GigaDevice Semiconductor Inc.

Arm® Cortex®-M3/4/23/33 32-bit MCU

应用笔记

AN041

目录

图索引

表索引

1. 开发环境

- 开发板：GD32 MCU开发板
- 调试器：J-Link V9/V10或GD-Link
- 操作系统：WIN7 64-bit OS
- IDE：eclipse-embedcpp-2021-03-R-win32-x86_64
- 交叉编译链：xpack-arm-none-eabi-gcc-10.2.1-1.1-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 V7.54b

2. 工程开发

2.1. 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 2-1. 新建 ARM C 工程

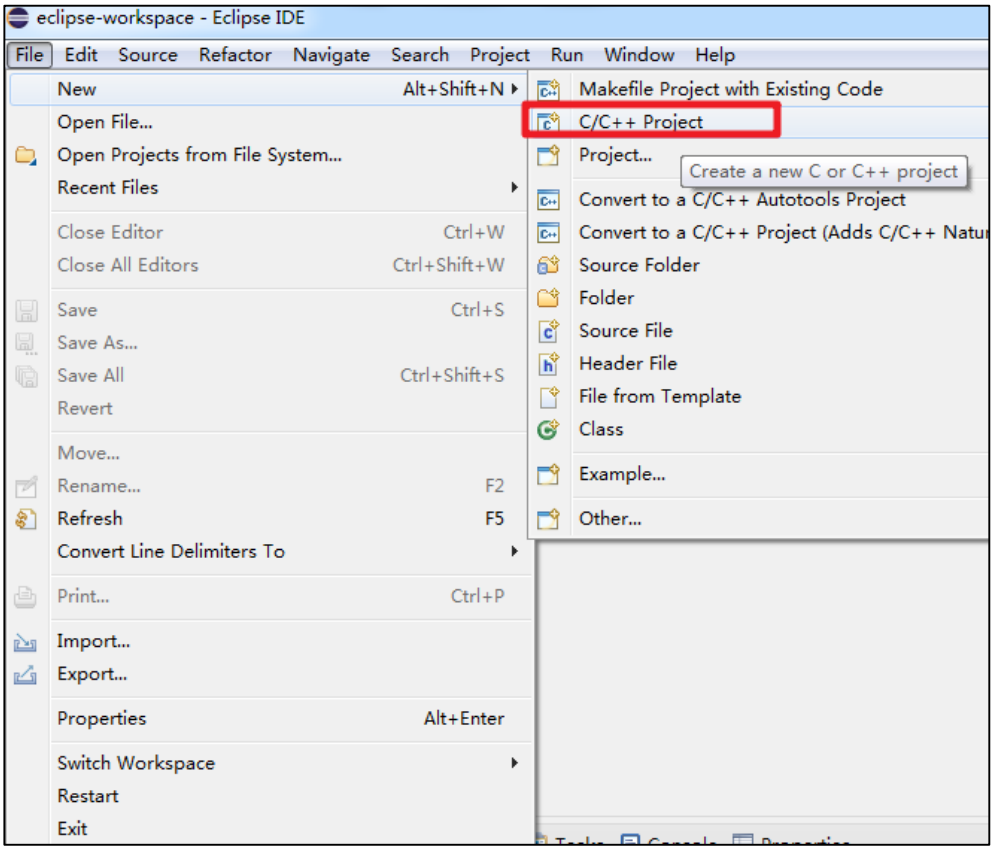
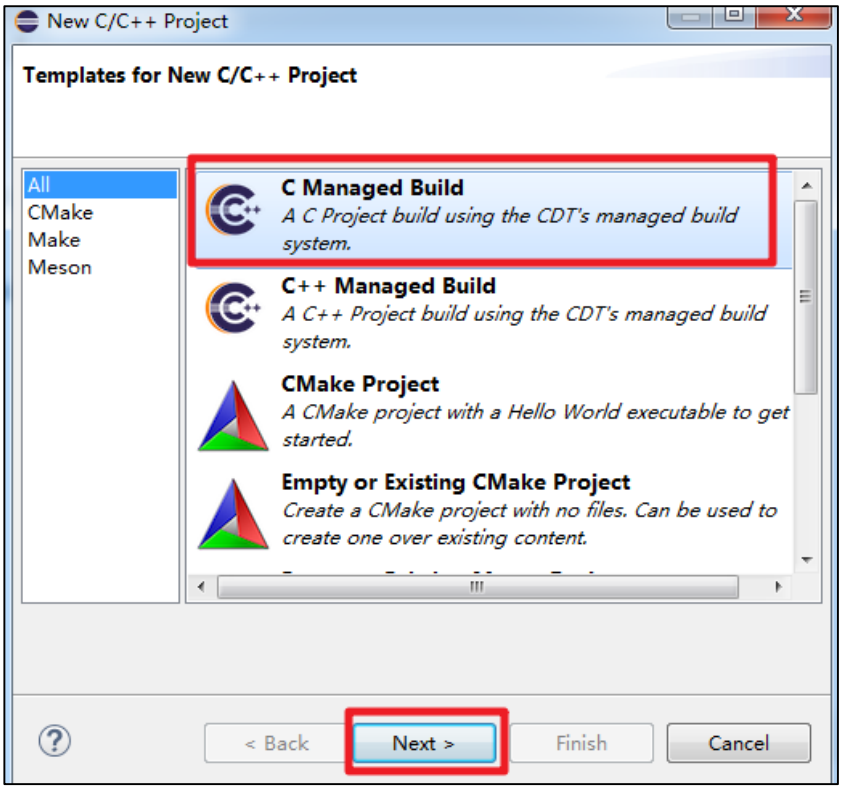
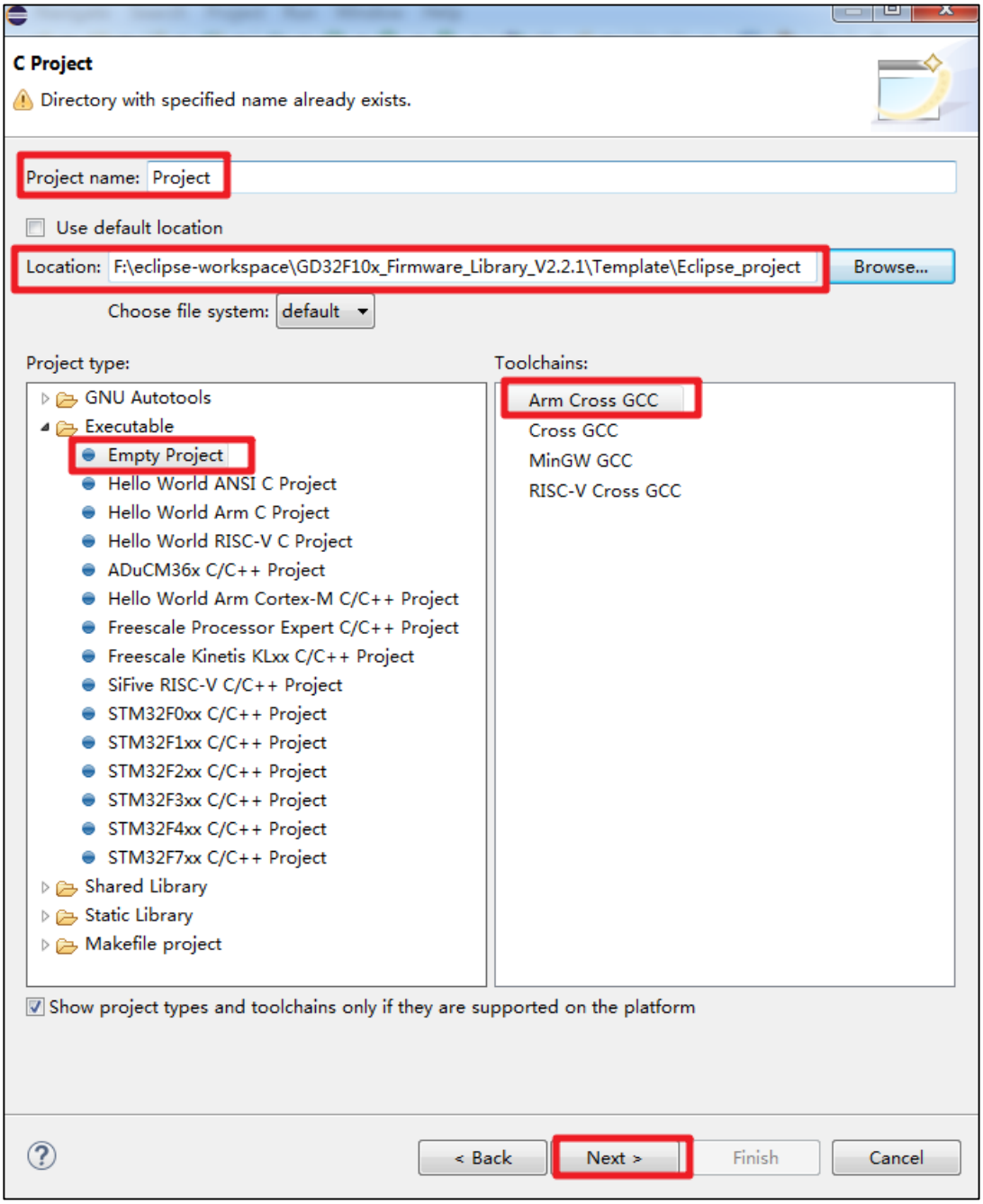


图 2-2. 选择 C Managed Build



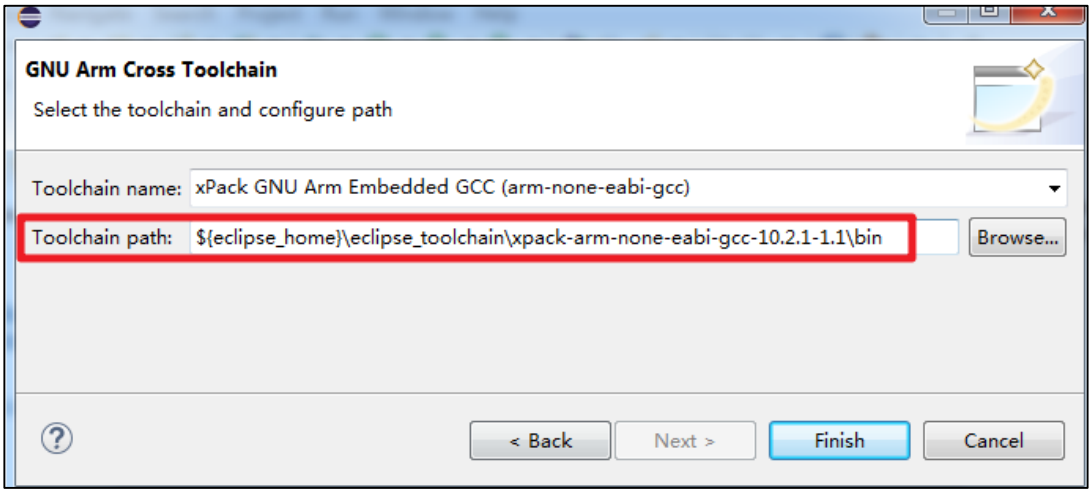
输入 Project name，配置工程类型，为了方便建议将工程放到 FW 目录下。编译链选择为 ARM Cross GCC。

