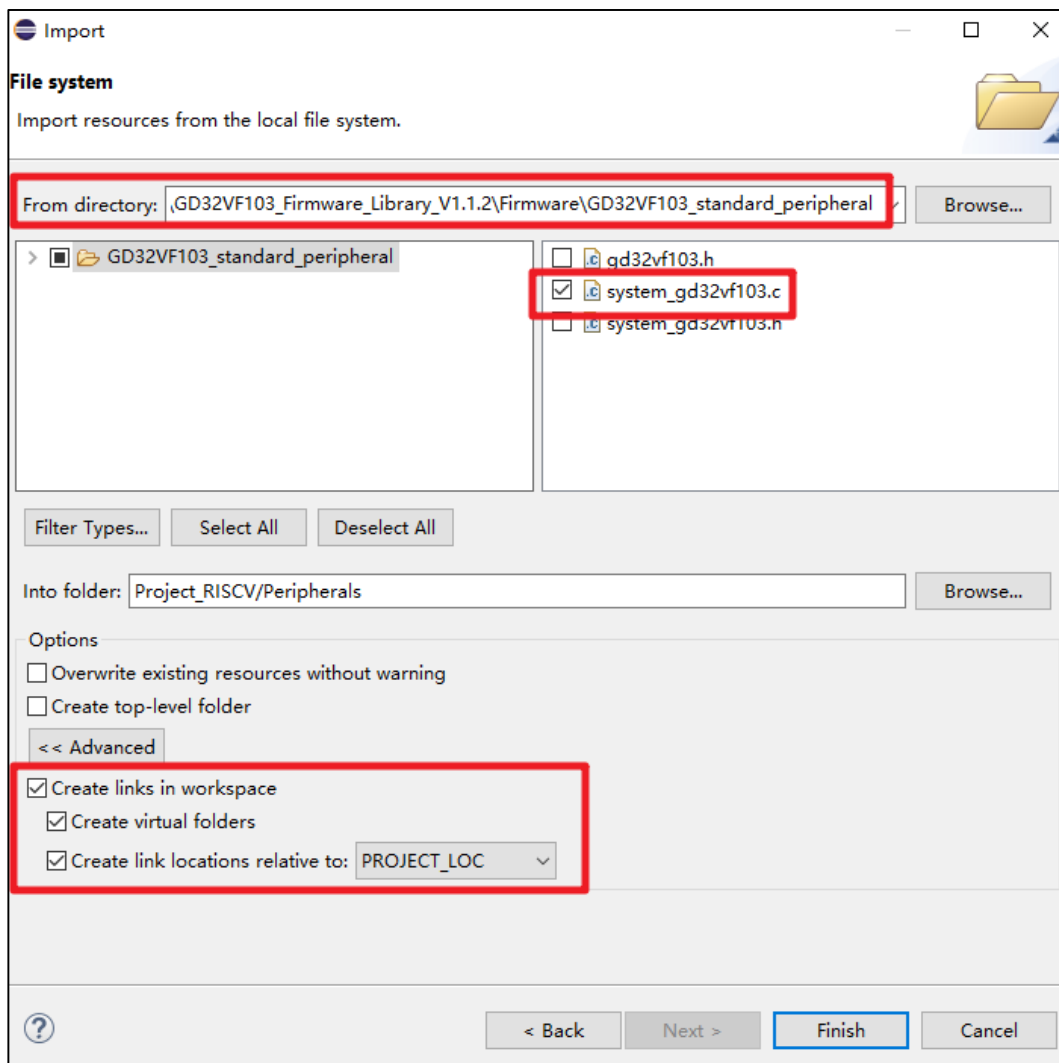


图 3-13. Import Peripheral-Source 文件夹文件

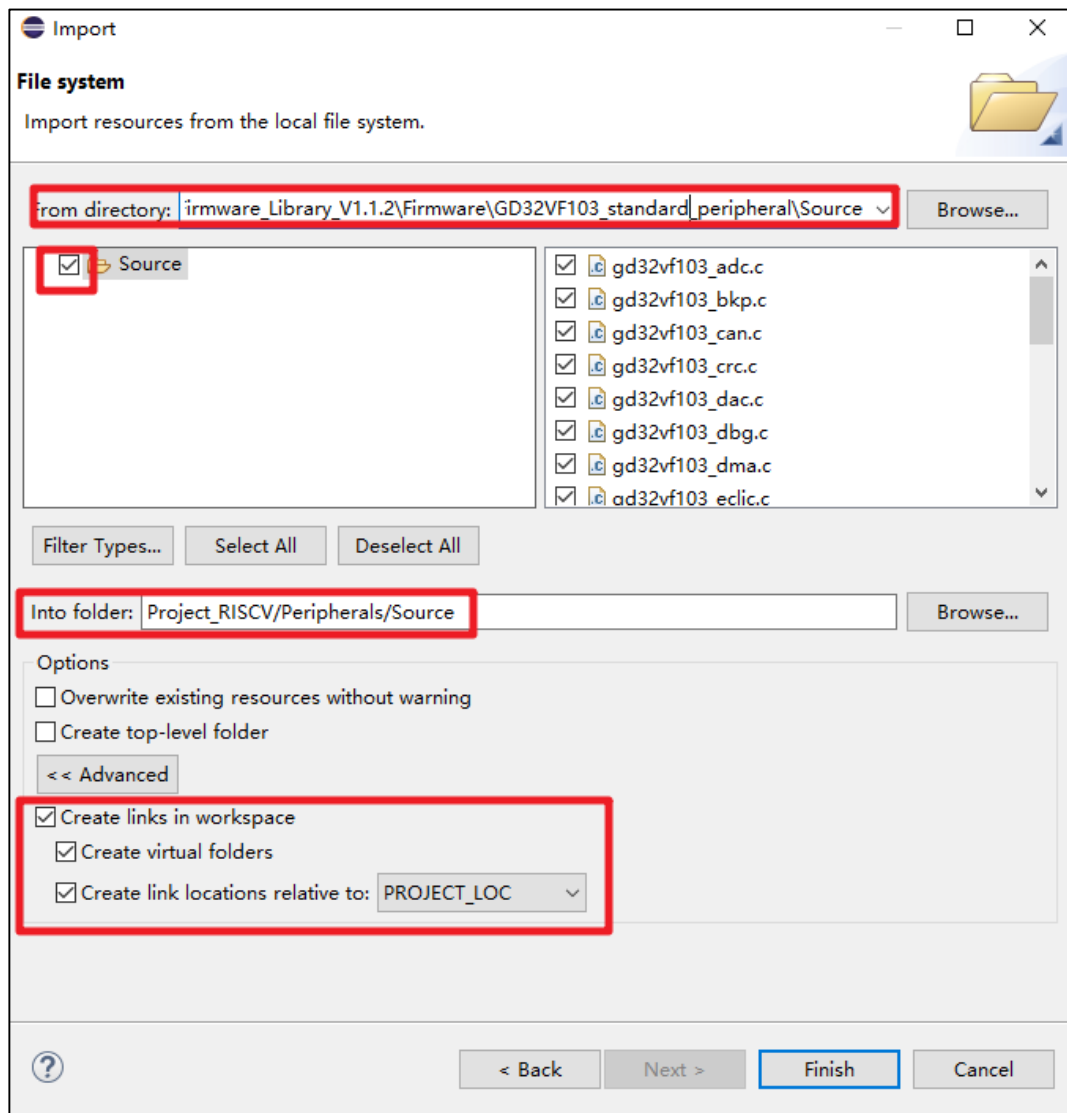


图 3-14. Import RISCV-env_Eclipse 文件夹文件

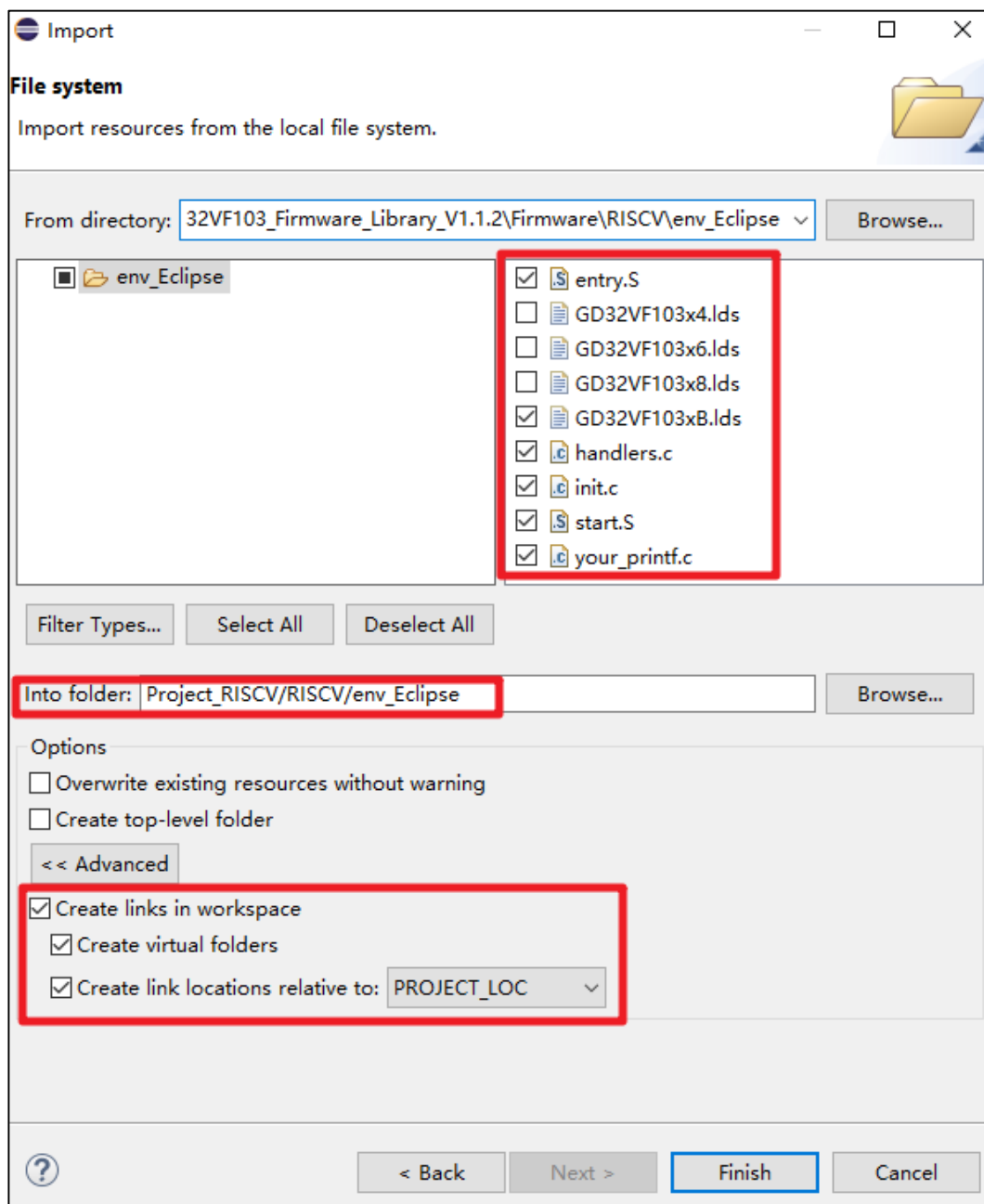


图 3-15. Import RISC-V-drivers 文件夹文件

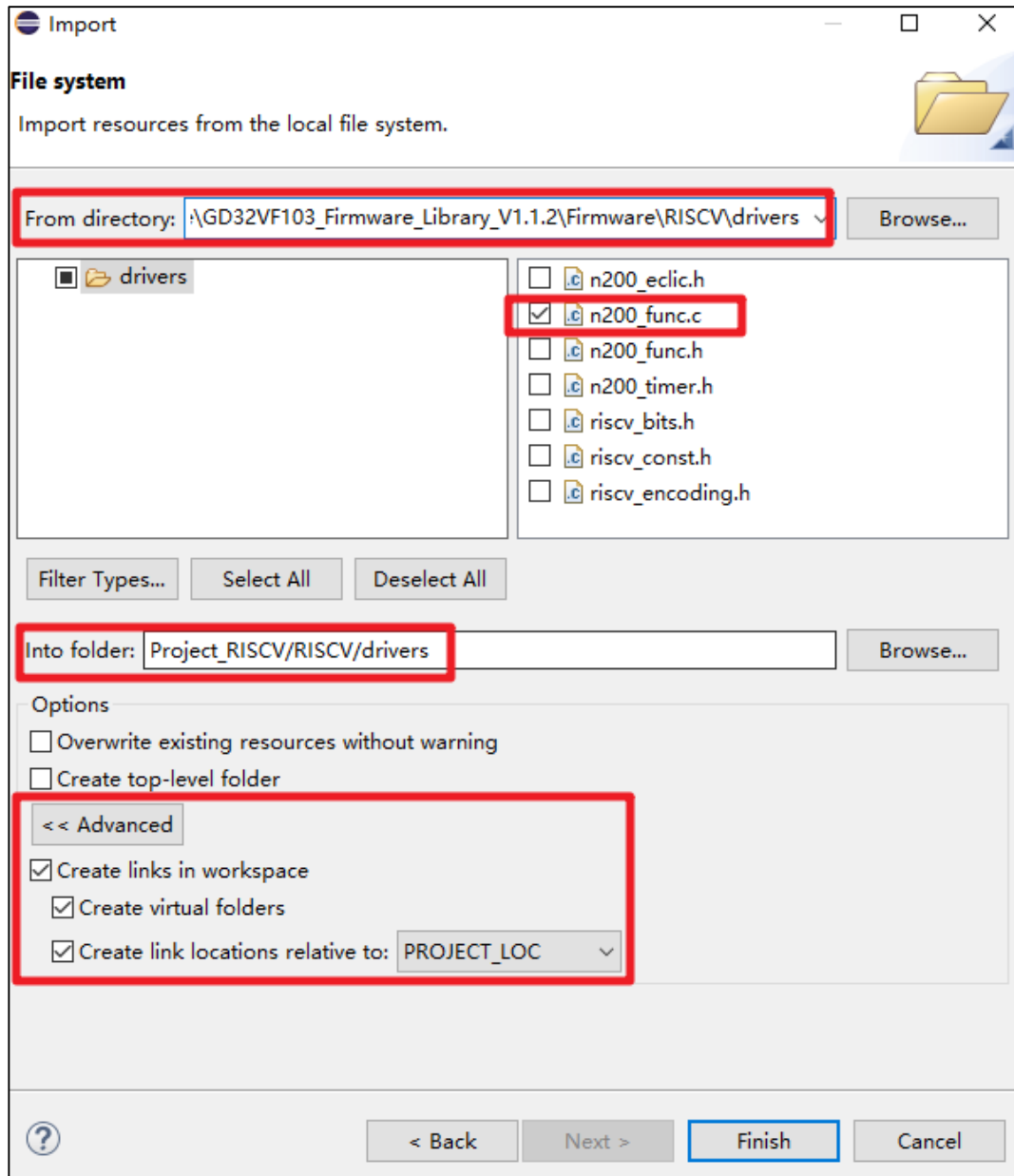


图 3-16. Import RISCv-stubs 文件夹文件

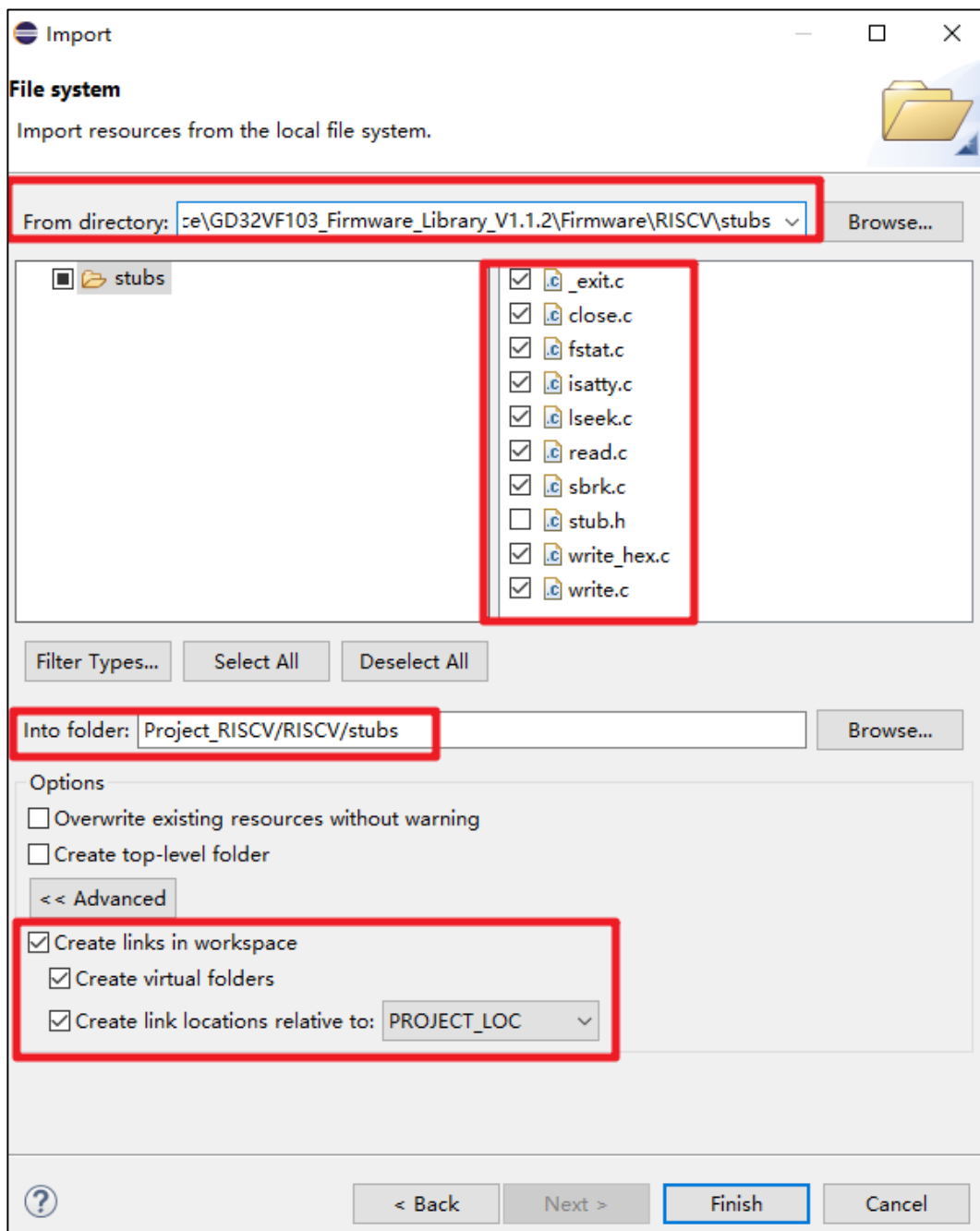


图 3-17. Import Doc 文件夹文件

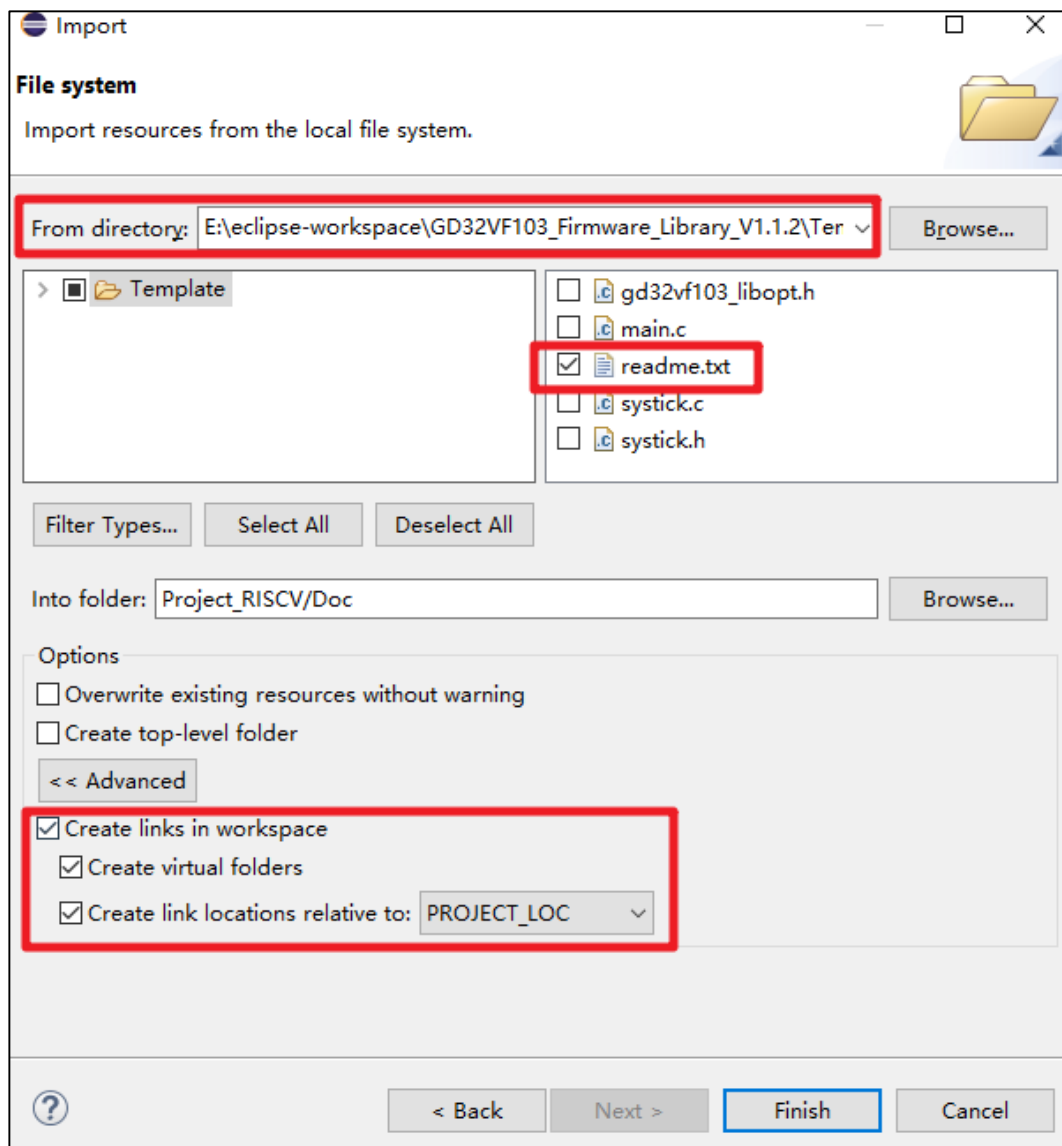


图 3-18. Import Utilities 文件夹文件

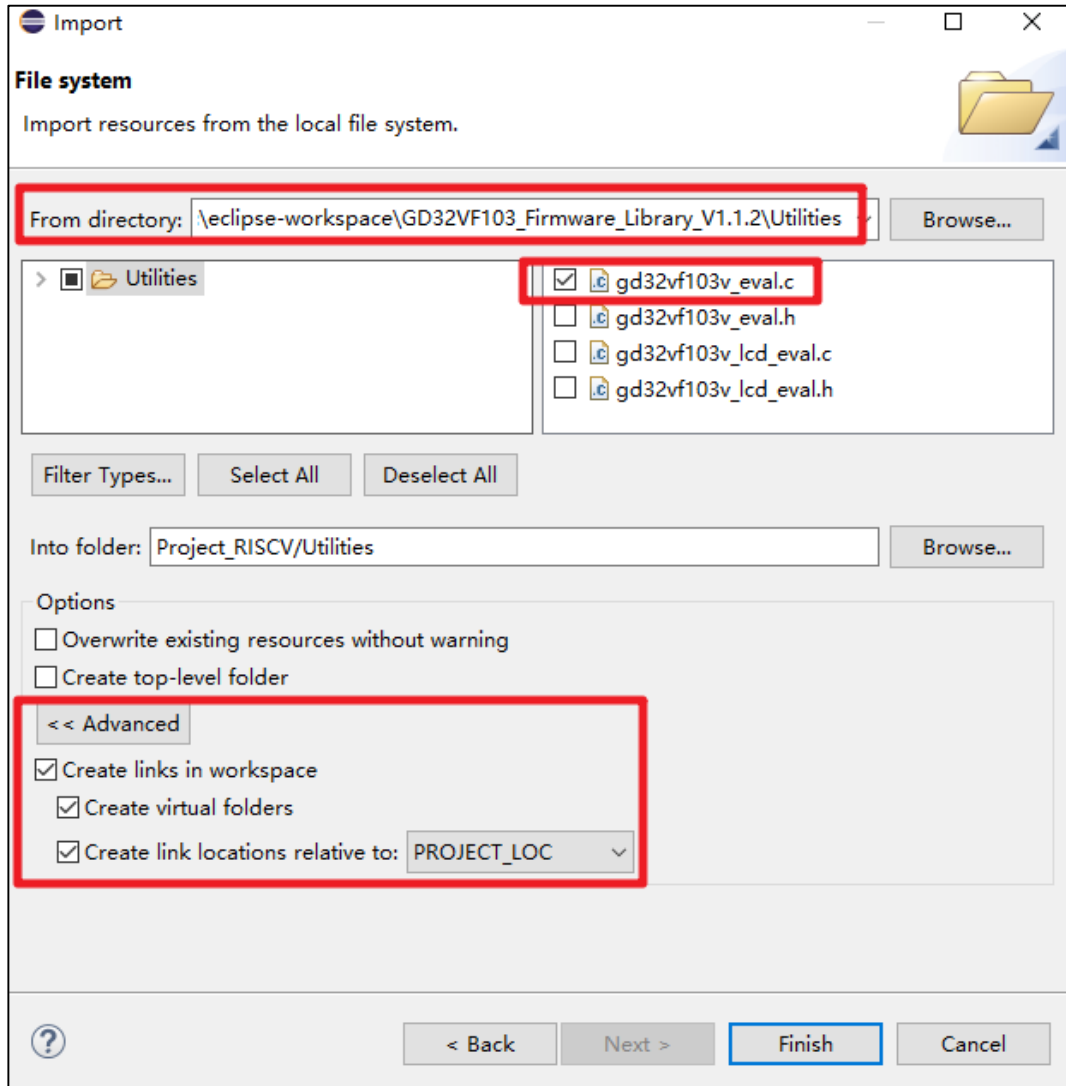
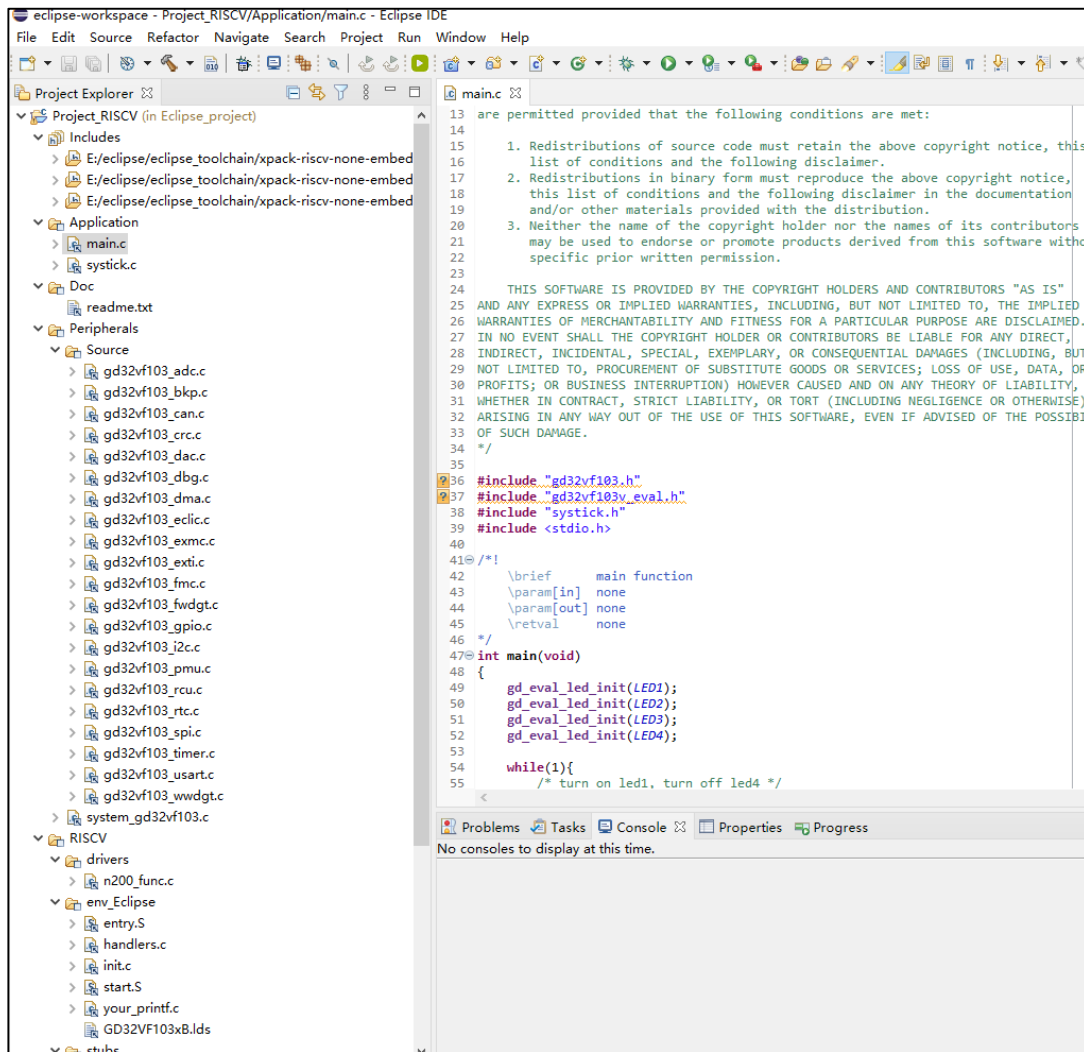


图 3-19. 最终 RISC-V 工程视图



3.2.2. “Refresh”方式建立文件夹及添加文件

除上述手动依次建立文件夹及导入对应文件的方式外，也可以将需要导入的文件连同文件内直接放在与建立的.cproject同级的文件夹内。在Eclipse IDE中，右击工程名，并选择Refresh，即可将文件夹及文件直接导入工程。

图 3-20. 工程文件夹结构

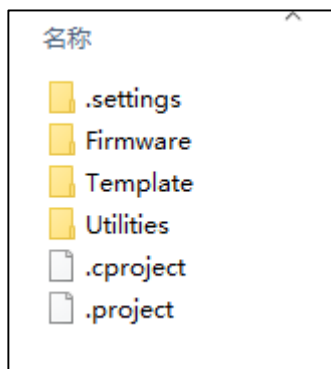


图 3-21. Refresh 工程

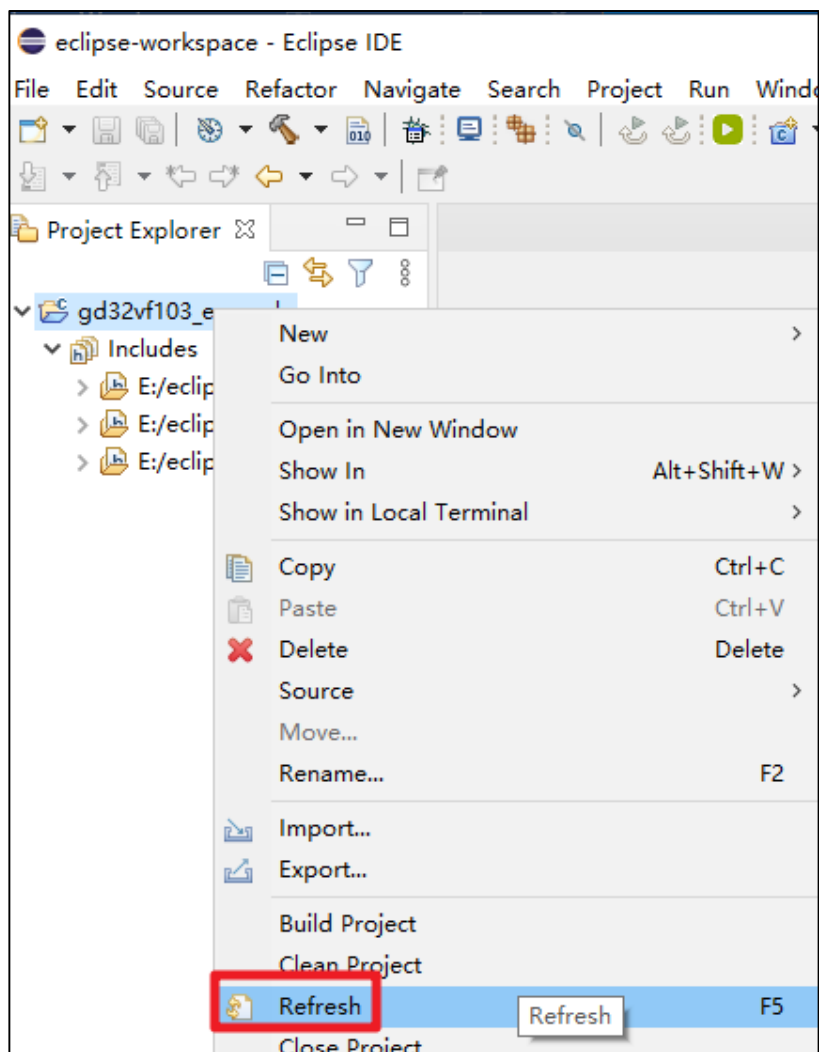
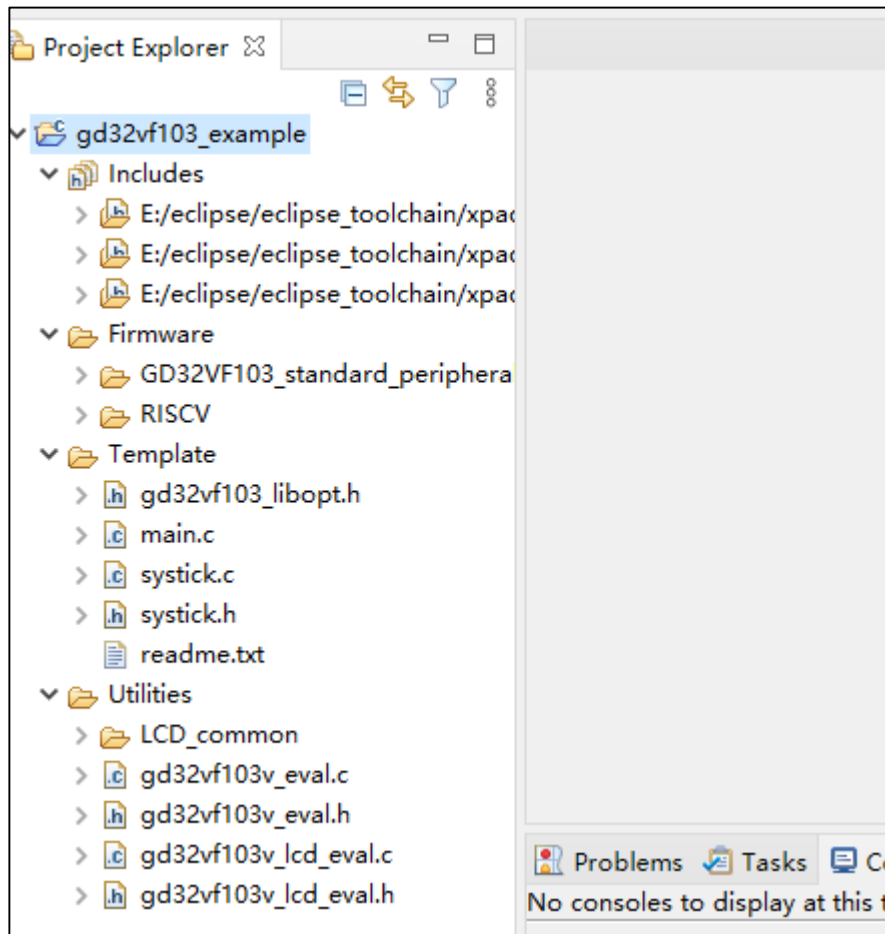