图 2-3. 新建 ARM 工程名称及选择工程存放路径



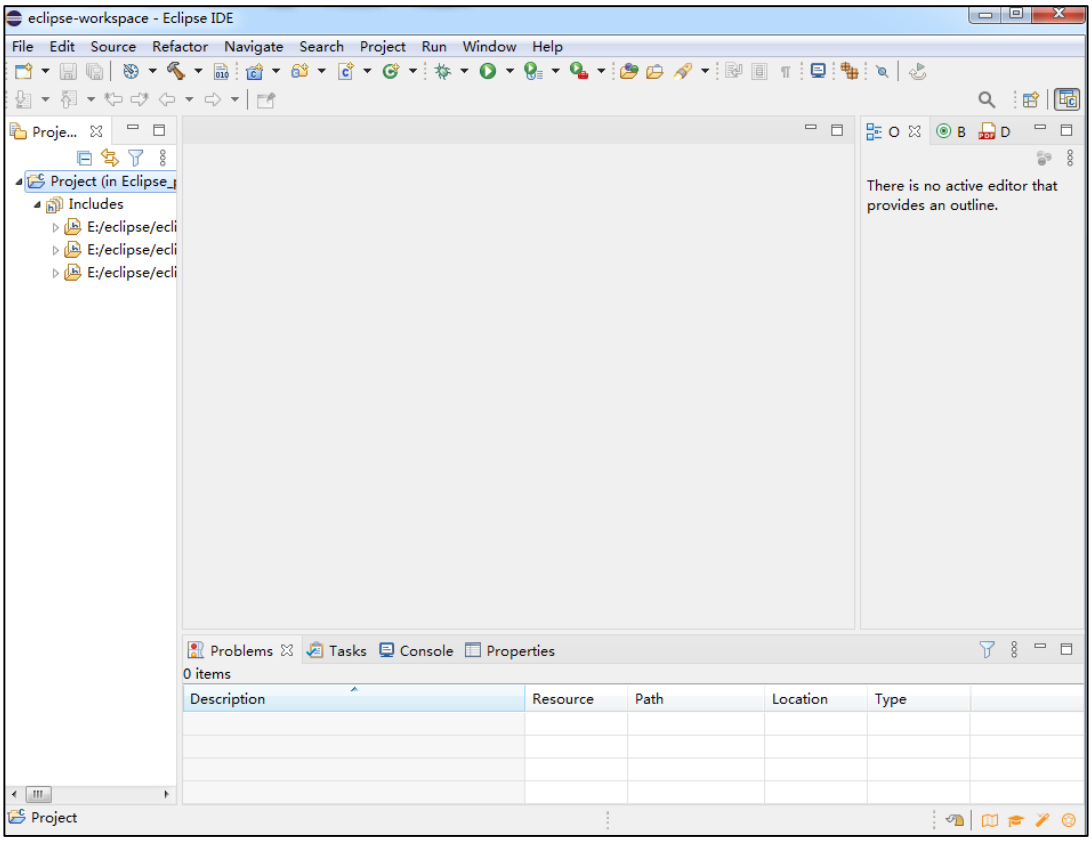
若Eclipse IDE已正确设置了ARM Toolchains Path，这里将会自动选择路径。若Eclipse IDE未设置ARM Toolchains Path，也可在这里选择到ARM Toolchains绝对路径。

图 2-4. 选择 ARM 交叉编译工具链



点击"Finish"，直到显示界面如*图2-5. 工程视图*所示。至此，完成Project的建立。

图 2-5. 工程视图

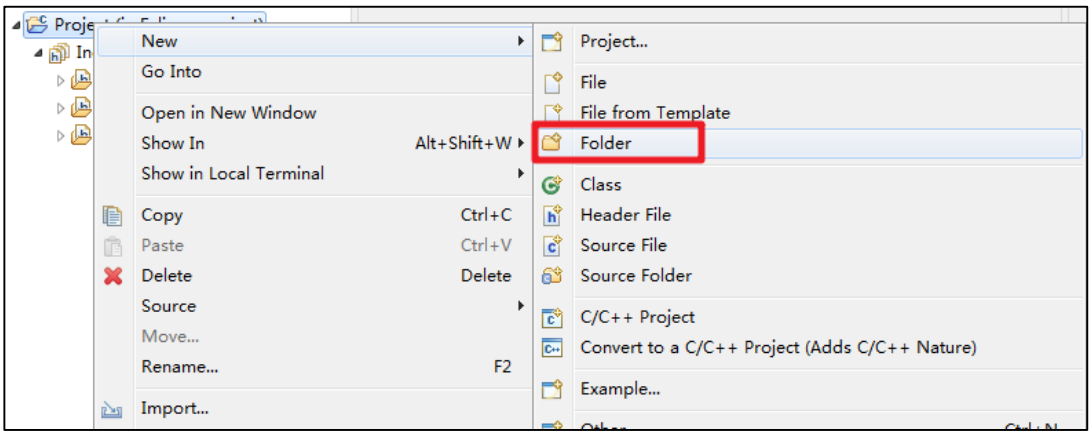


2.2. 新建工程文件夹并添加文件

2.2.1. 手动建立文件夹及添加文件

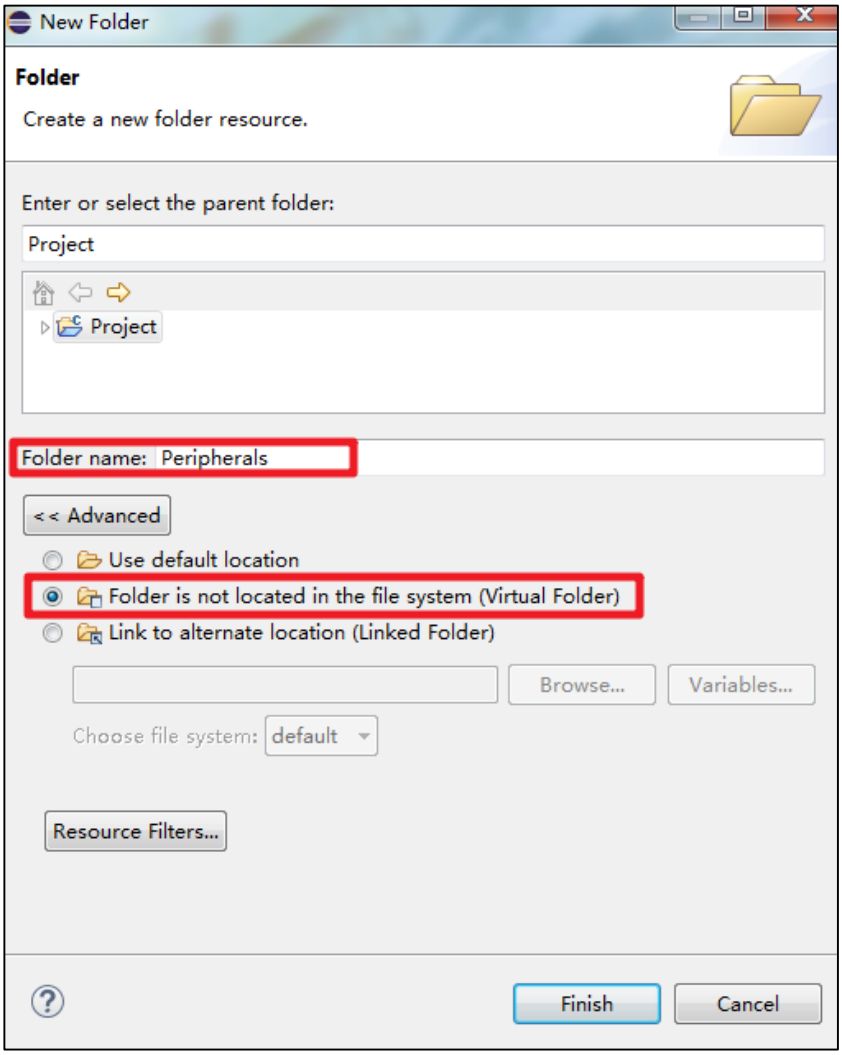
右击工程名，选择new->Folder。

图 2-6. 新建工程文件夹



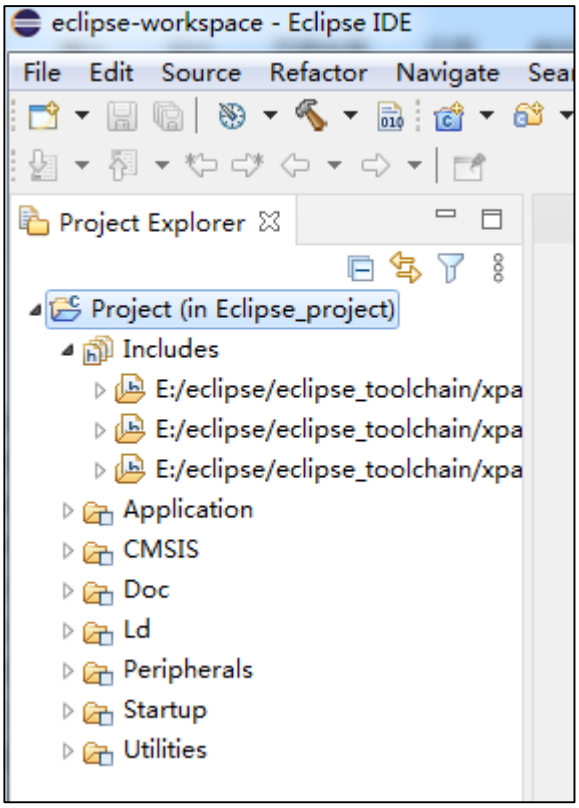
建立虚拟文件夹Peripherals。

图 2-7. 建立虚拟子文件夹



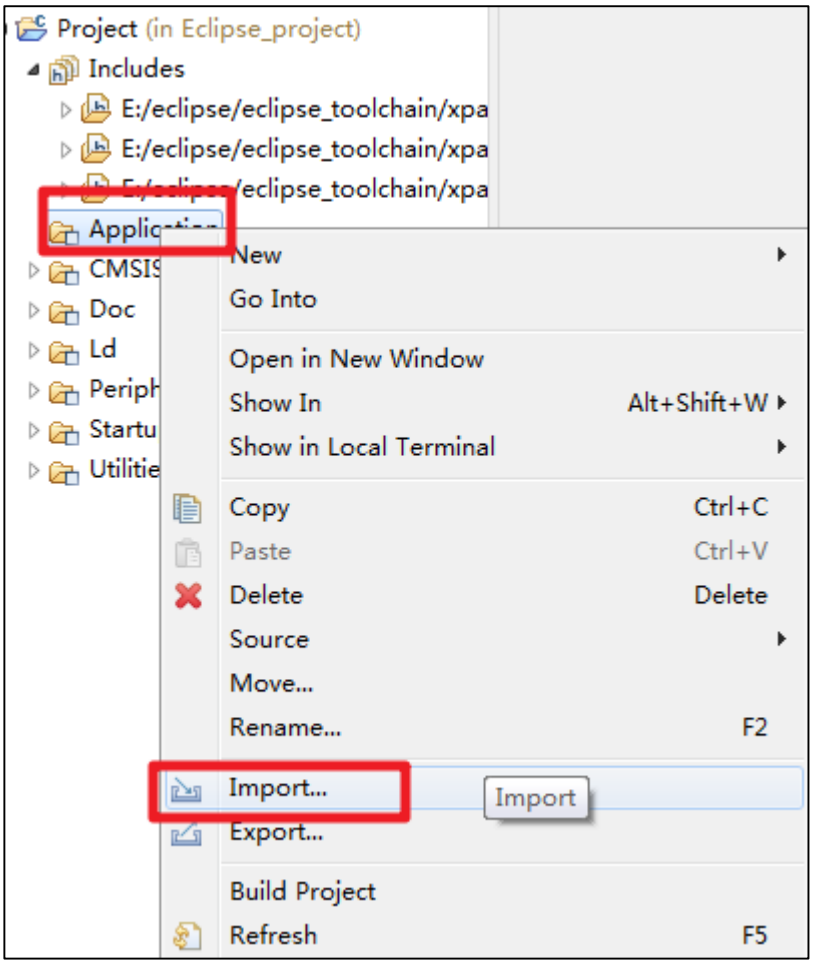
同样方法建立Application、CMSIS、Doc、Ld、Startup和Utilities文件夹。