图 3-22. Eclipse IDE 内工程结构

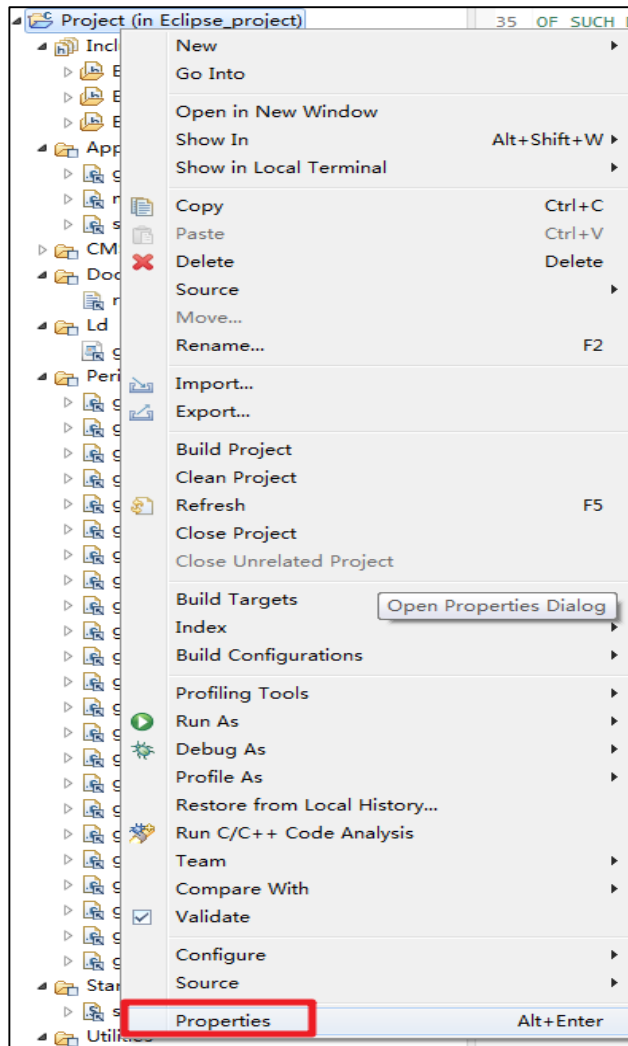


注意：“Refresh”方式建立的文件及文件夹，均是真实存在的，并且一旦在Eclipse IDE中删除其中的某个文件，该文件就会直接从磁盘中删除了。

3.3. 工程配置

右击工程，选择工程属性Properties选项打开。

图 3-23. 工程属性配置

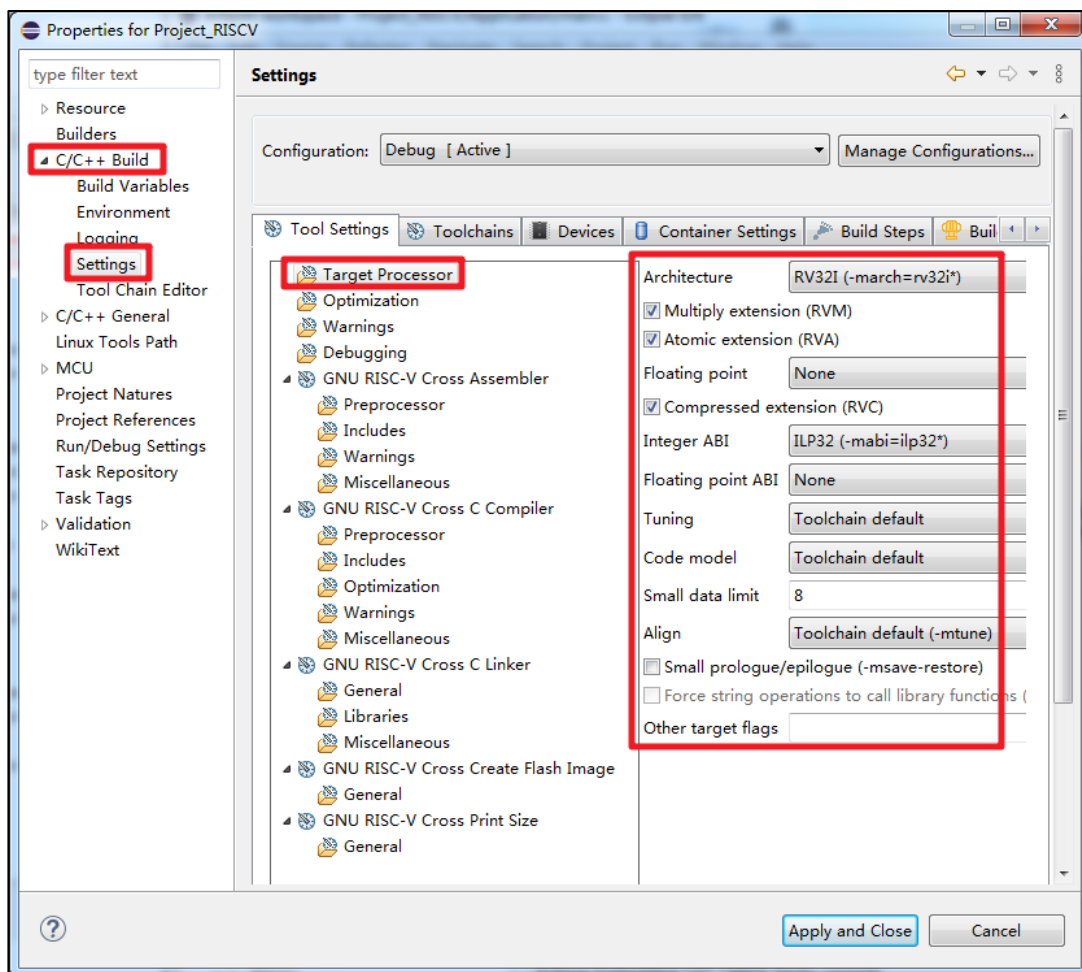


3.3.1. Target Processor 配置

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Target。在本例中，选择RV32I。

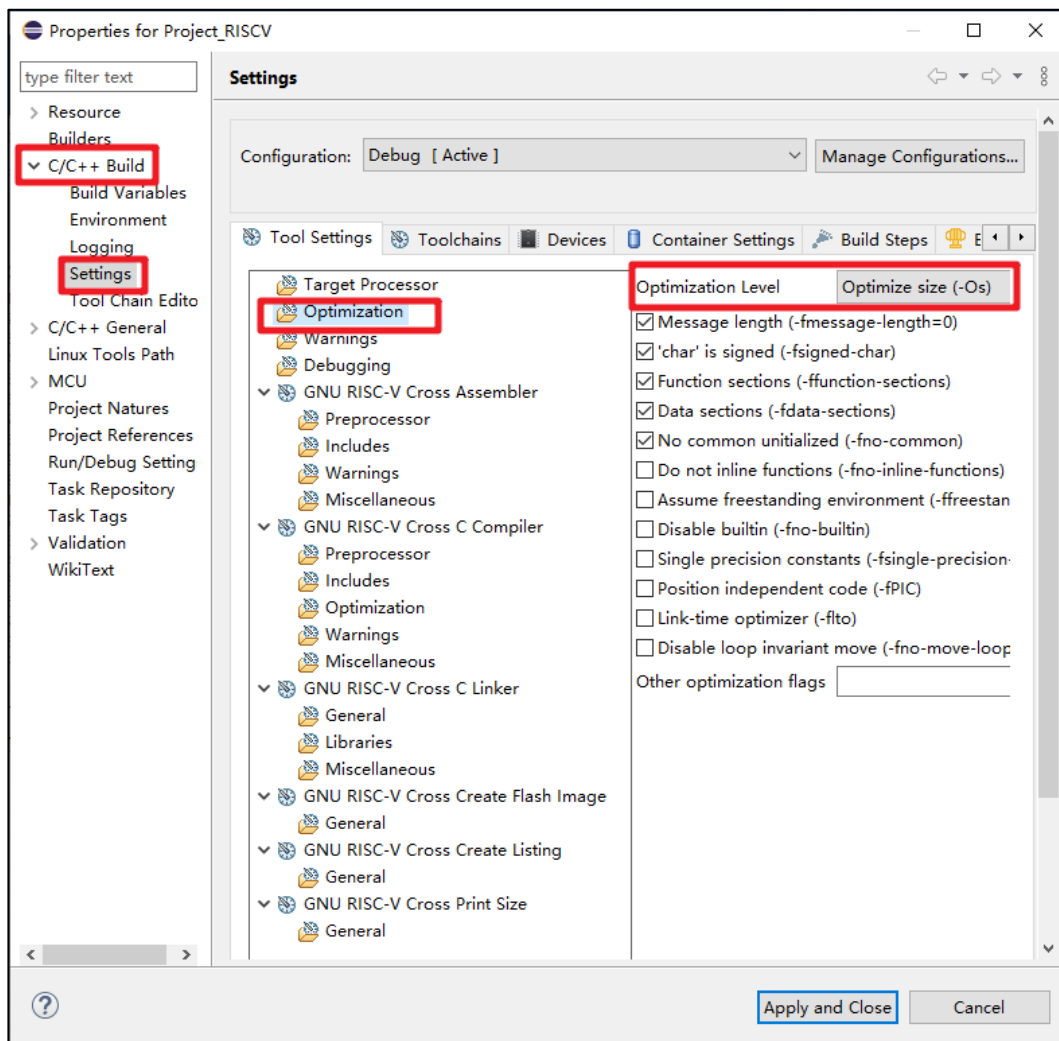
图 3-24. RISC-V 工程 Target Processor 配置



3.3.2. Optimization 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。

图 3-25. Optimization 配置



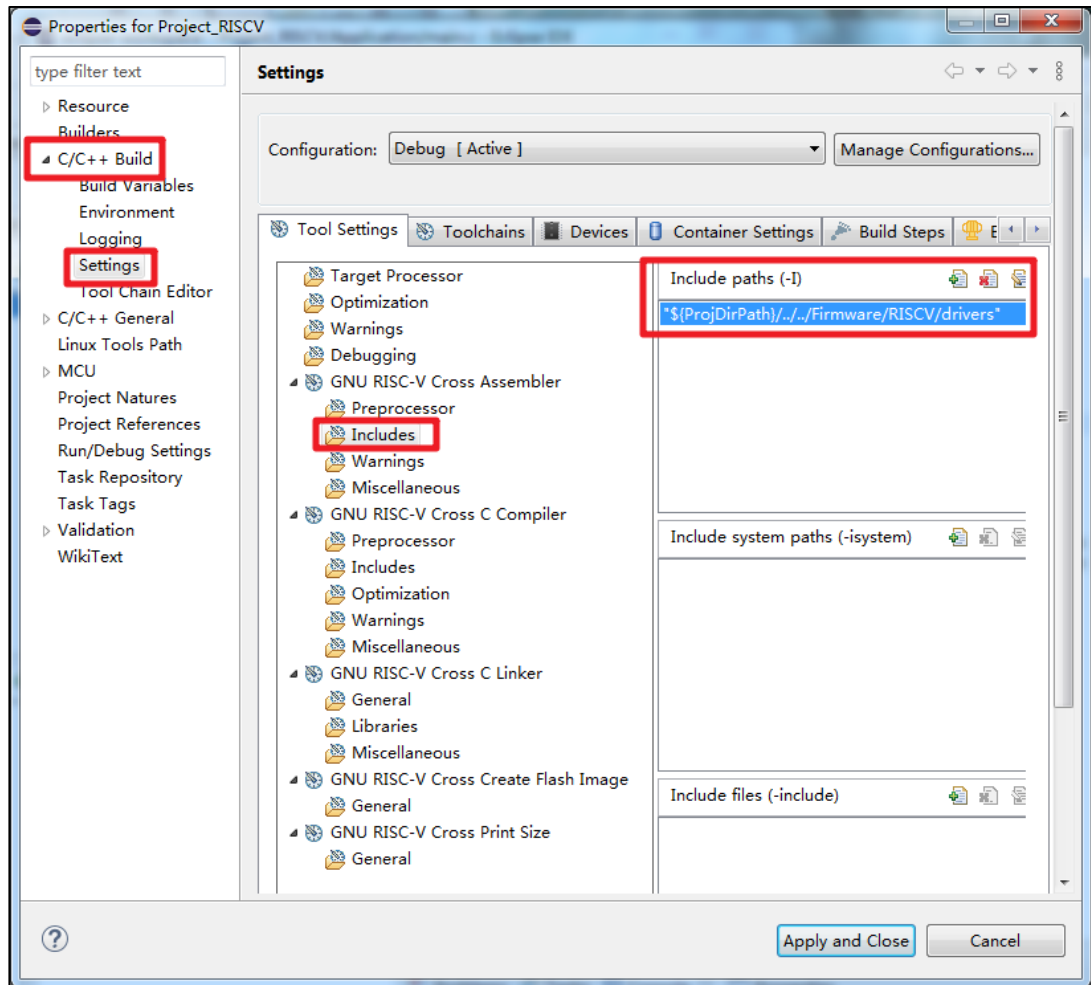
3.3.3. GNU RISC-V Cross Assembler 配置

在“C/C++ Build->Settings->Tool Settings->GNU RISC-V Cross Assembler”选项中配置Cross汇编编译选项。

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

```
"${ProjDirPath}/../Firmware/RISCV/drivers"
```

图 3-26. GNU RISC-V Cross C Assembler -> Includes 配置

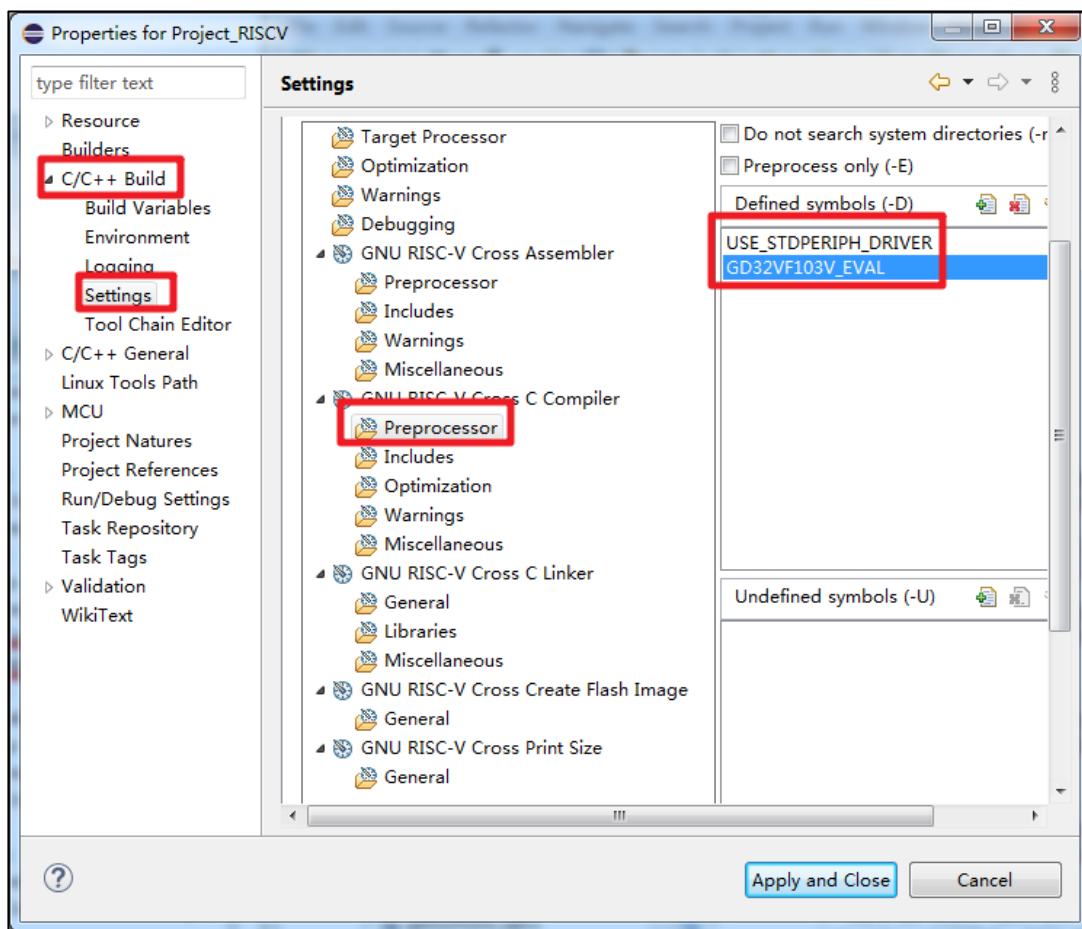


3.3.4. GNU RISC-V Cross C Compiler 配置

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

本例中，在“Preprocessor->Defined symbols”选项中添加USE_STDPERIPH_DRIVER和GD32VF103V_EVAL预编译宏。

图 3-27. GNU RISC-V Cross C Compiler -> Preprocessor 配置



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/RISCV/env_Eclipse"
```

```
"${ProjDirPath}/../Firmware/RISCV/drivers"
```

```
"${ProjDirPath}/../Firmware/GD32VF103_standard_peripheral/Include"
```

```
"${ProjDirPath}/../Template"
```

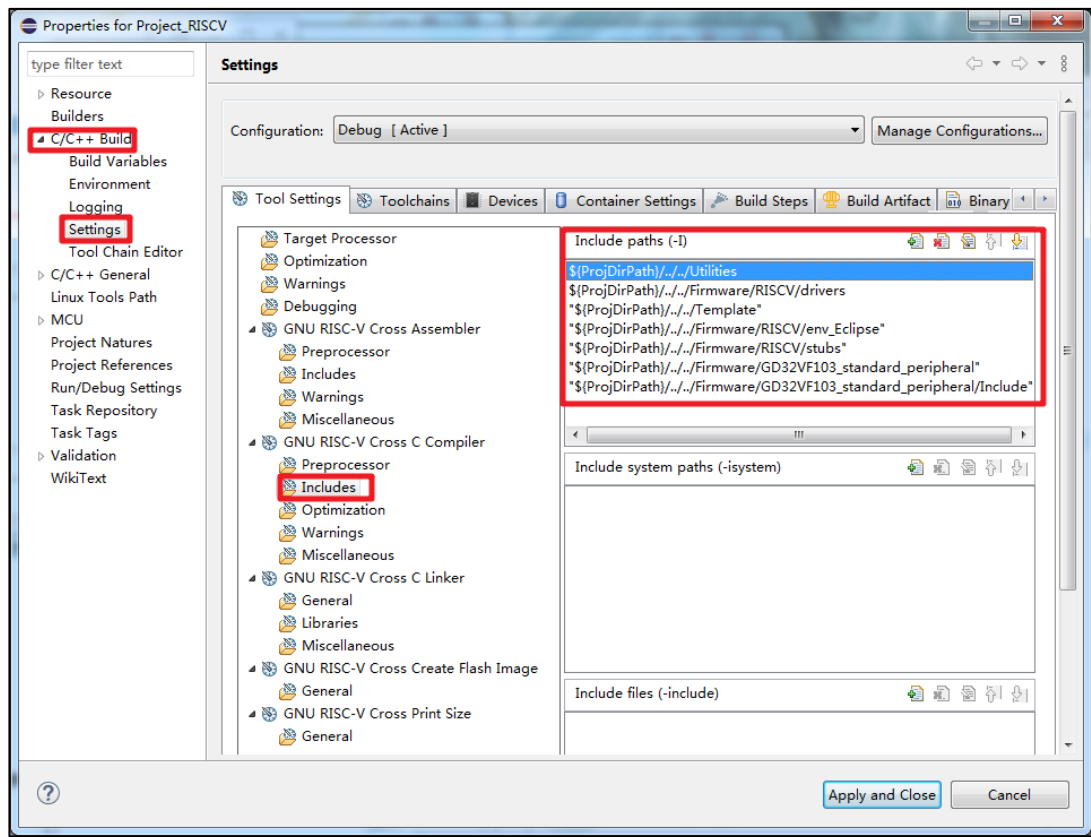
```
"${ProjDirPath}/../Utilities"
```

```
"${ProjDirPath}/../Firmware/RISCV/stubs"
```

```
"${ProjDirPath}/../Firmware/GD32VF103_standard_peripheral"
```

注意：本例中添加的头文件路径为相对路径。这里也可以直接添加绝对路径。

图 3-28. GNU RISC-V Cross C Compiler -> Includes 配置



3.3.5. GNU RISC-V Cross C Linker 配置

在“C/C++ Build->Settings->Tool Settings->GNU RISC-V Cross C Linker”配置Cross C链接选项。

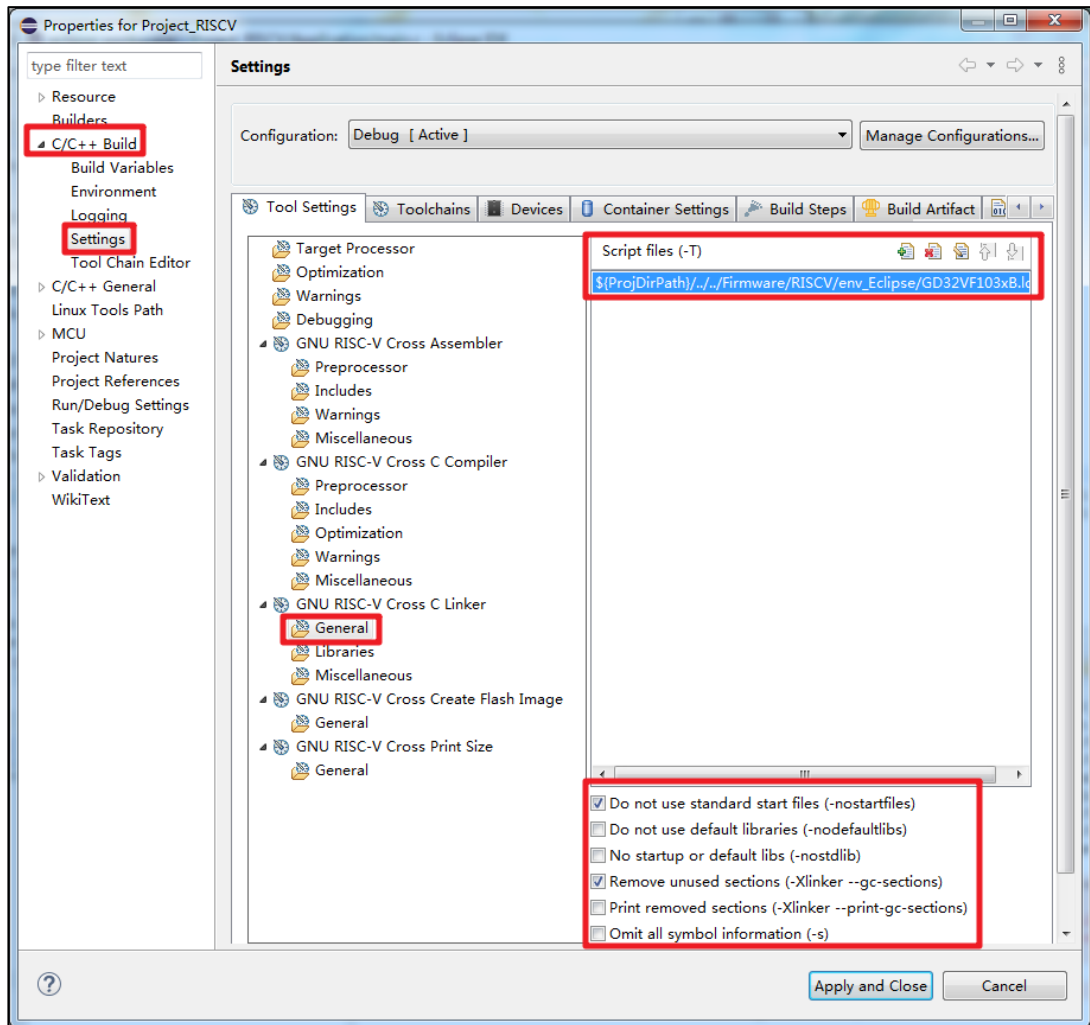
在“General ->Script files”选项中添加：

```
"${ProjDirPath}/../Firmware/RISCV/env_Eclipse/GD32VF103xB.Ids"
```

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。

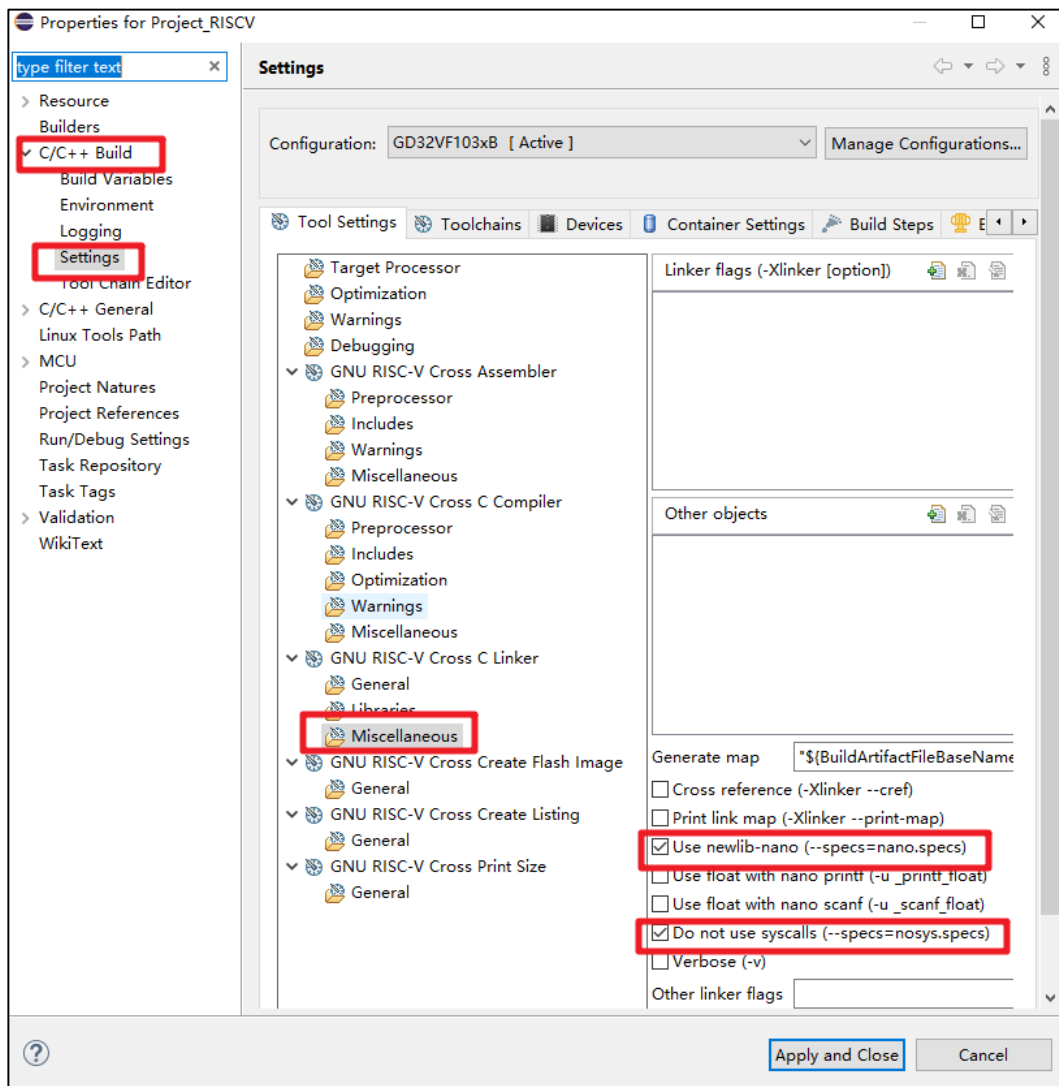
注意：本例中添加的ld文件路径为相对路径。这里也可以直接添加绝对路径。

图 3-29. GNU RISC-V Cross C Linker -> General 配置



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

图 3-30. GNU RISC-V Cross C Linker -> Miscellaneous 配置

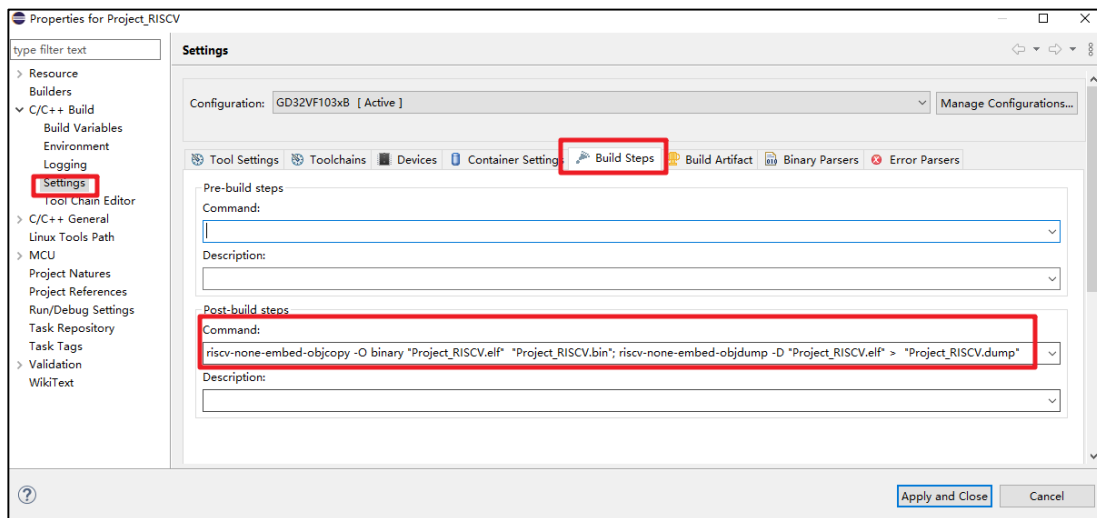


3.3.6. Build Steps 配置-生成 bin 文件

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

本例中添加：`riscv-none-embed-objcopy -O binary "Project_RISCV.elf" "Project_RISCV.bin"; riscv-none-embed-objdump -D "Project_RISCV.elf" > "Project_RISCV.dump"`

图 3-31. Build Steps 配置

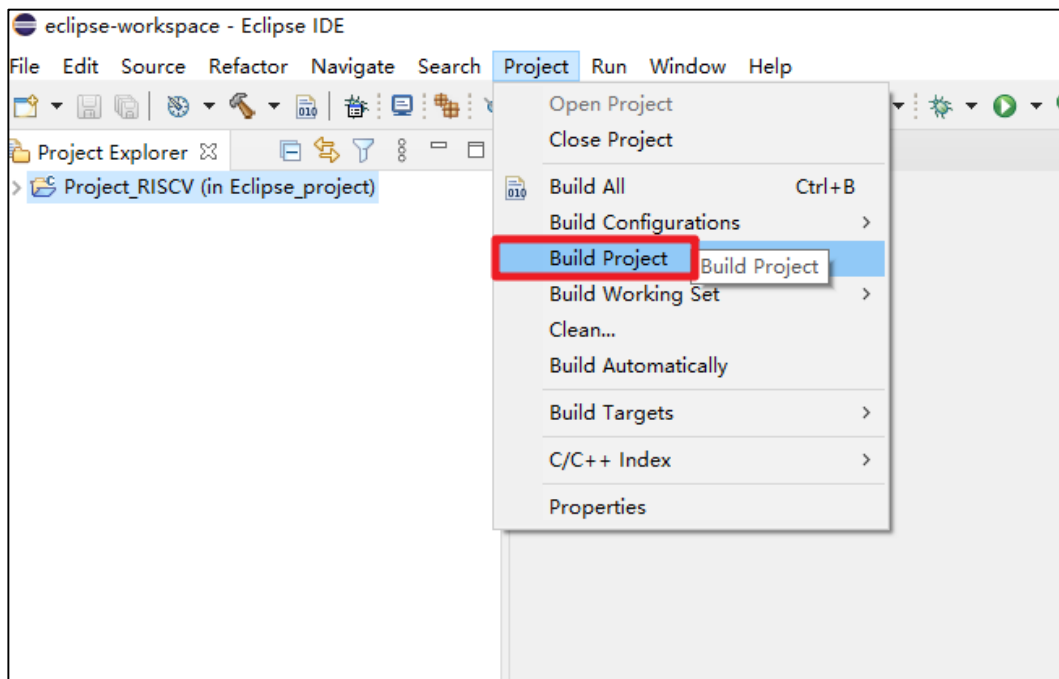


3.4. 编译工程

选择Project->Build Project可编译当前工程。

注意：Build Project是编译当前工程，Build All是编译当前workspace所有工程。

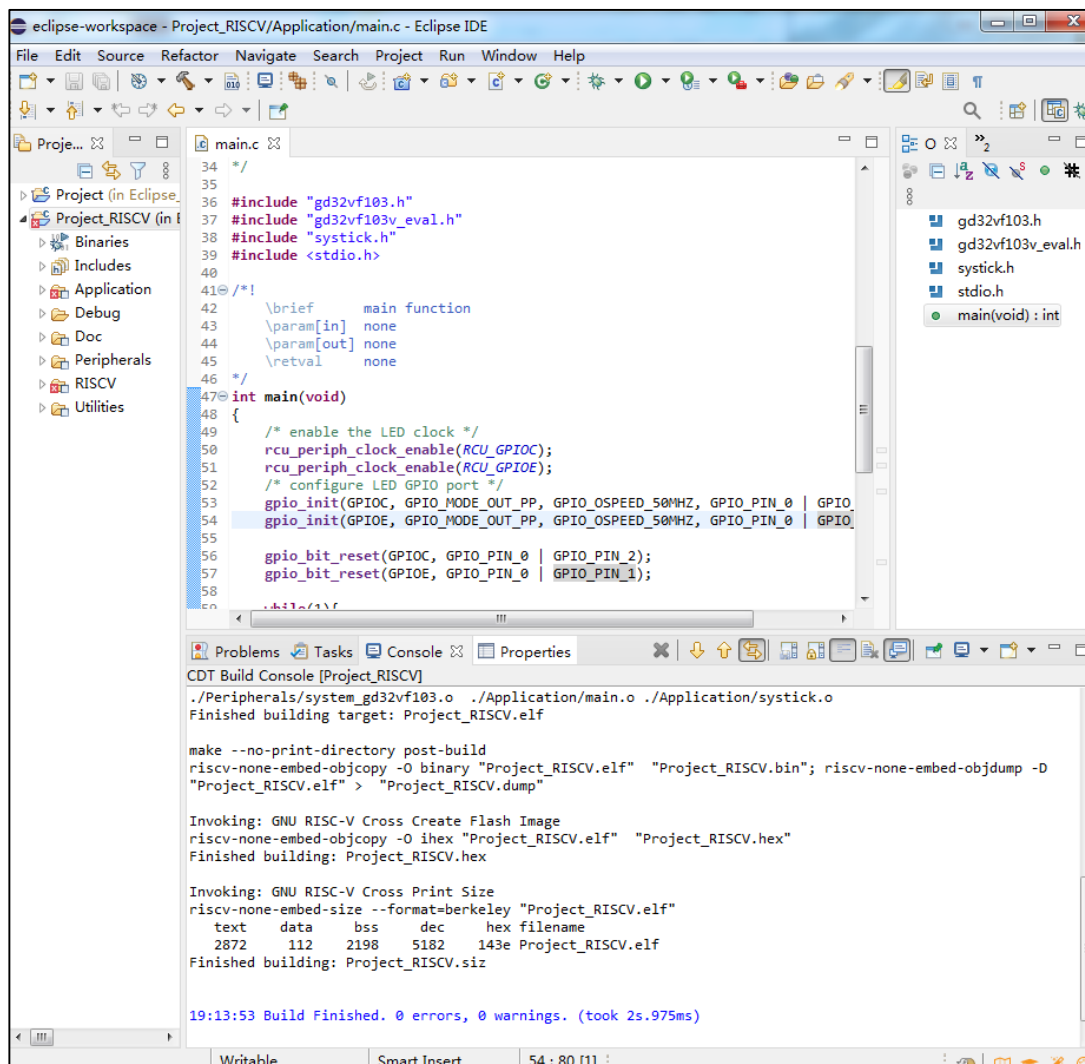
图 3-32. Build 工程



注意：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。

图 3-33. Build RISC-V 工程完成

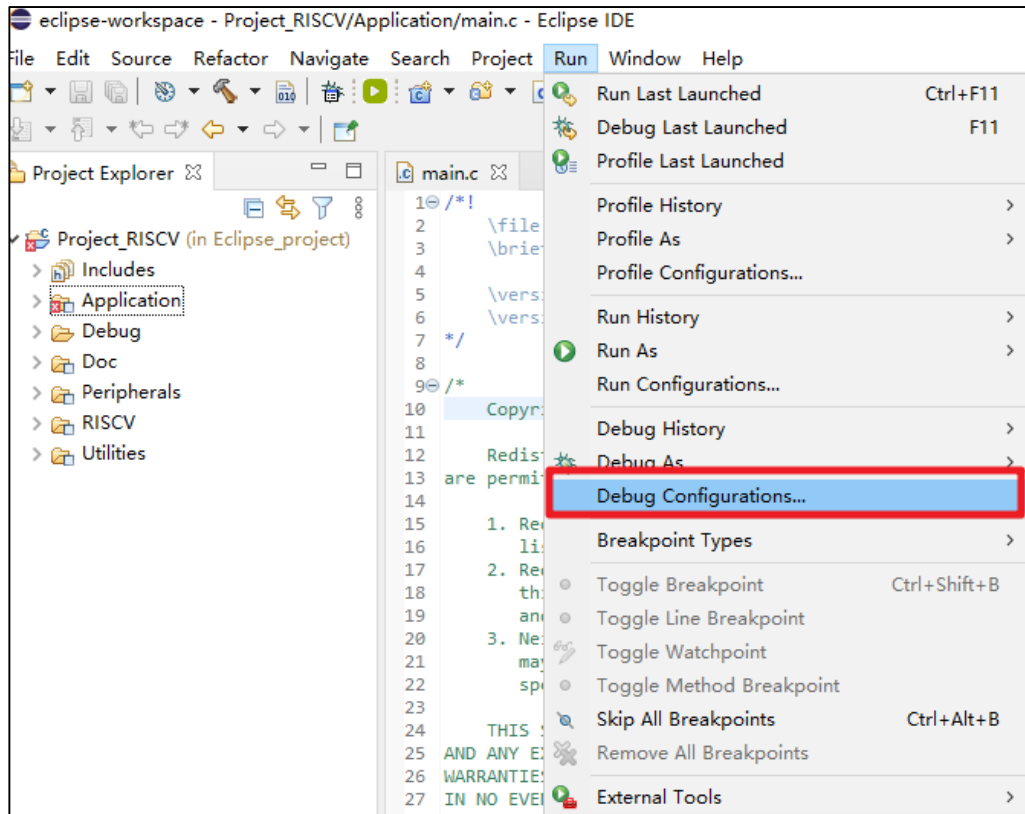


3.5. 使用 J-Link 下载及调试工程

3.5.1. Debug 配置界面

在菜单栏中，点击Run->Debug Configurations，进入Debug配置界面。

图 3-34. 进入 Debug Configurations 界面

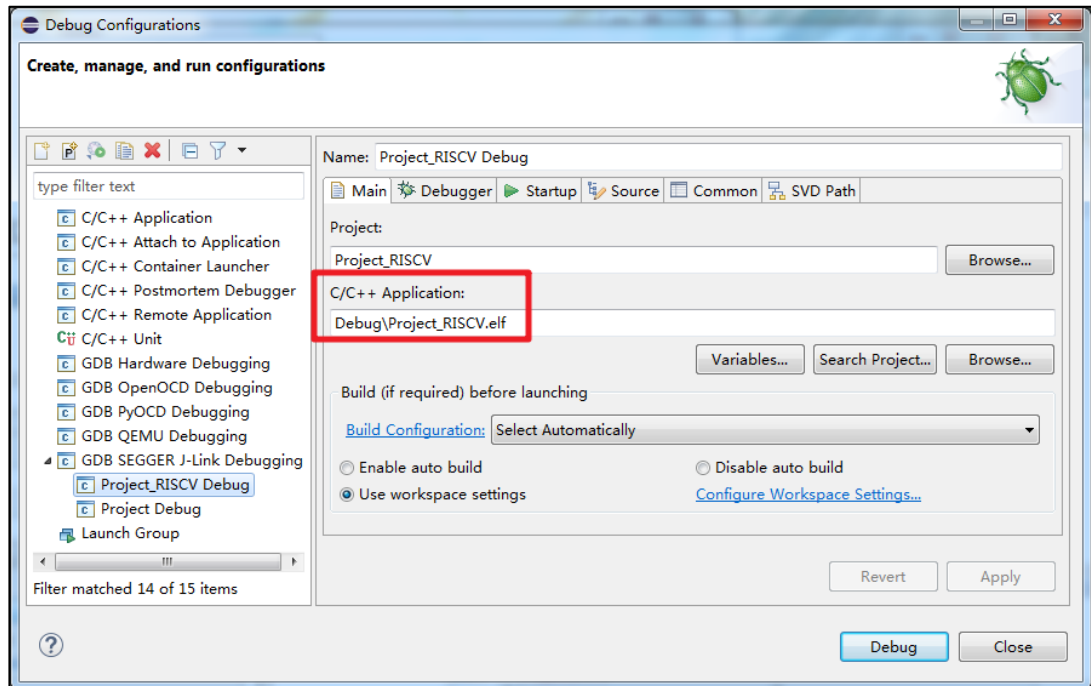


使用J-Link GDBServerCL作为GDB Server，使用GCC工具链中的GDB工具作为GDB Client。

双击GDB SEGGER J-Link Debugging，新建一套J-Link的配置选项。

3.5.2. Main 选项卡

图 3-35. GDB SEGGER J-Link Debugging-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。

注意：如果之前编译了多个型号，需要选择对应的可执行文件.elf。为了方便，也可以对每个型号都新建一套Debug的配置。

3.5.3. Debugger 选项卡

在Debugger选项卡中，填写目标芯片型号Device name，本例中为GD32VF103VBT6。

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。

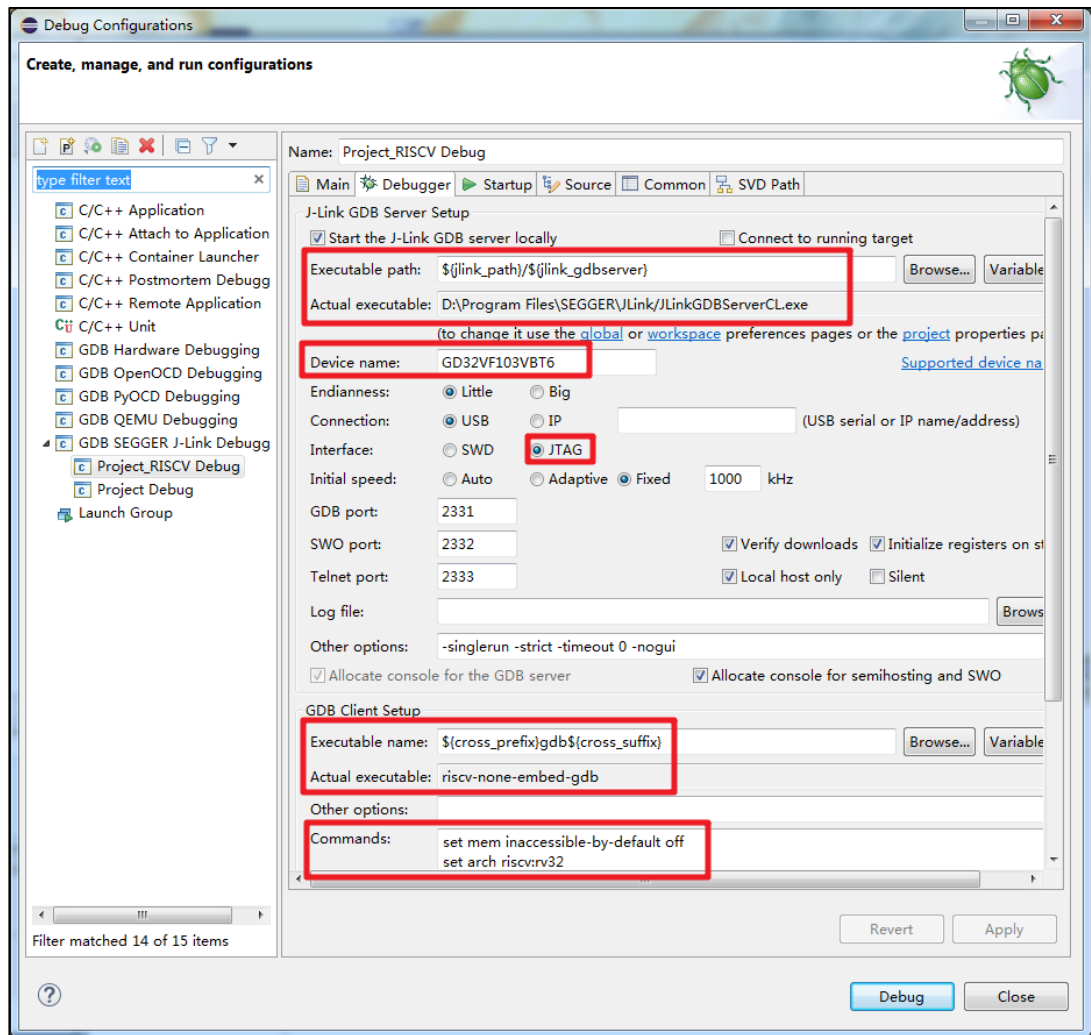
Commands栏填写：