图 2-8. ARM 工程视图



右击Application，选择Import选项，可导入文件。

图 2-9. 添加文件



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

图 2-10. 选择 Import 文件

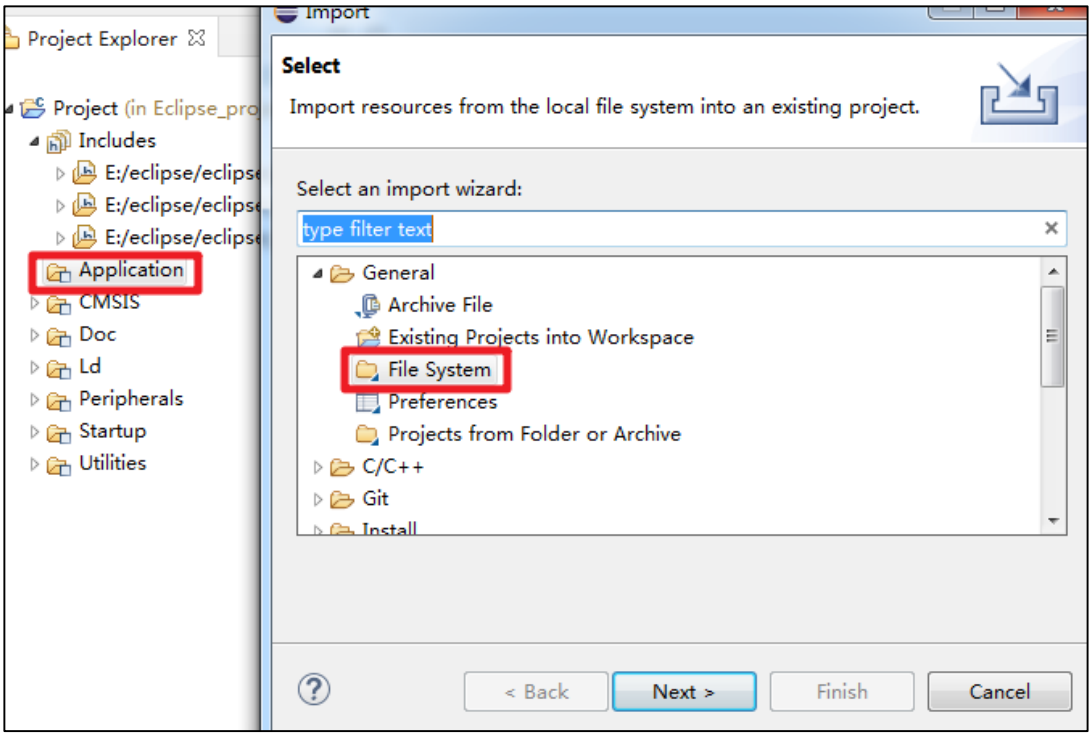
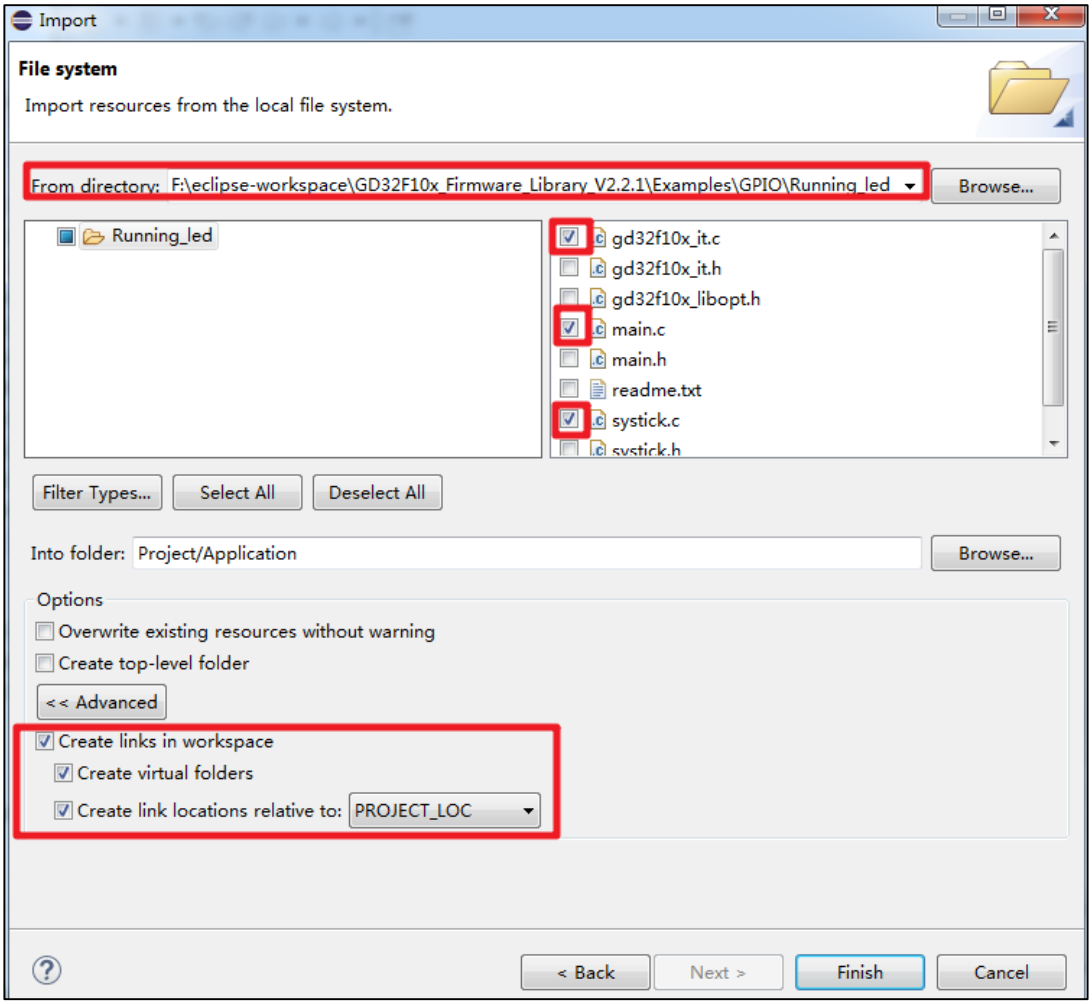