```
set mem inaccessible-by-default off
```

```
set arch riscv:rv32
```

注意：填写的芯片型号，在所配置的J-Link驱动中必须支持。如果不支持，请升级J-Link驱动版本。

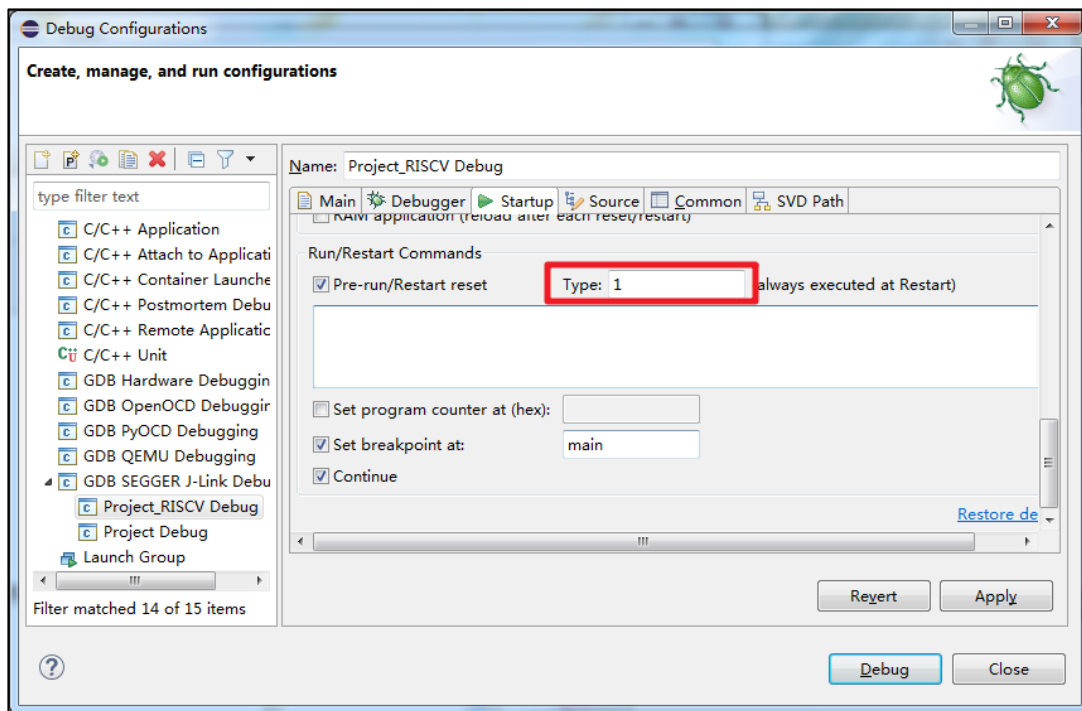
图 3-36. GDB SEGGER J-Link Debugging-Debugger 选项卡



3.5.4. Startup 选项卡

在Startup选项卡中配置:

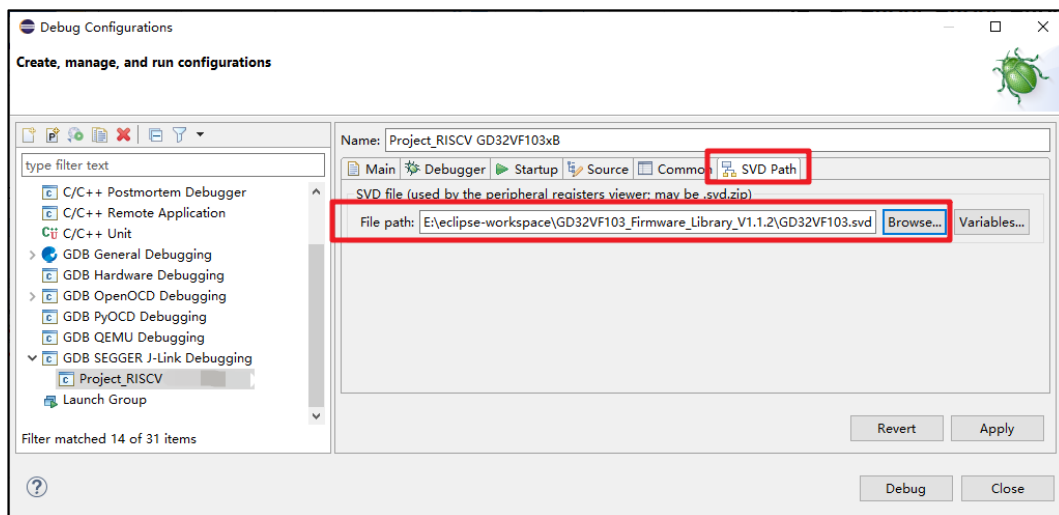
图 3-37. GDB SEGGER J-Link Debugging-Startup 选项卡



3.5.5. SVD Path 选项卡

在 SVD Path 选项卡，选择目标芯片所需的 SVD 文件。

图 3-38. GDB SEGGER J-Link Debugging-SVD Path 选项卡

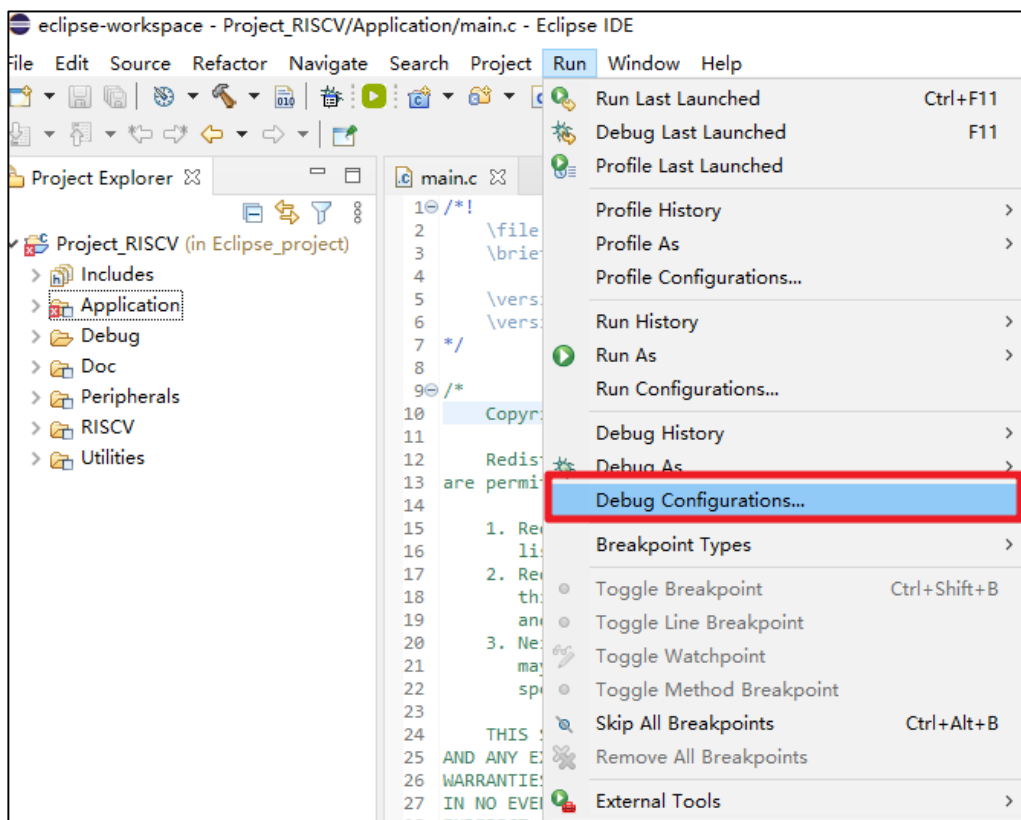


3.6. 使用 GD-Link 下载及调试工程

3.6.1. Debug 配置界面

在菜单栏中，点击Run->Debug Configurations，进入Debug配置界面。

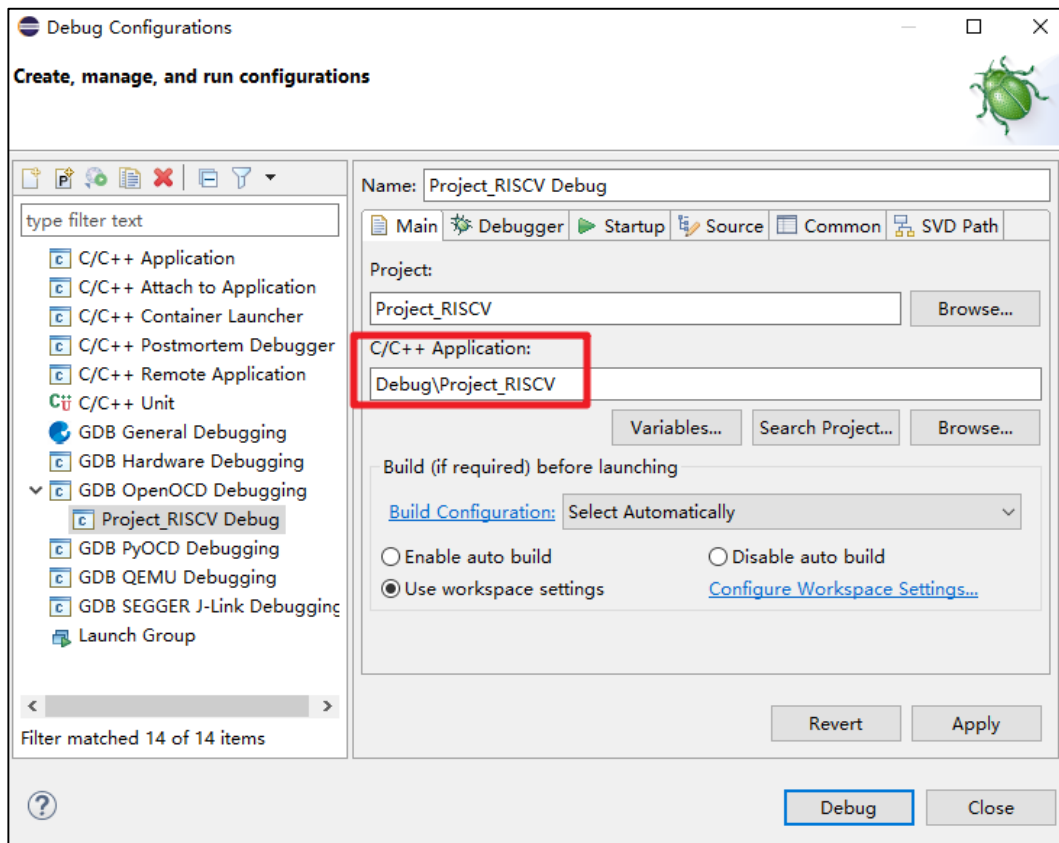
图 3-39. 进入 Debug Configurations 界面



使用OpenOCD作为GDB Server，使用GCC工具链中的GDB工具作为GDB Client。双击GDB OpenOCD Debugging，新建一套OpenOCD的配置选项。

3.6.2. Main 选项卡

图 3-40. GDB OpenOCD Debugging-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。

注意：如果之前编译了多个型号，需要选择对应的可执行文件.elf。为了方便，也可以对每个型号都新建一套Debug的配置。

3.6.3. Debugger 选项卡

若在搭建Eclipse环境时已正确配置OpenOCD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OpenOCD的绝对路径。

在Config options栏中，填写使用的cfg文件。本例中为：

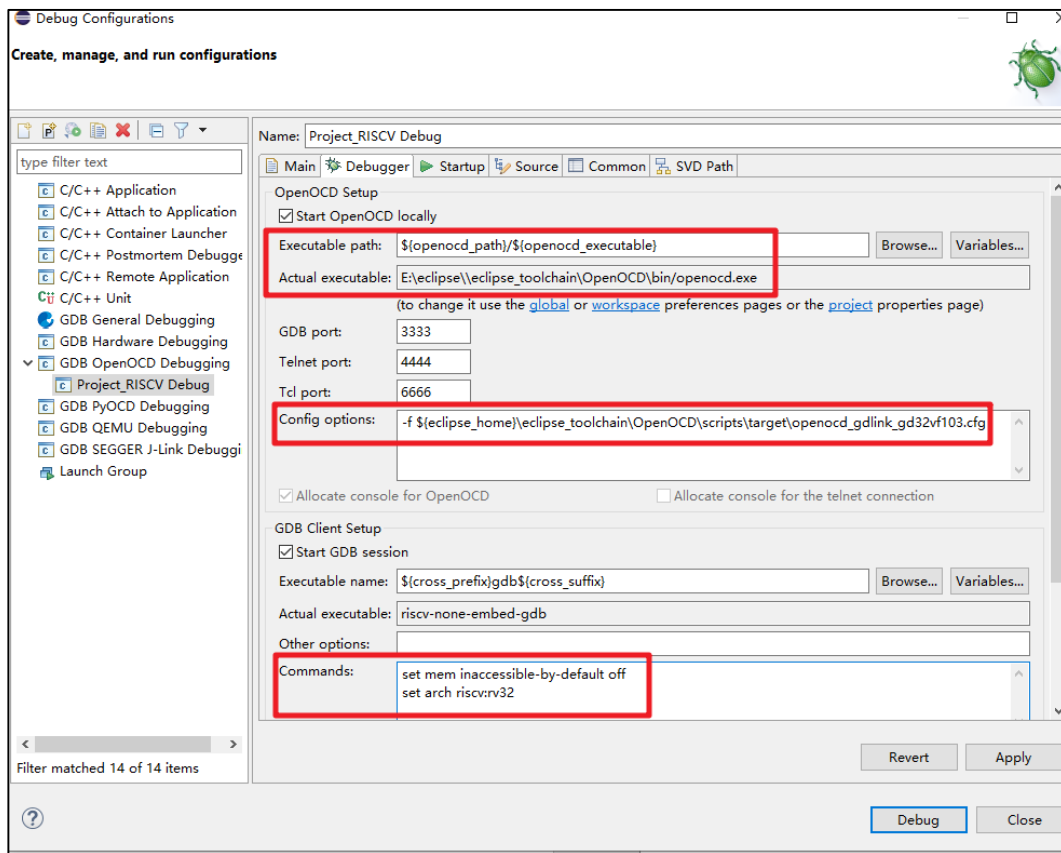
```
-f ${eclipse_home}\eclipse_toolchain\OpenOCD\scripts\target\openocd_gdlink_gd32vf103.cfg
```

OpenOCD的cfg文件提供了调试器、调试协议、目标芯片识别及目标芯片烧写算法选择等信息。

在Commands栏中，填写：

```
set arch riscv:rv32
```

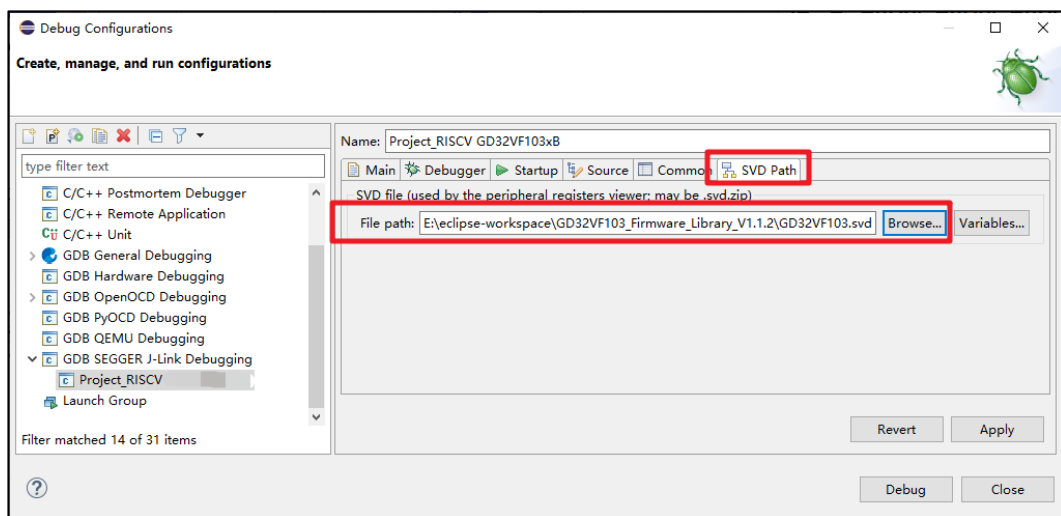
图 3-41. GDB OpenOCD Debugging-Debugger 选项卡



3.6.4. SVD Path 选项卡

在 SVD Path 选项卡，选择目标芯片所需的 SVD 文件。

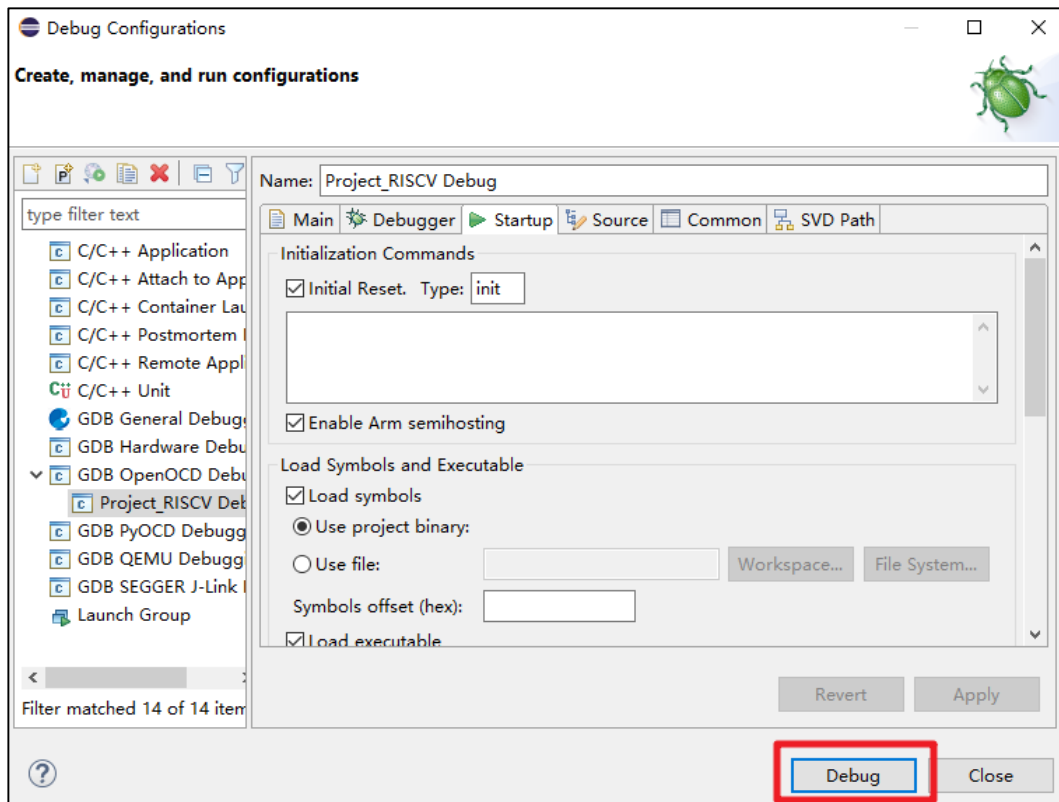
图 3-42. GDB OpenOCD Debugging-SVD Path 选项卡



3.7. Debug 界面

Debug Configurations配置完成后，点击Debug，进入Debug视图。

图 3-43. 进入 Debug 视图-1



切换至Debug视图。

图 3-44. 进入 Debug 视图-2

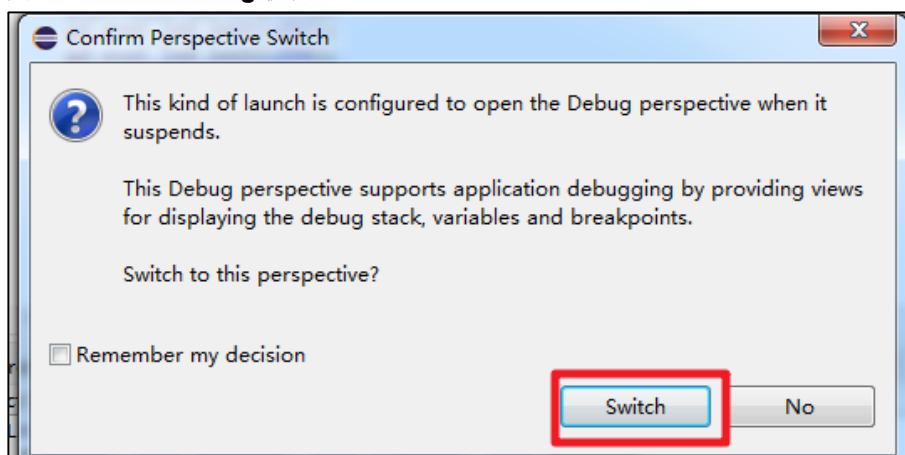
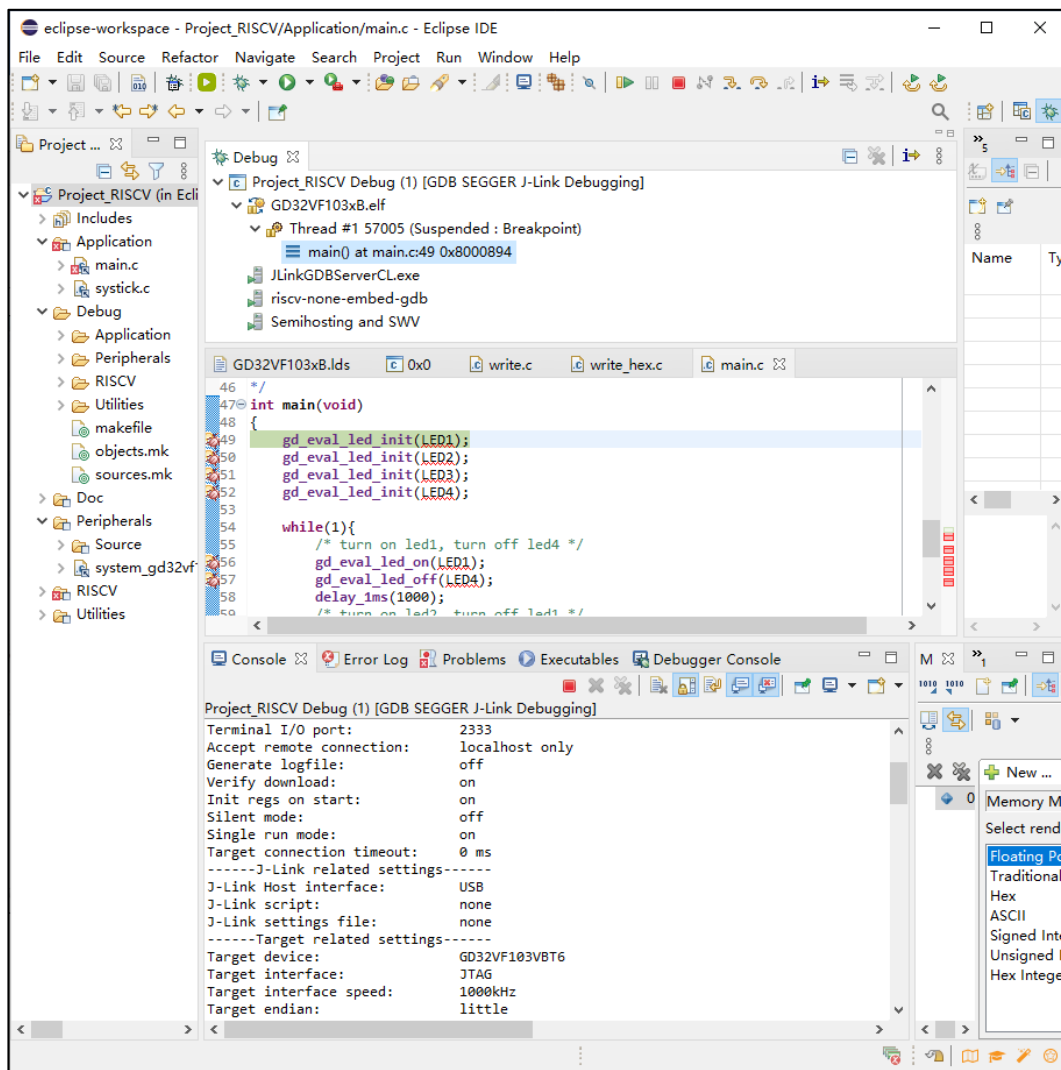



图 3-45. Debug 视图



3.7.1. 工具栏介绍


 : resume 全速运行

 : suspend 暂停

 : terminate 退出

 : step into

 : step over

 : step out : reset

3.7.2. Registers 窗口

在菜单栏，选择Window->Show view->Registers选项，打开即可查看通用寄存器的值。

图 3-46. 打开 Registers 窗口

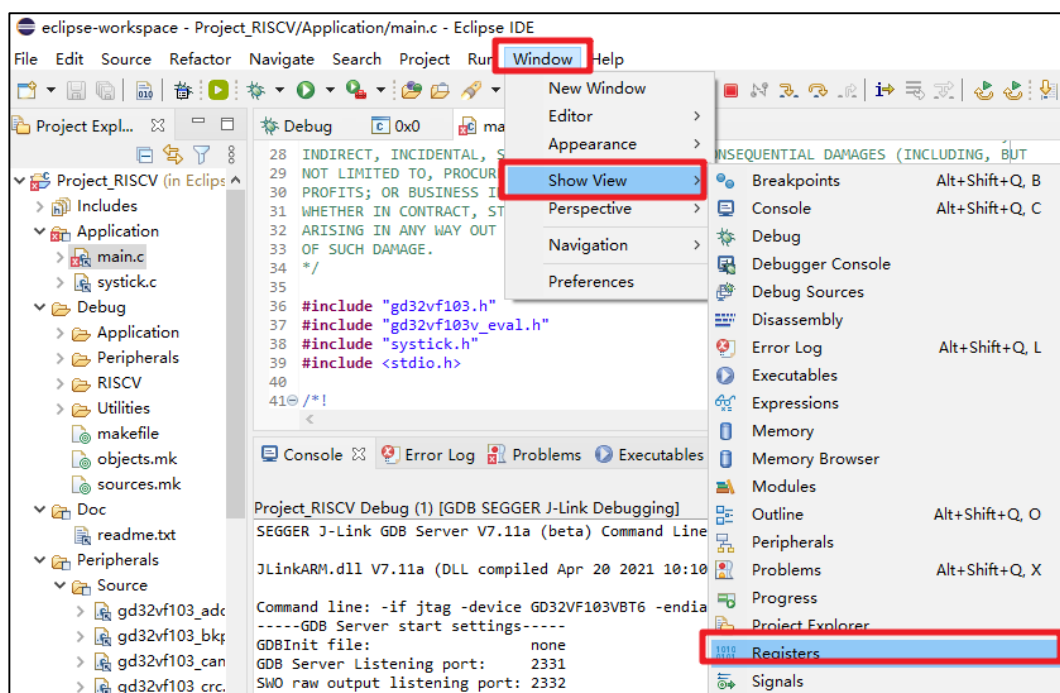
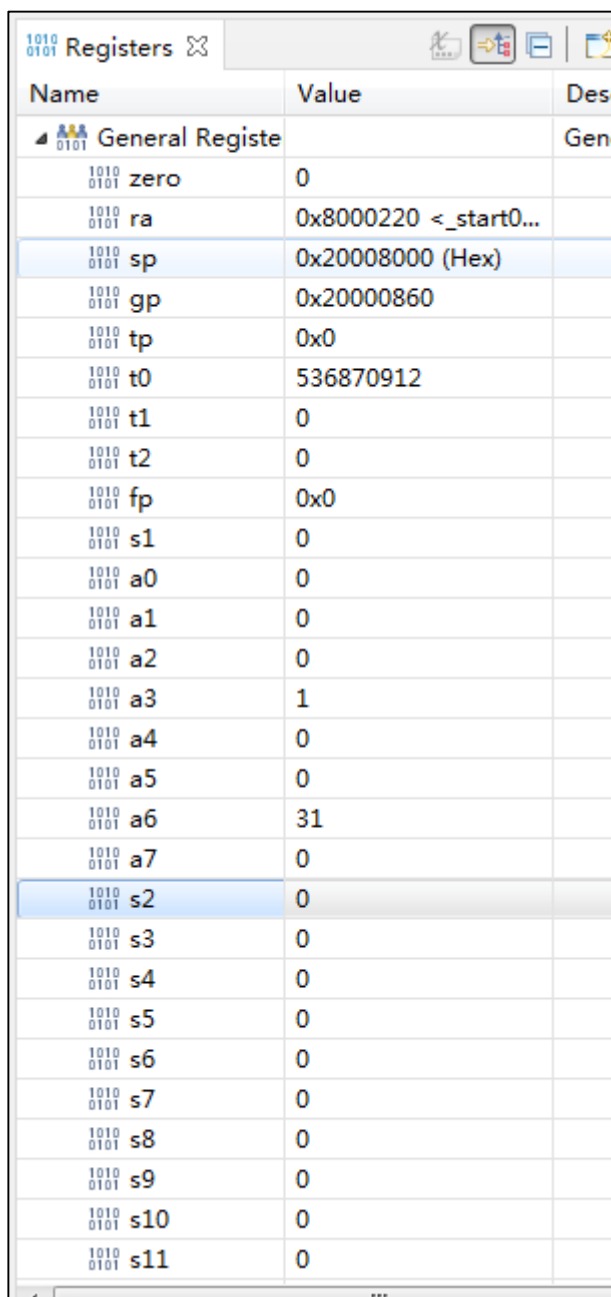


图 3-47. Registers 窗口



The screenshot shows the 'Registers' window in Eclipse IDE. The window title is 'Registers' with a search icon. It contains a table with three columns: 'Name', 'Value', and 'Description'. The table lists various RISC-V registers, including General Registers (zero, ra, sp, gp, tp, t0, t1, t2, fp, s1, a0, a1, a2, a3, a4, a5, a6, a7, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11) and their current values. The 'sp' register is highlighted in blue, showing a value of '0x20008000 (Hex)'. The 's2' register is also highlighted in blue, showing a value of '0'. The 'a3' register has a value of '1', and the 'a6' register has a value of '31'. All other registers have a value of '0'.

Name	Value	Description
General Register		General Register
zero	0	
ra	0x8000220 <_start0...	
sp	0x20008000 (Hex)	
gp	0x20000860	
tp	0x0	
t0	536870912	