图 2-11. Import Application 文件夹内文件



同理，将所需文件导入CMSIS、Doc、Ld、Peripherals、Startup及Utilities文件夹。

图 2-12. Import CMSIS 文件夹文件

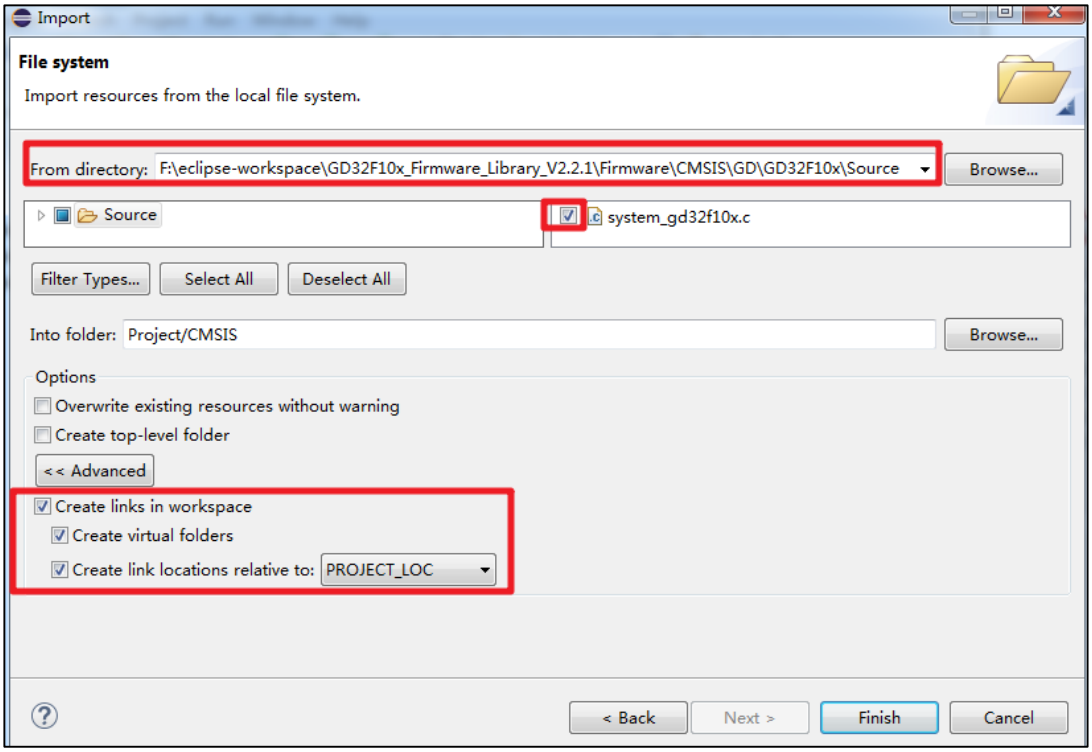


图 2-13. Import Doc 文件夹文件

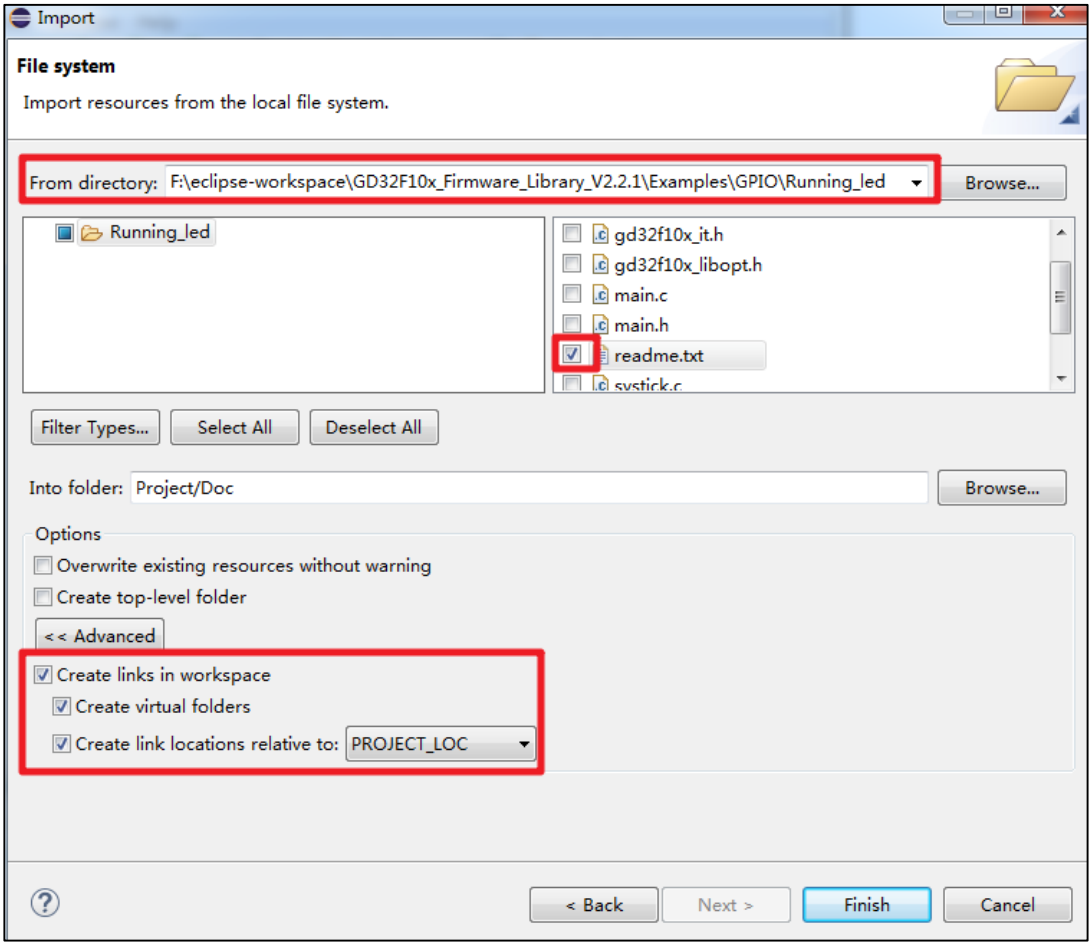


图 2-14. Import Ld 文件夹文件

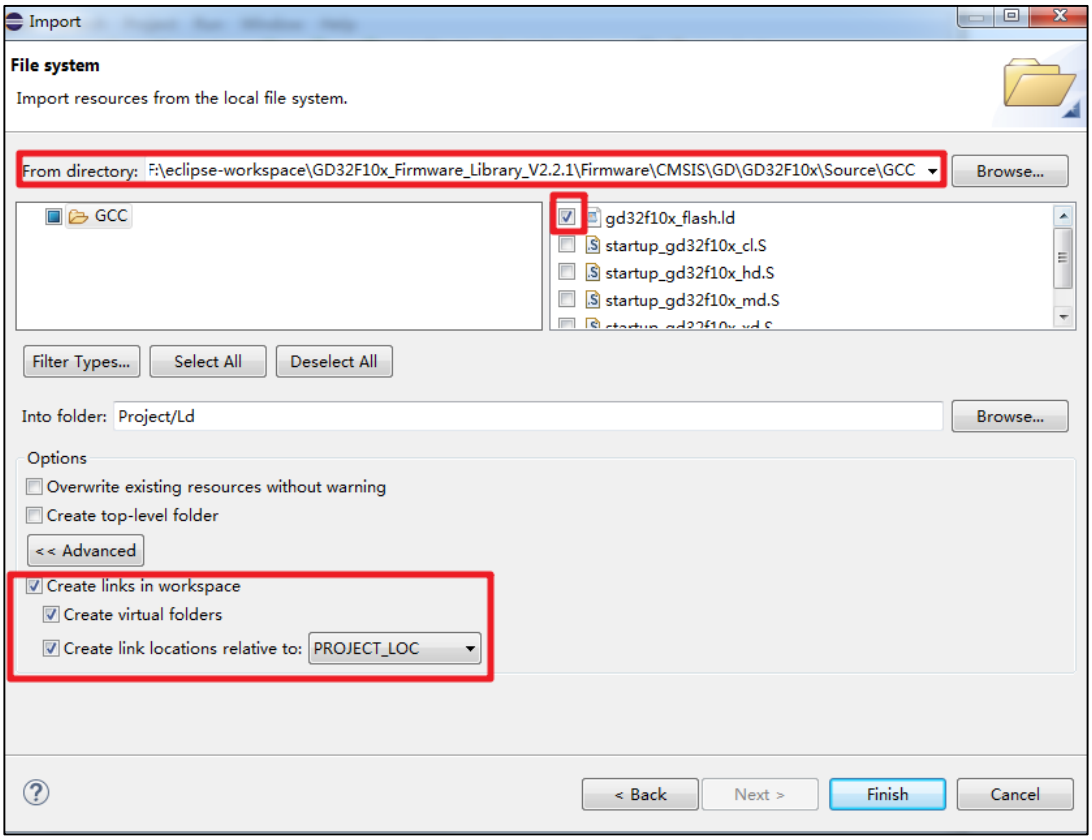


图 2-15. Import Peripherals 文件夹文件

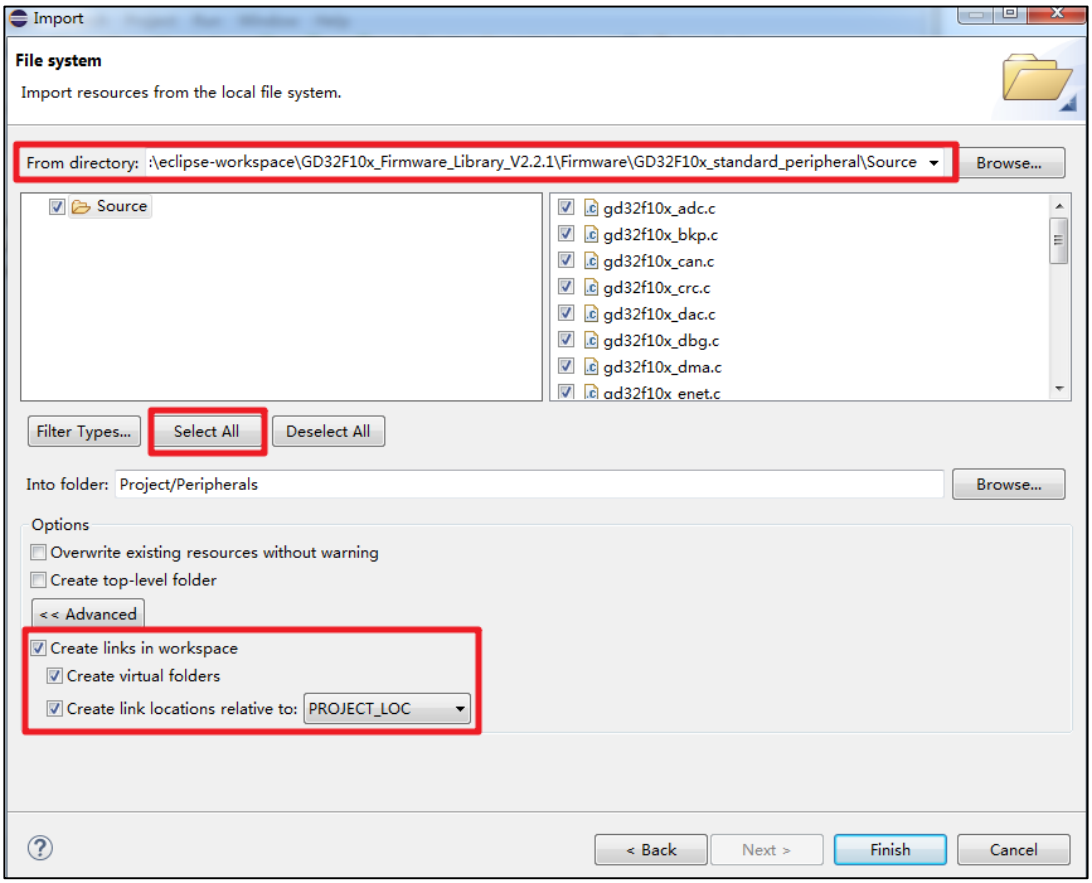


图 2-16. Import Startup 文件夹文件

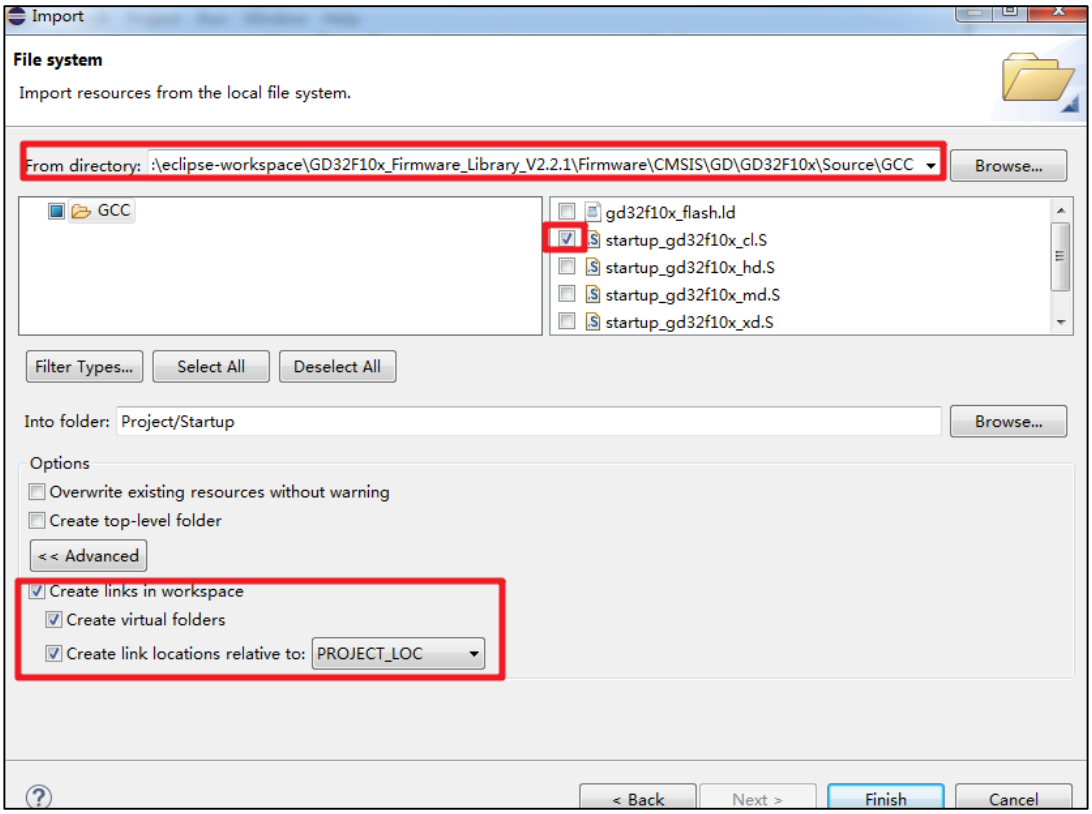


图 2-17. Import Utilities 文件夹文件

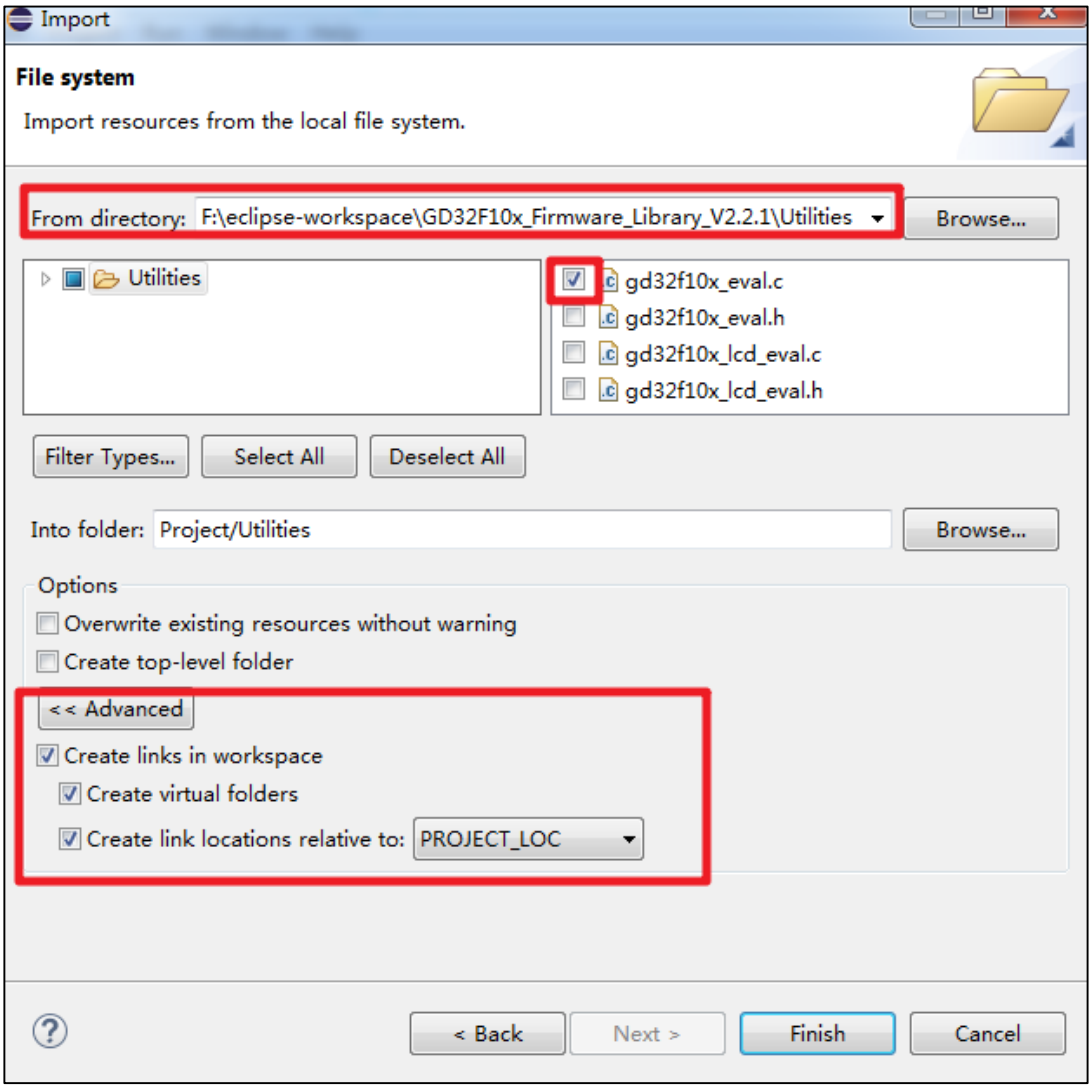
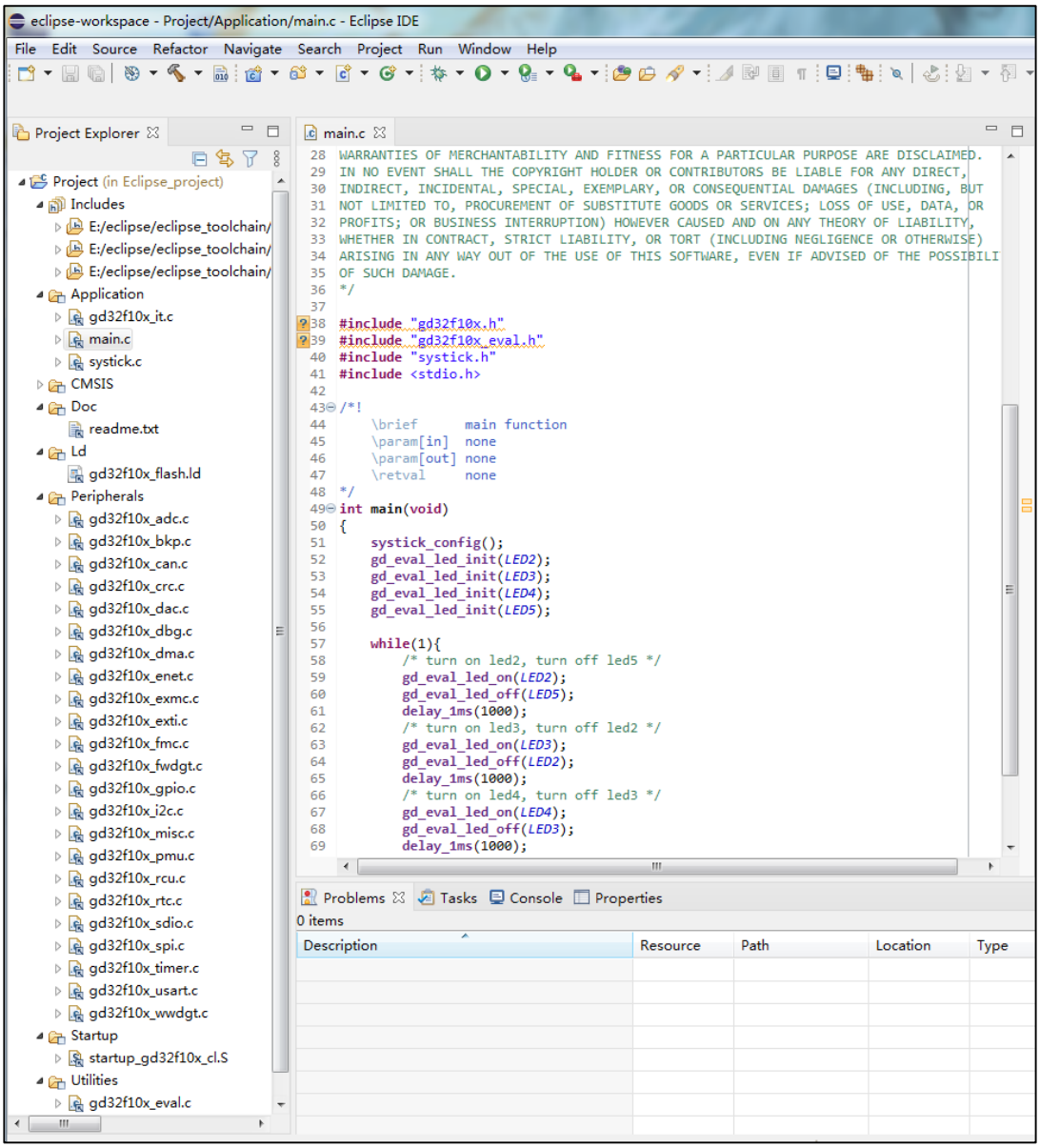