t1	0	
t2	0	
fp	0x0	
s1	0	
a0	0	
a1	0	
a2	0	
a3	1	
a4	0	
a5	0	
a6	31	
a7	0	
s2	0	
s3	0	
s4	0	
s5	0	
s6	0	
s7	0	
s8	0	
s9	0	
s10	0	
s11	0	

3.7.3. Peripherals 窗口

在菜单栏，选择Window->Show view->Peripherals选项，打开即可查看外设寄存器的值。

图 3-48. 打开 Peripherals 窗口

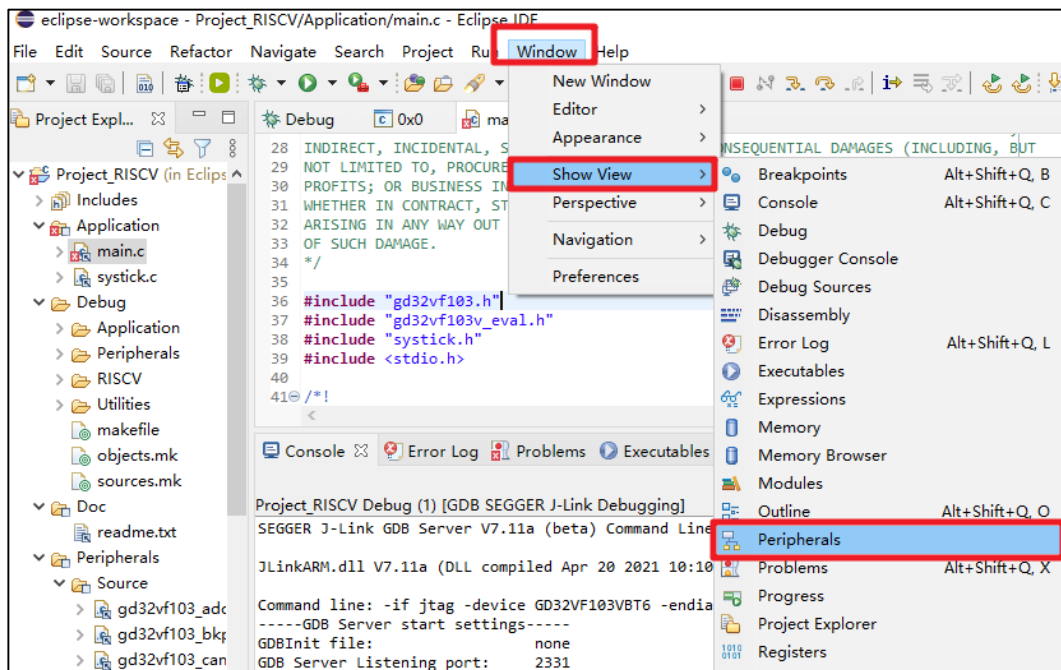
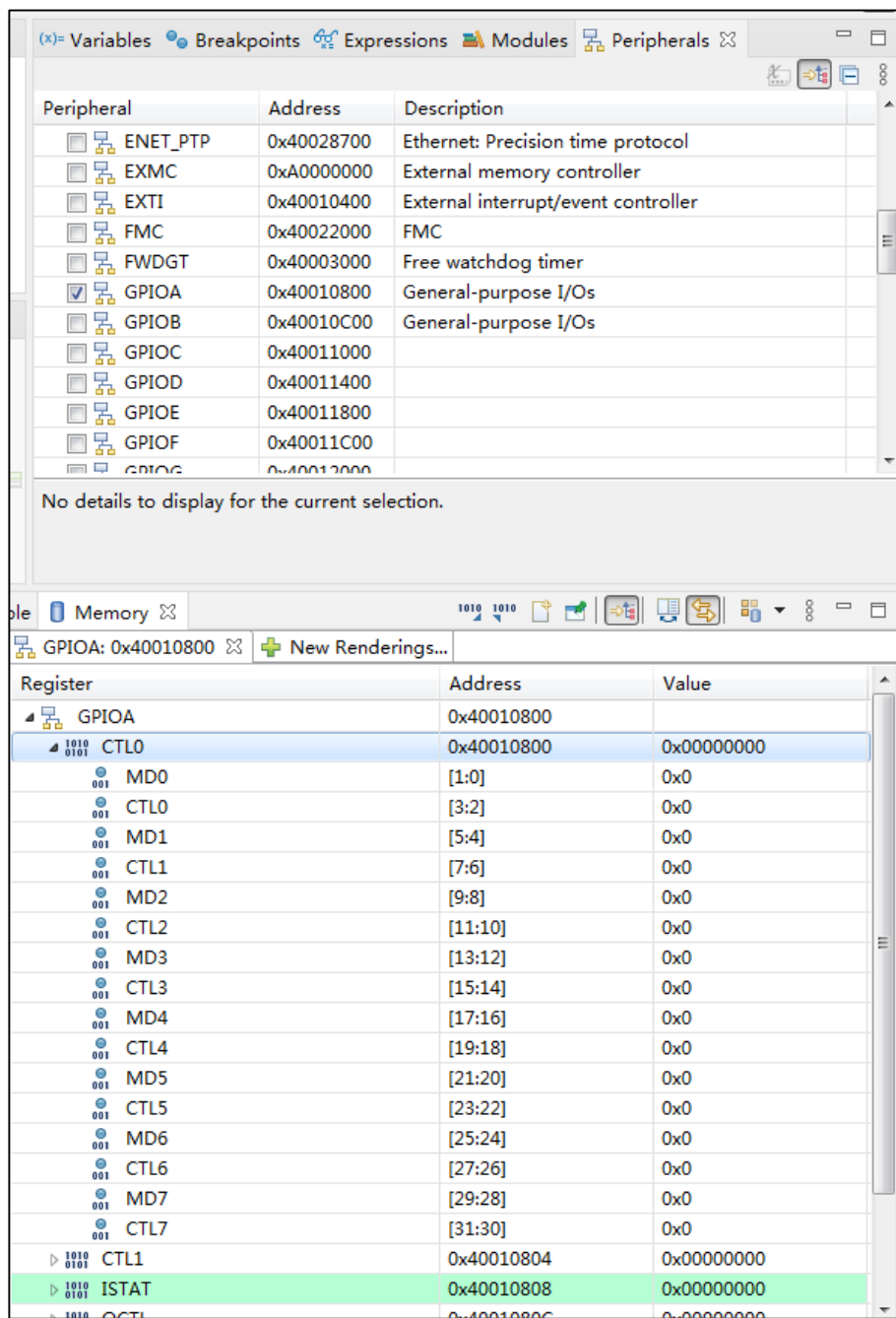


图 3-49. Peripherals 窗口



3.7.4. Memory 窗口

在菜单栏，选择“Window->Show View->Memory”，点击Memory窗口上方的“+”号，可以打开对应的Memory地址。

图 3-50. 打开 Memory 窗口

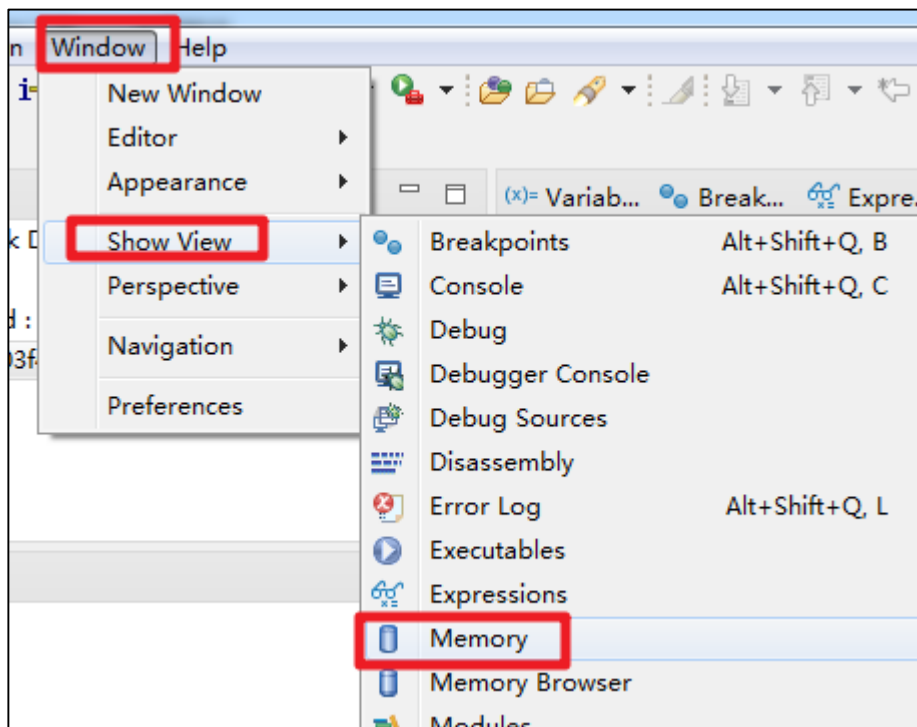
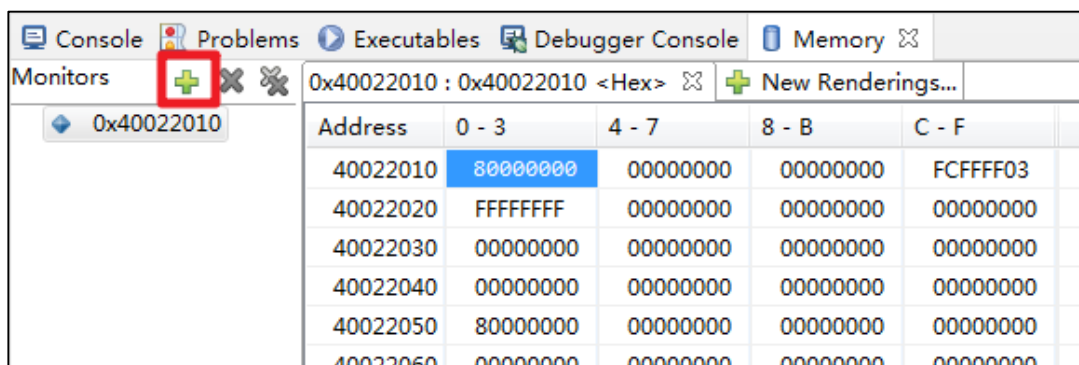


图 3-51. Memory 窗口



3.7.5. Expressions 窗口

在菜单栏，选择“Window->Show View->Expressions”，点击Expressions窗口内“+”号，可以添加和查看相应变量的值。

图 3-52. 打开 Expressions 窗口

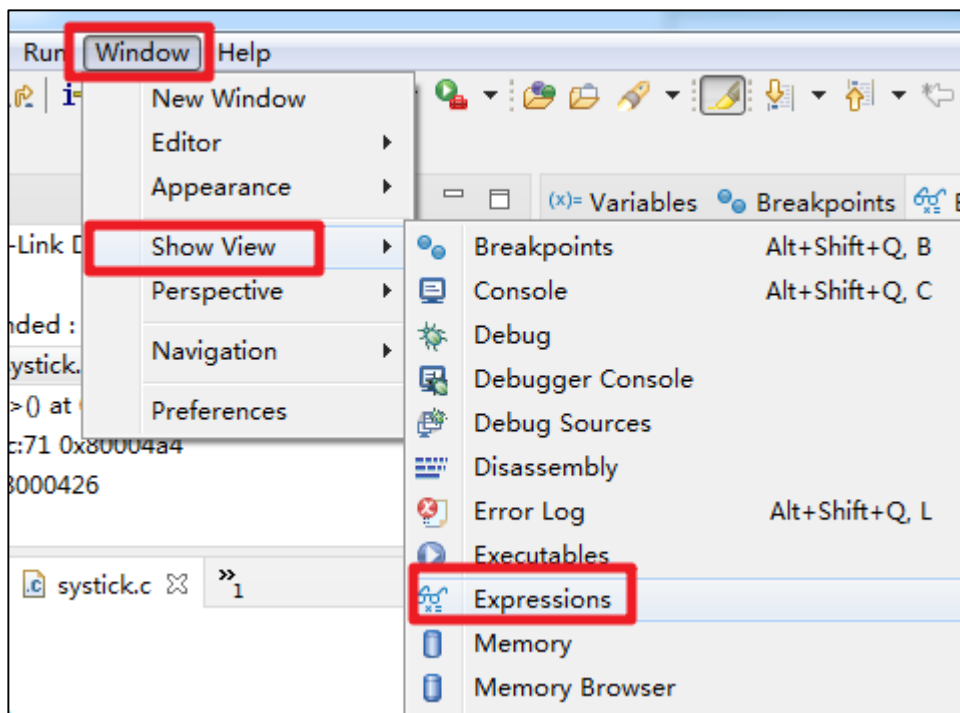
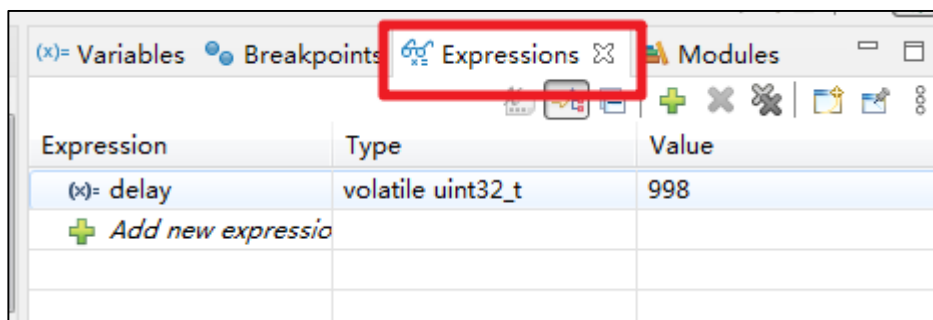


图 3-53. Expressions 窗口

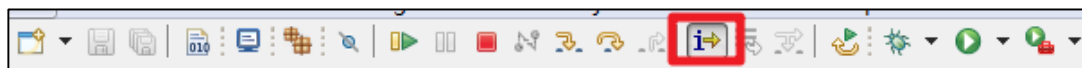


注意: Eclipse只有在代码停止运行的状态下才可以查看变量的值,暂时无法实时更新变量的值。

3.7.6. 反汇编窗口

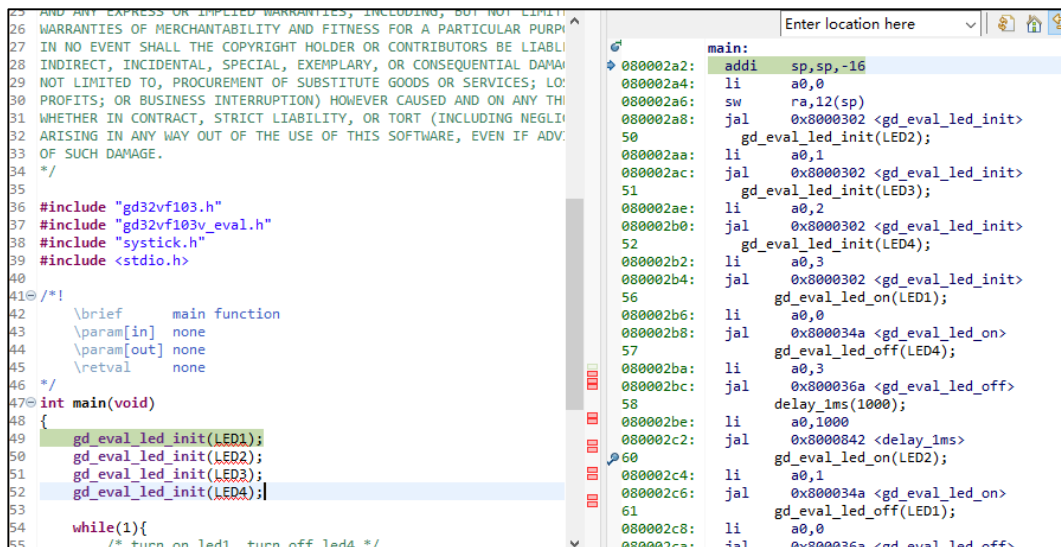
在调试工具栏选择Instruction Stepping Mode按钮, 打开反汇编窗口。

图 3-54. 打开反汇编窗口



在反汇编窗口, 可启用断点, 可单步执行汇编指令等。

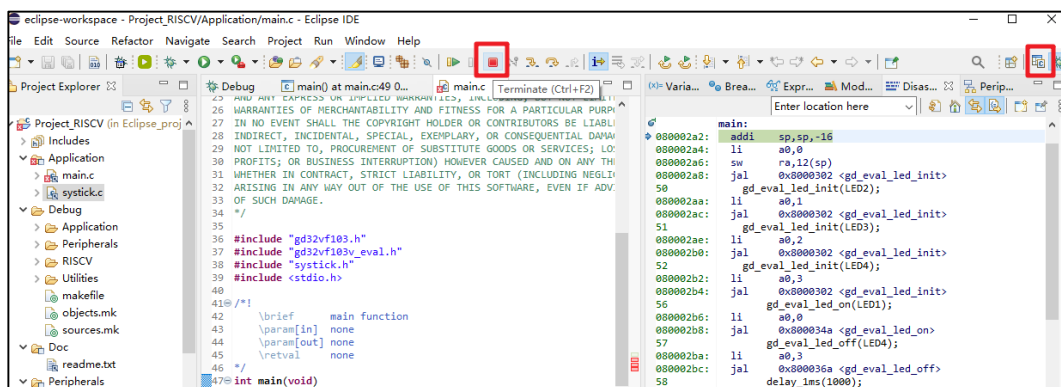
图 3-55. 反汇编窗口



3.7.7. 退出 Debug 视图

先点击“停止调试”按钮，再点击“C/C++”，即可进入工程视图。

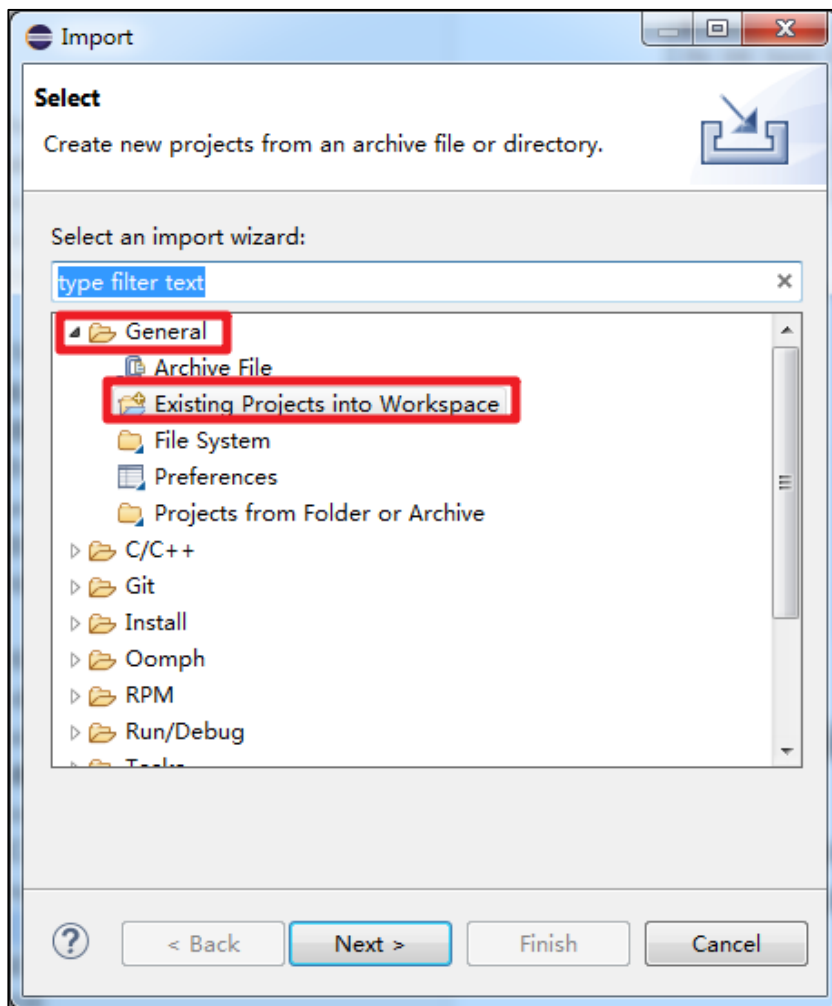
图 3-56. 退出 Debug 视图



4. 导入现有工程

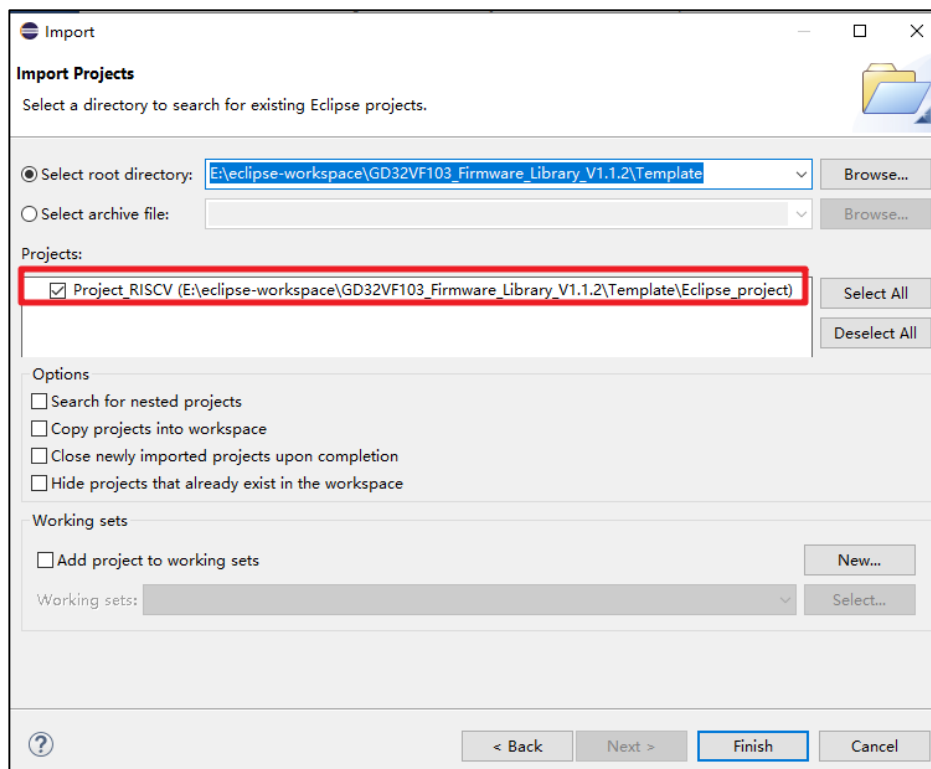
除新建工程外，也可直接导入已有Eclipse工程。在菜单栏中，点击File->Import，选择General->Existing Projects into Workspace导入已有工程，点击“Next”。

图 4-1. 导入现有工程-1



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。

图 4-2. 导入现有工程-2



5. 在 RAM 中调试

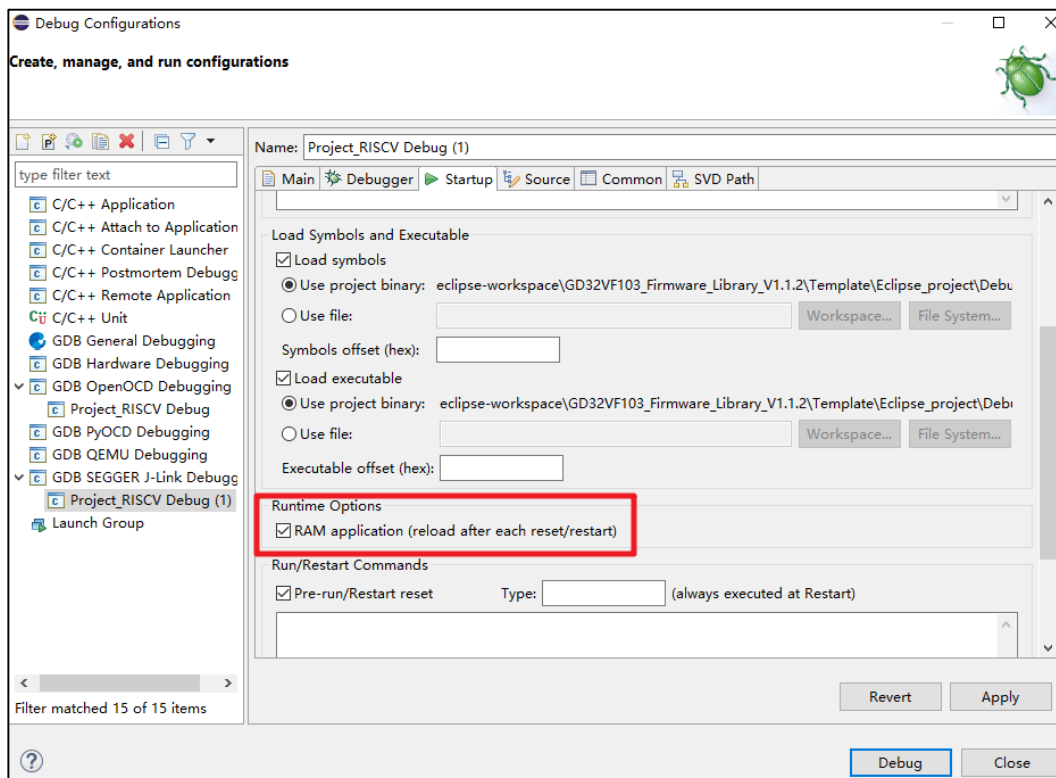
步骤1: 修改链接脚本并重新编译工程（建议先clean再build），例如修改如[图5-1. 在RAM中调试时ld文件memory map](#)。

图 5-1. 在 RAM 中调试时 ld 文件 memory map