图 2-18. 最终 ARM 工程视图



2.2.2. “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 2-19. 工程文件夹结构

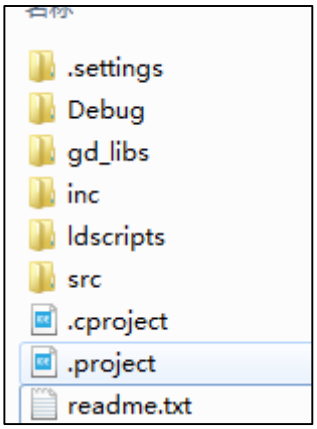


图 2-20. Refresh 工程

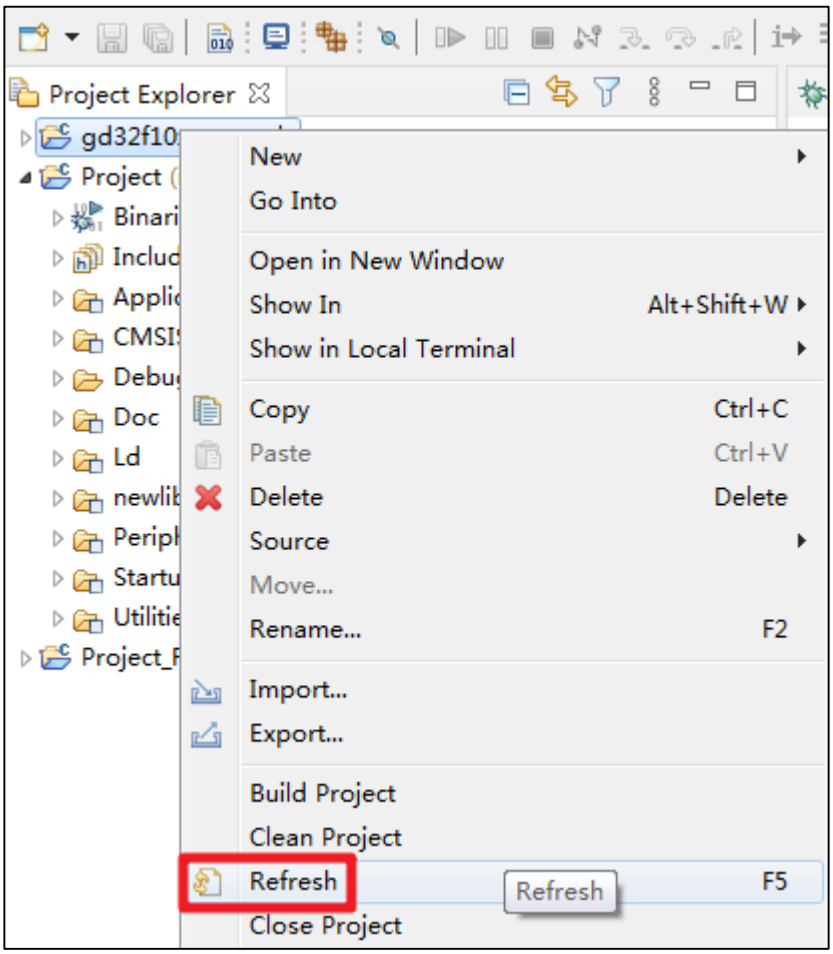
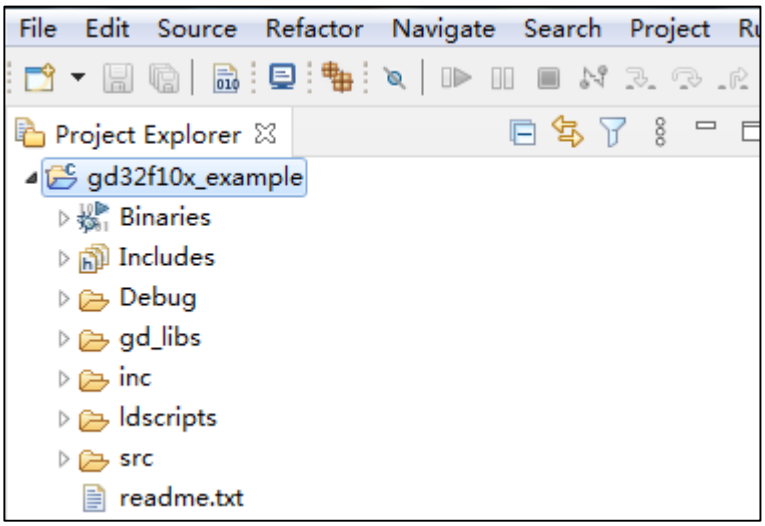


图 2-21. Eclipse IDE 内工程结构

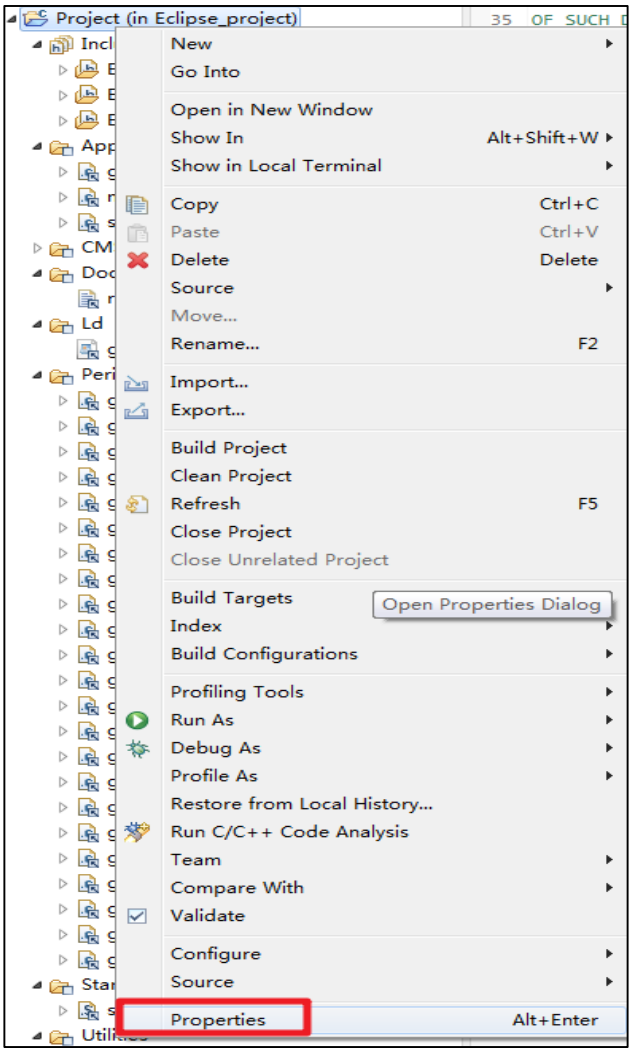


注意：“Refresh”方式建立的文件及文件夹，均是真实存在的，并且一旦在Eclipse IDE中删除其中的某个文件，该文件就会直接从磁盘中删除了。

2.3. 工程配置

右击工程，选择工程属性Properties选项打开。

图 2-22. 工程属性配置

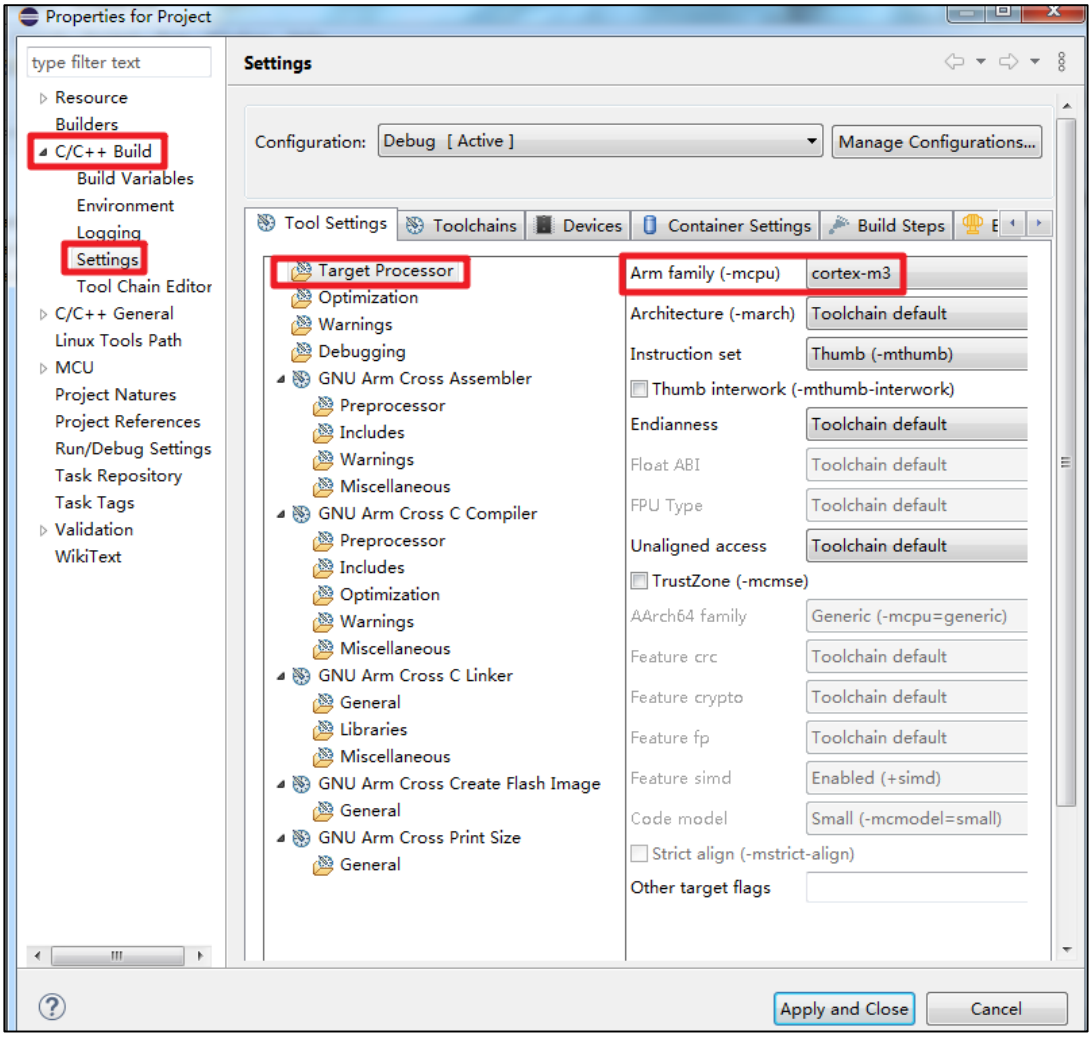


2.3.1. Target Processor 配置

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cortex-m3、cortex-m4、cortex-m23或cortex-m33。在本例中，选择cortex-m3。

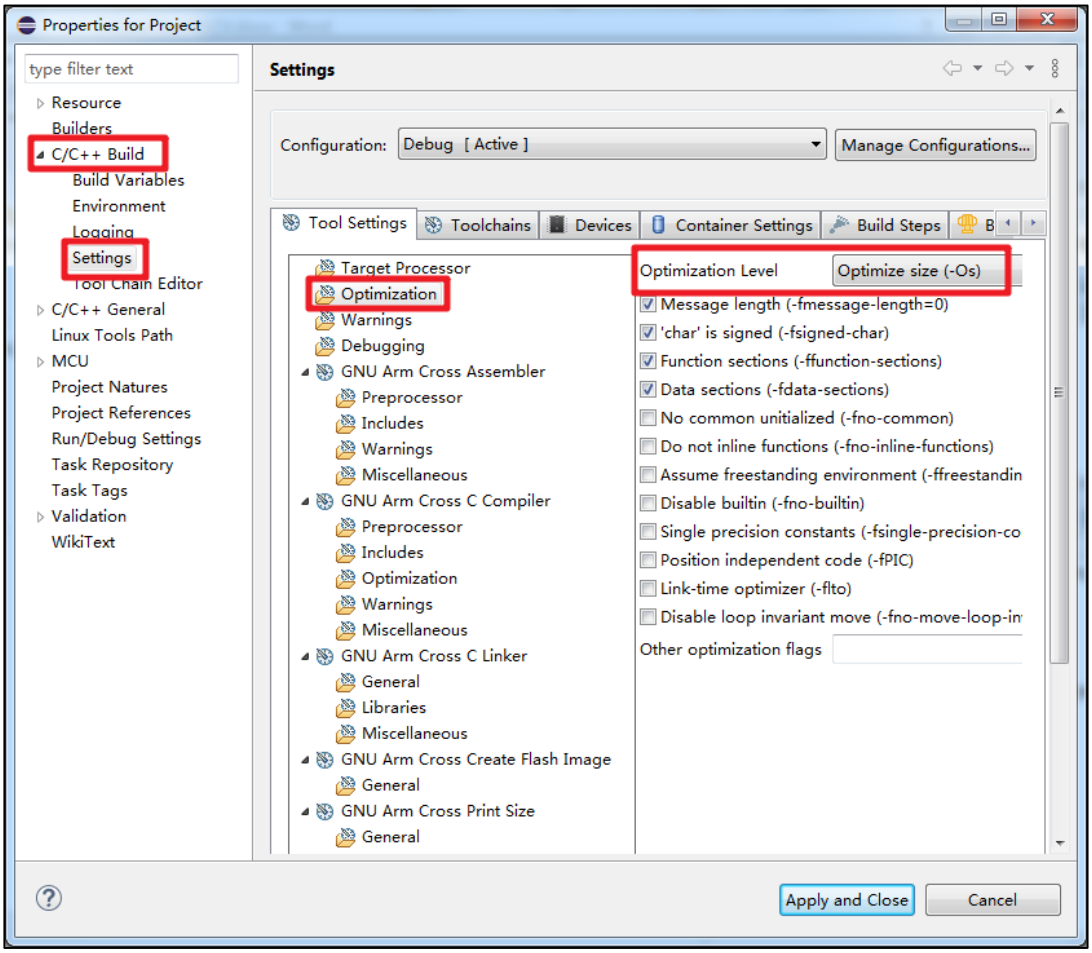
图 2-23. Target Processor 配置



2.3.2. 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 2-24. Optimization 配置

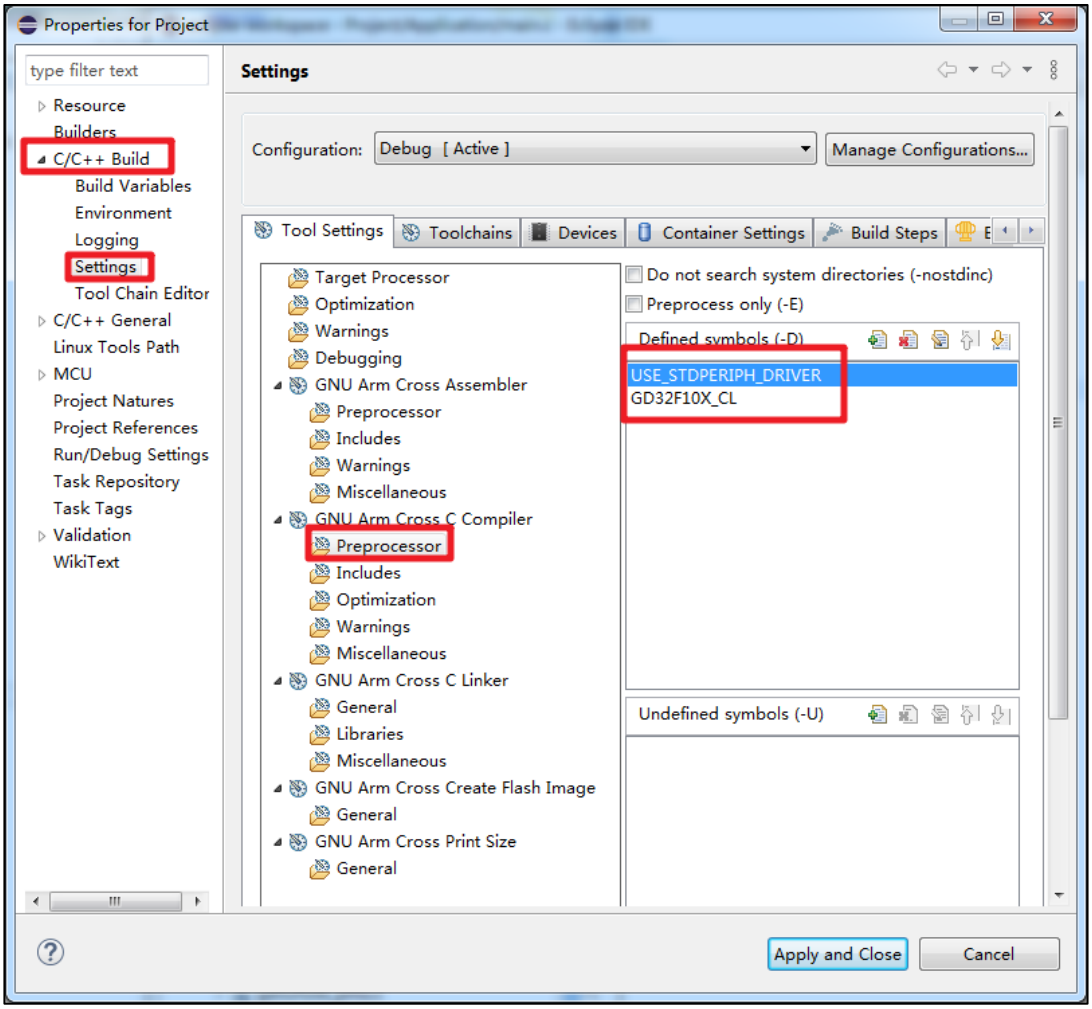


2.3.3. GNU Arm Cross C Compiler 配置

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”选项中添加USE_STDPERIPH_DRIVER和GD32F10X_CL预编译宏。

图 2-25. GNU Arm 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/CMSIS/GD/GD32F10x/Include"

"${ProjDirPath}/../../Firmware/CMSIS"

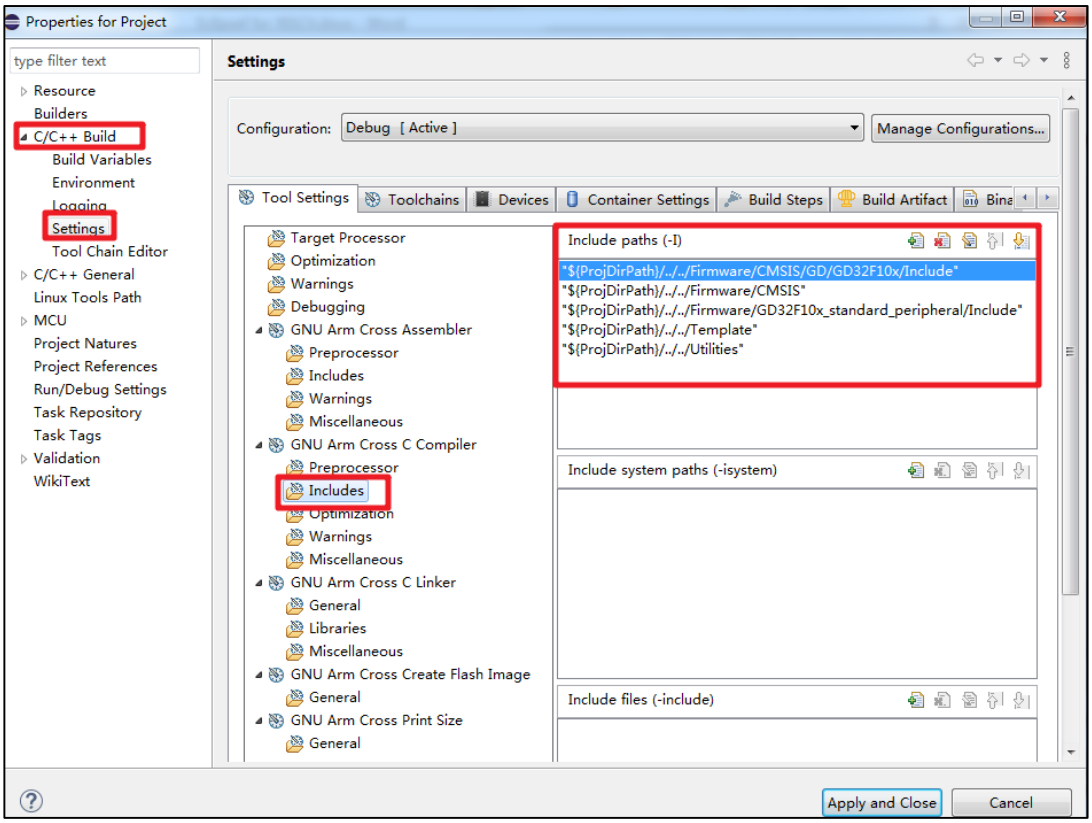
"${ProjDirPath}/../../Firmware/GD32F10x_standard_peripheral/Include"

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

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

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 2-26. GNU Arm Cross C Compiler -> Includes 配置



2.3.4. 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。

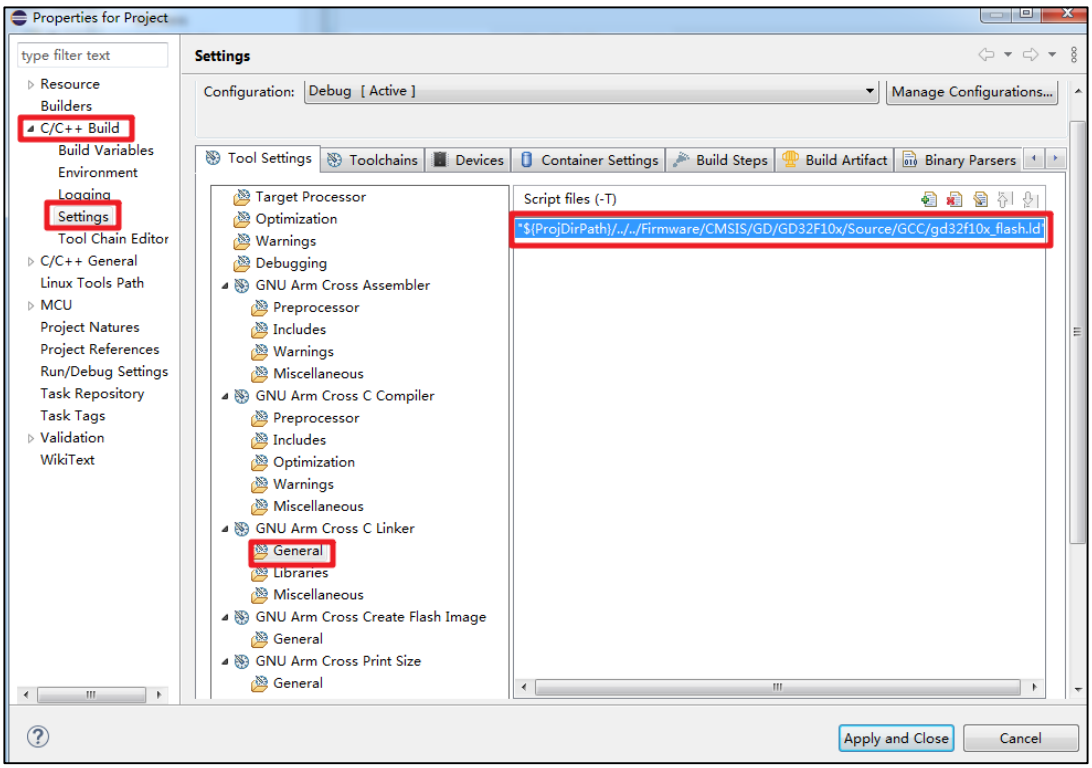
在“General ->Script files”选项中添加：

"${ProjDirPath}/../../Firmware/CMSIS/GD/GD32F10x/Source/GCC/gd32f10x_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。