```
5 MEMORY
6 {
7     /* Run in FLASH */
8 /*
9     flash (rxai!w) : ORIGIN = 0x08000000, LENGTH = 128k
10    ram  (wxa!ri) : ORIGIN = 0x20000000, LENGTH = 32k
11 */
12    /* Run in RAM */
13    flash (rxai!w) : ORIGIN = 0x20000000, LENGTH = 24k
14    ram  (wxa!ri) : ORIGIN = 0x20006000, LENGTH = 8K
15
16 }
17
```

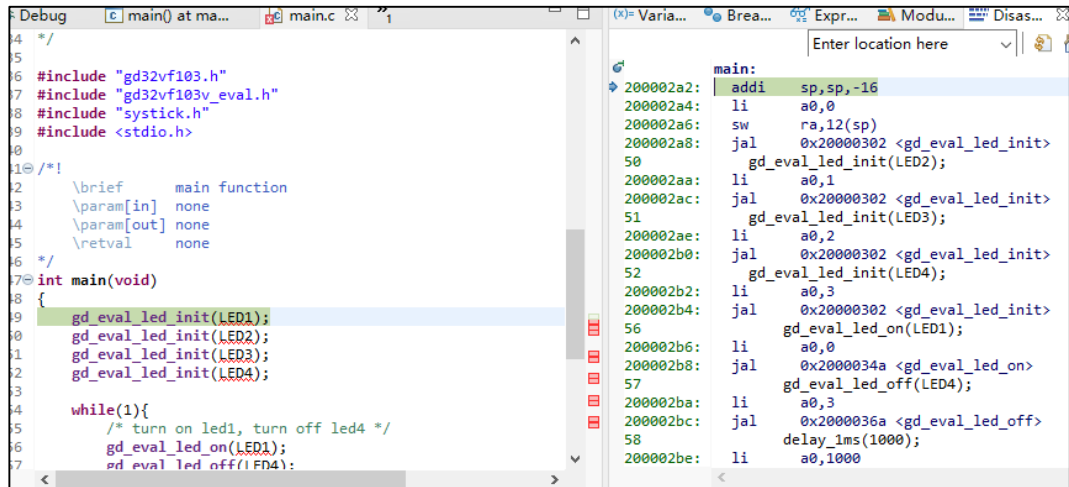
步骤2: 在 Debug Configurations->Startup 选项中，勾选 RAM application。

图 5-2. 在 RAM 中调试时 Debug Configurations



步骤3: 进入RAM中调试时的Debug视图，如下图。

图 5-3. 在 RAM 中调试时 Debug 视图



6. 使用 printf 打印

6.1. 使用步骤

步骤1: 添加如下_write函数定义。