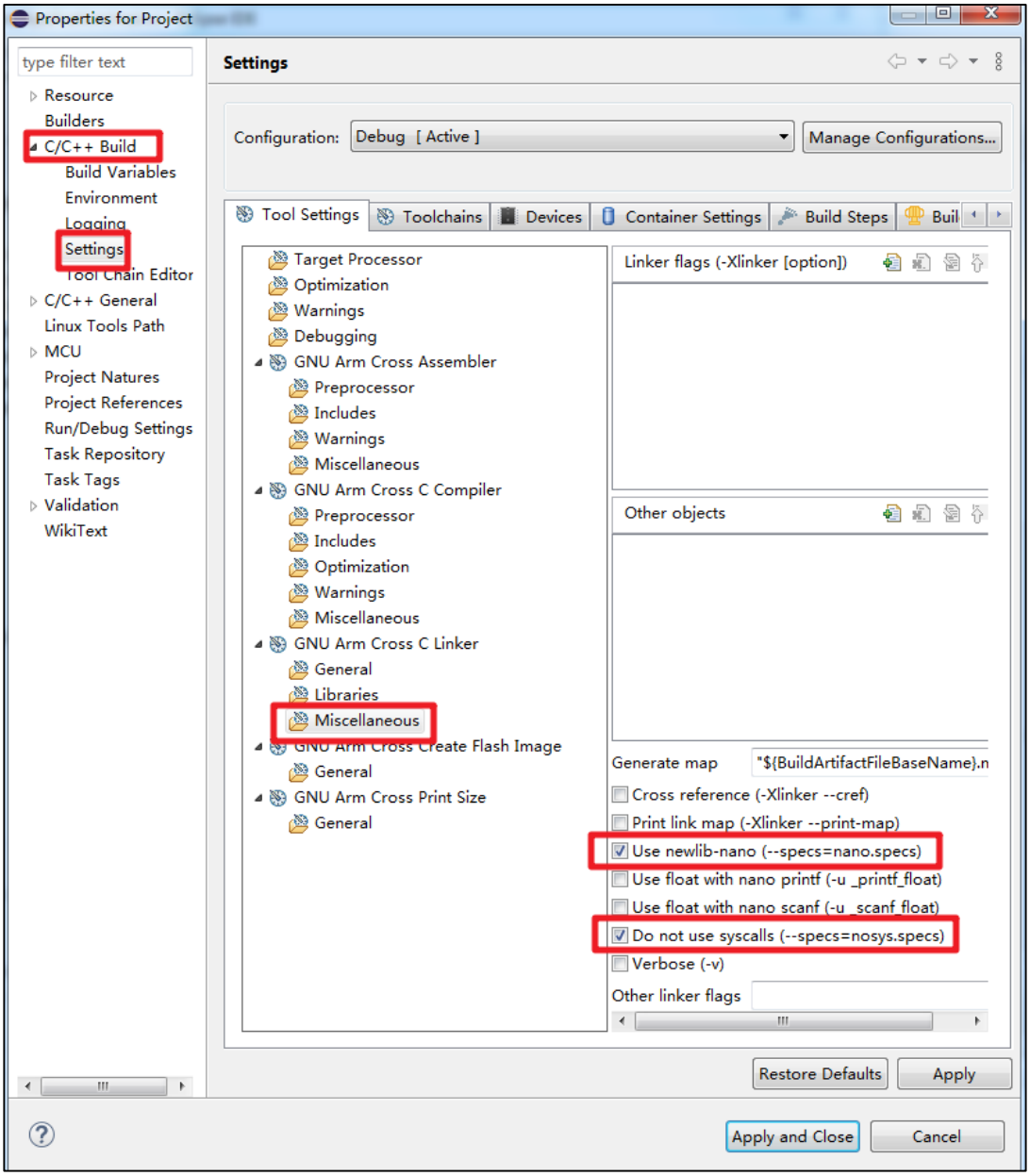
注意：本例中添加的ld文件路径为相对路径。这里也可以直接添加绝对路径。

图 2-27. GNU Arm 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 2-28. GNU Arm Cross C Linker -> Miscellaneous 配置