```
ssize_t _write(int fd, const void* ptr, size_t len) {
    const uint8_t* current = (const uint8_t*) ptr;
    {
        for (size_t jj = 0; jj < len; jj++) {
            _put_char(current[jj]);

            if (current[jj] == '\n') {
                _put_char('\r');
            }
        }
        return len;
    }

    return _stub(EBADF);
}
```

步骤2: 将usart重定向到put_char函数。

```
int _put_char(int ch)
{
    usart_data_transmit(USART0, (uint8_t) ch);
    while ( usart_flag_get(USART0, USART_FLAG_TBE)== RESET){
    }

    return ch;
}
```

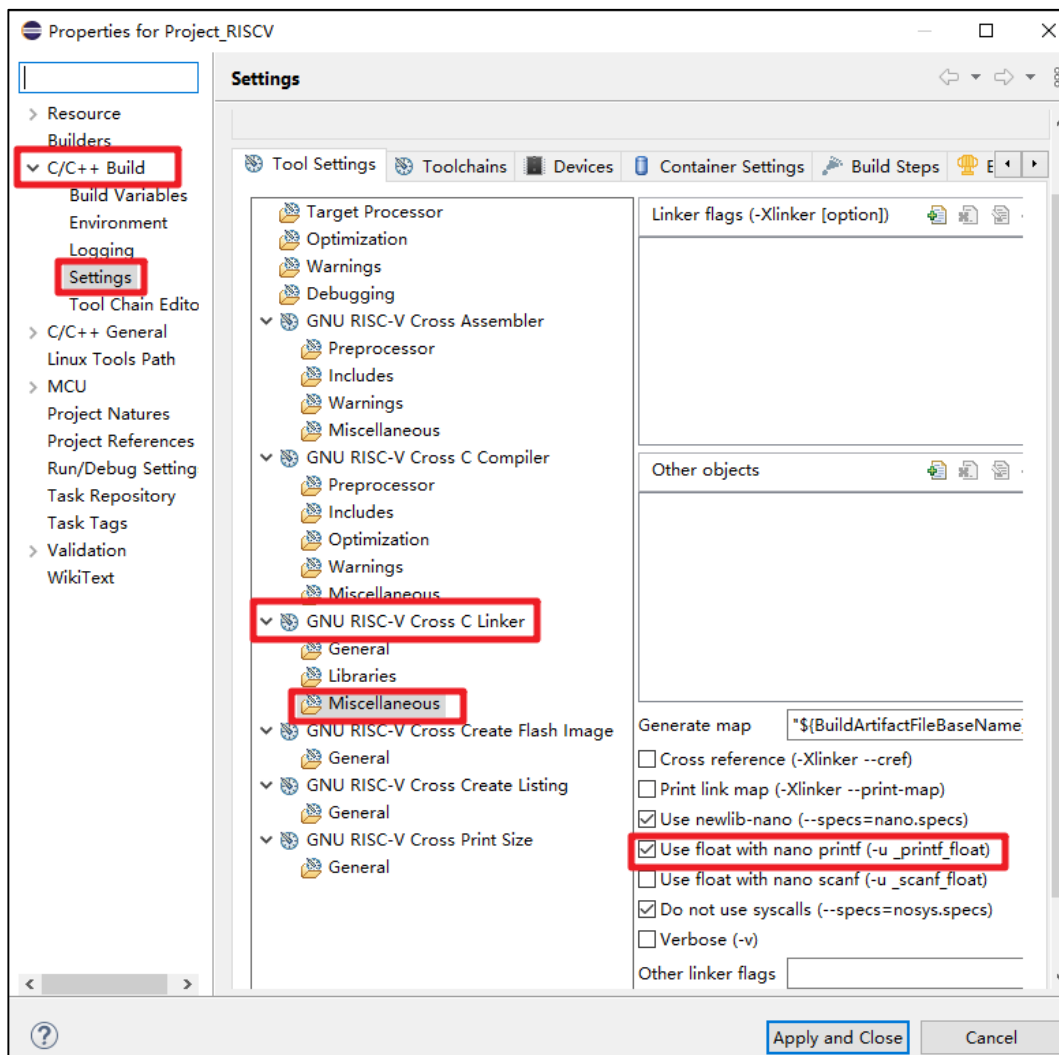
步骤3: 使用printf函数，即可正常打印。注意printf中需要加上\n用来刷新输出流。

```
printf("Running led test!\r\n");
```

6.2. 打印浮点数据配置

打印浮点数据配置：在工程Properties->C/C++ Build->Settings->Tool Settings->GNU RISC-V Cross C Linker->Miscellaneous中勾选-u _printf_float选项。

图 6-1. 打印浮点数据配置



注意： 1.使用printf时需要在打印的内容的末尾添加\r\n，例如printf("Running led test!\r\n")。2.在GCC中使用printf会大大增加代码的size，如果是对codesized大小要求很高的场合，不推荐使用printf。

7. 版本历史

表 7-1. 版本历史

版本号	描述	日期
1.0	首次发布	2022 年 6 月 20 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. 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. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.