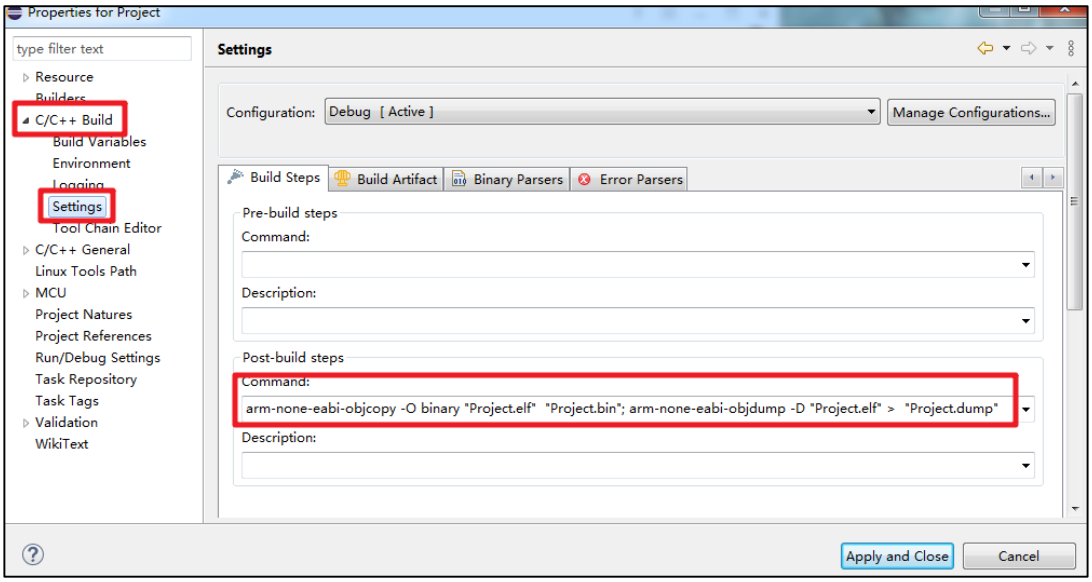


2.3.5. Build Steps 配置-生成 bin 文件

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

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

图 2-29. Build Steps 配置

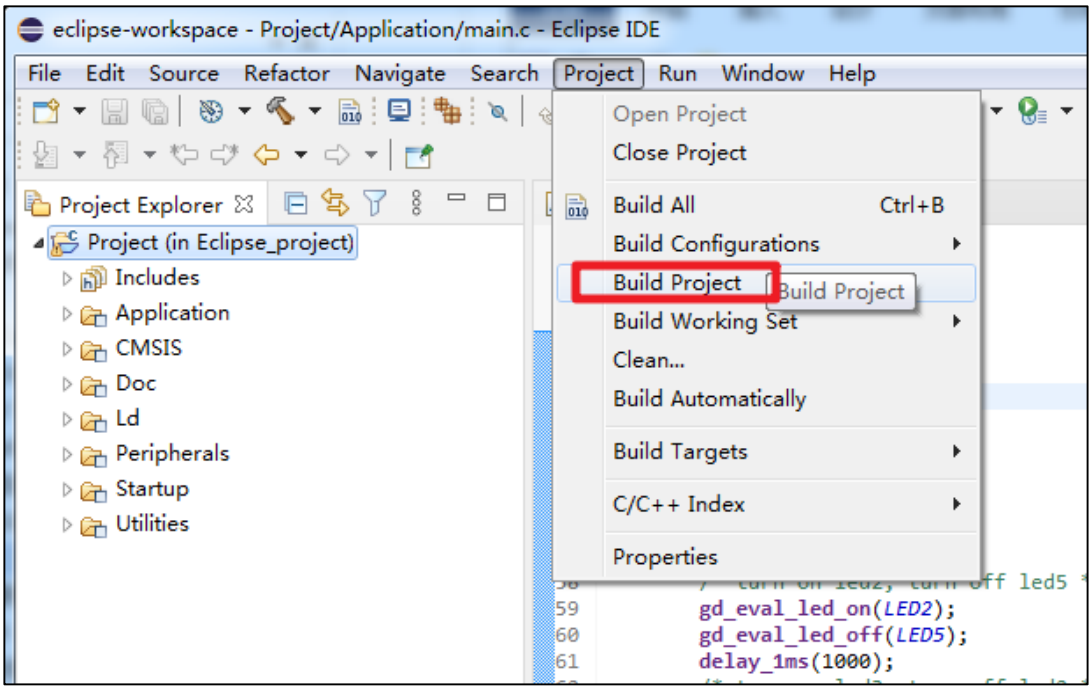


2.4. 编译工程

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。

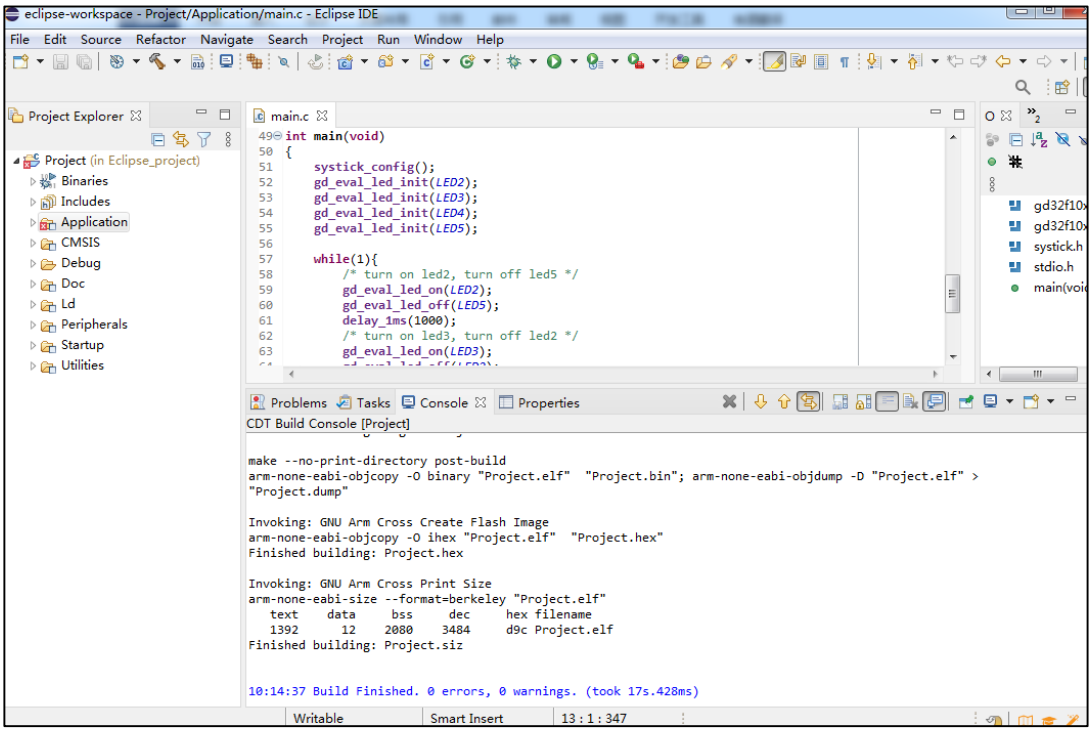
图 2-30. Build 工程



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 2-31. Build ARM 工程完成

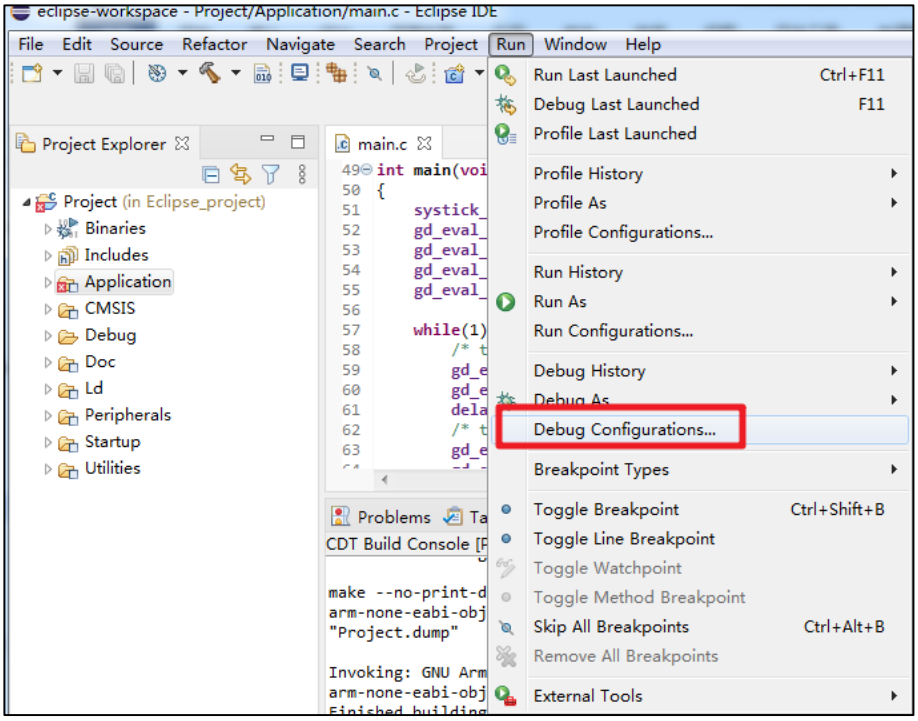


2.5. 使用 J-Link 下载及调试工程

2.5.1. Debug 配置界面

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

图 2-32. 进入 Debug Configurations 界面

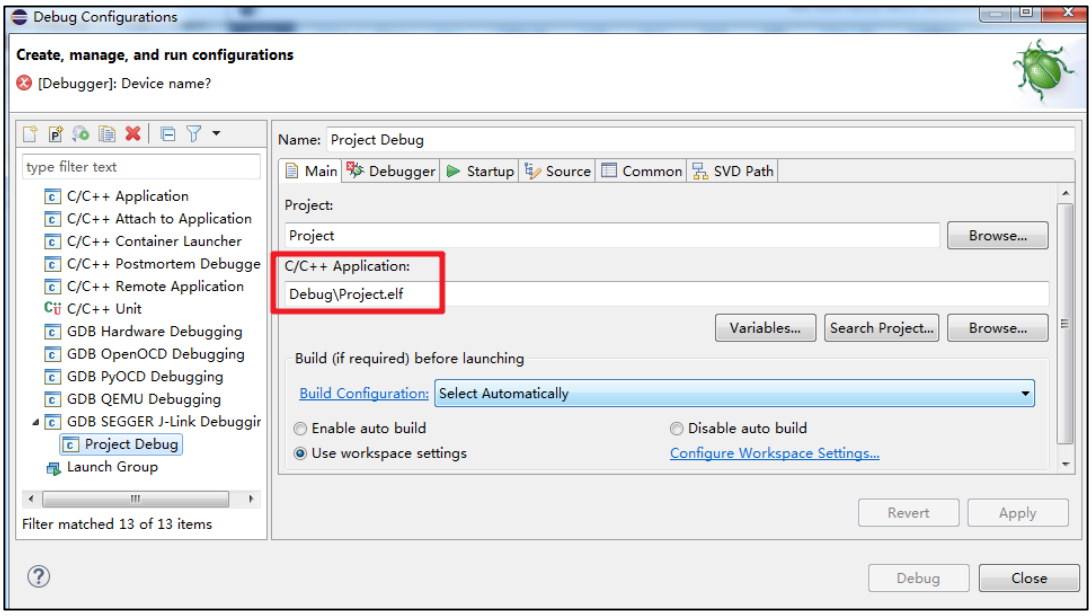


使用J-Link GDBServerCL作为GDB Server，使用GCC工具链中的GDB工具作为GDB Client。

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

2.5.2. Main 选项卡

图 2-33. GDB SEGGER J-Link Debugging-Main 选项卡



在Main选项卡中，选择当前工程，一般会自动添加当前工程下的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。

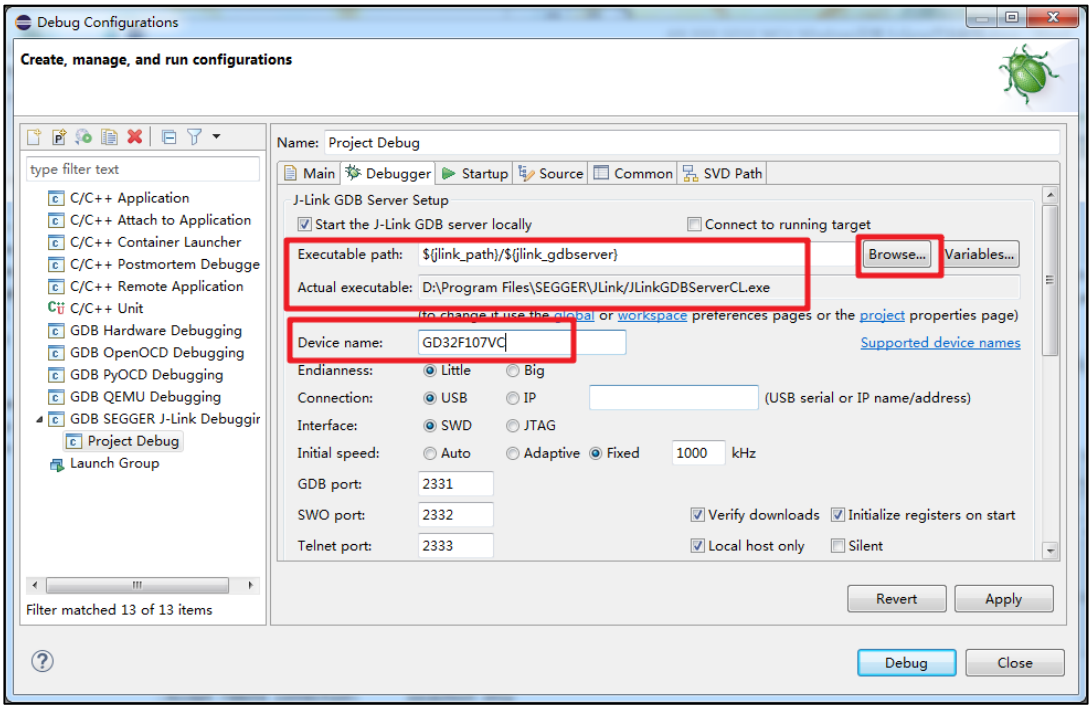
2.5.3. Debugger 选项卡

在Debugger选项卡中，填写目标芯片型号Device name，本例中为GD32F107VC。

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。

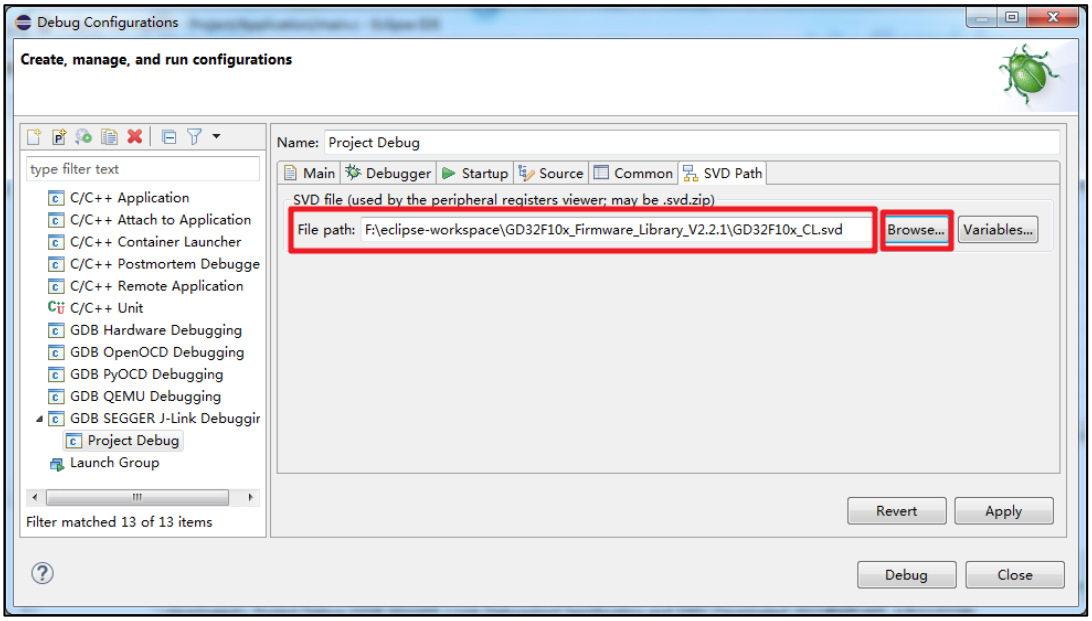
图 2-34. GDB SEGGER J-Link Debugging-Debugger 选项卡



2.5.4. SVD Path 选项卡

在SVD Path选项卡，选择目标芯片所需的SVD文件。

图 2-35. GDB SEGGER J-Link Debugging-SVD Path 选项卡

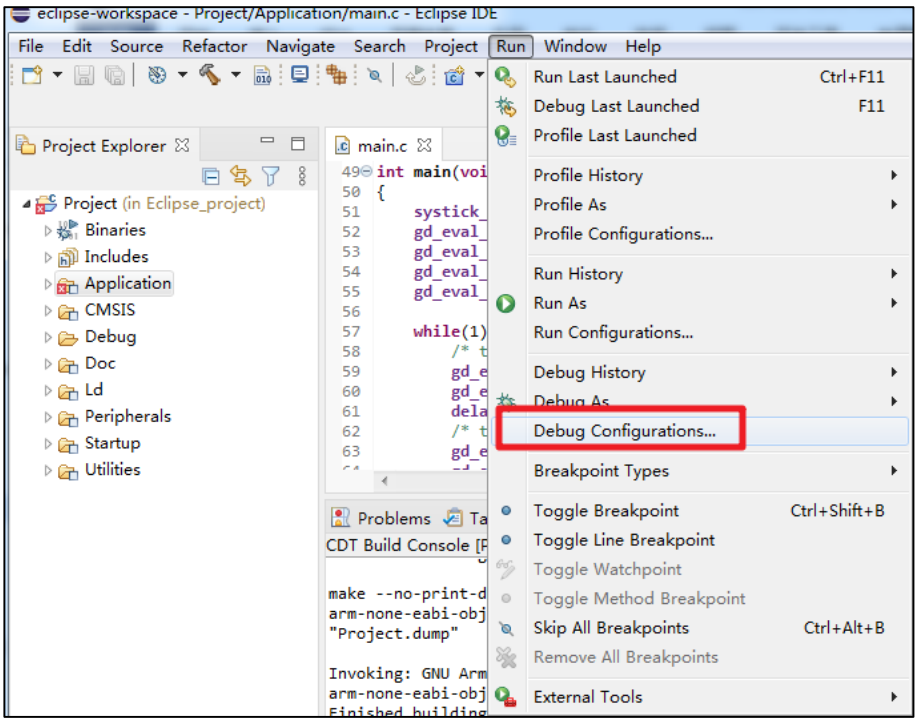


2.6. 使用 GD-Link 下载及调试工程

2.6.1. Debug 配置界面

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

图 2-36. 进入 Debug Configurations 界面

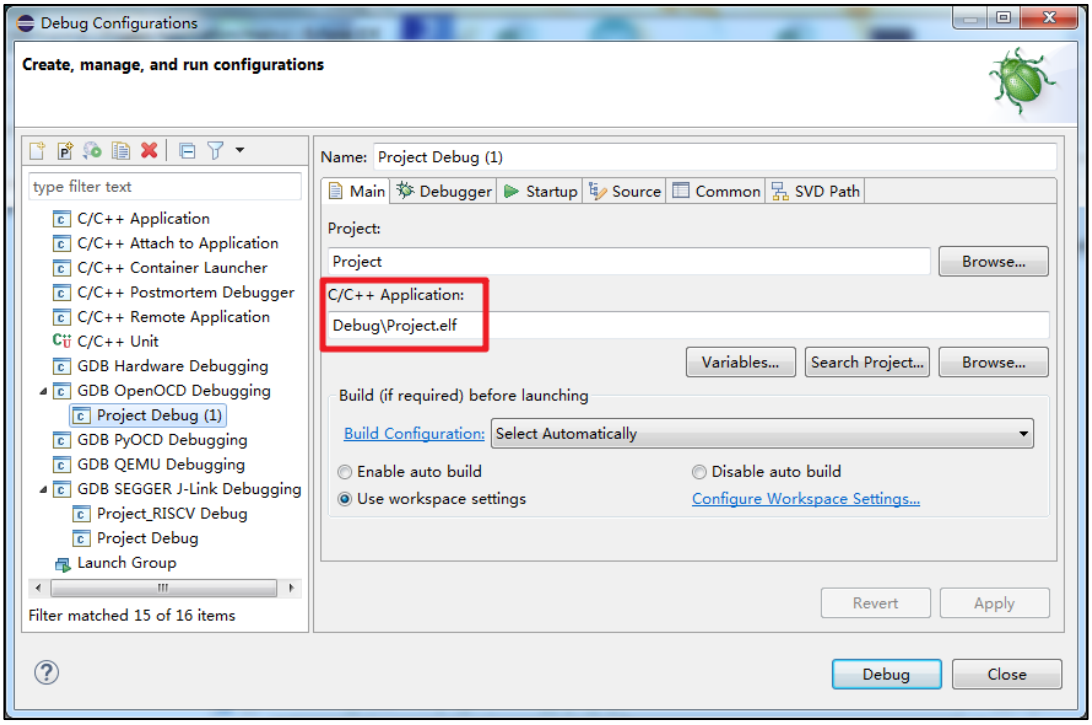


使用OpenOCD作为GDB Server，使用GCC工具链中的GDB工具作为GDB Client。双击GDB

OpenOCD Debugging，新建一套OpenOCD的配置选项。

2.6.2. Main 选项卡

图 2-37. GDB OpenOCD Debugging-Main 选项卡



在Main选项卡中，选择当前工程，一般会自动添加当前工程下的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。

2.6.3. Debugger 选项卡

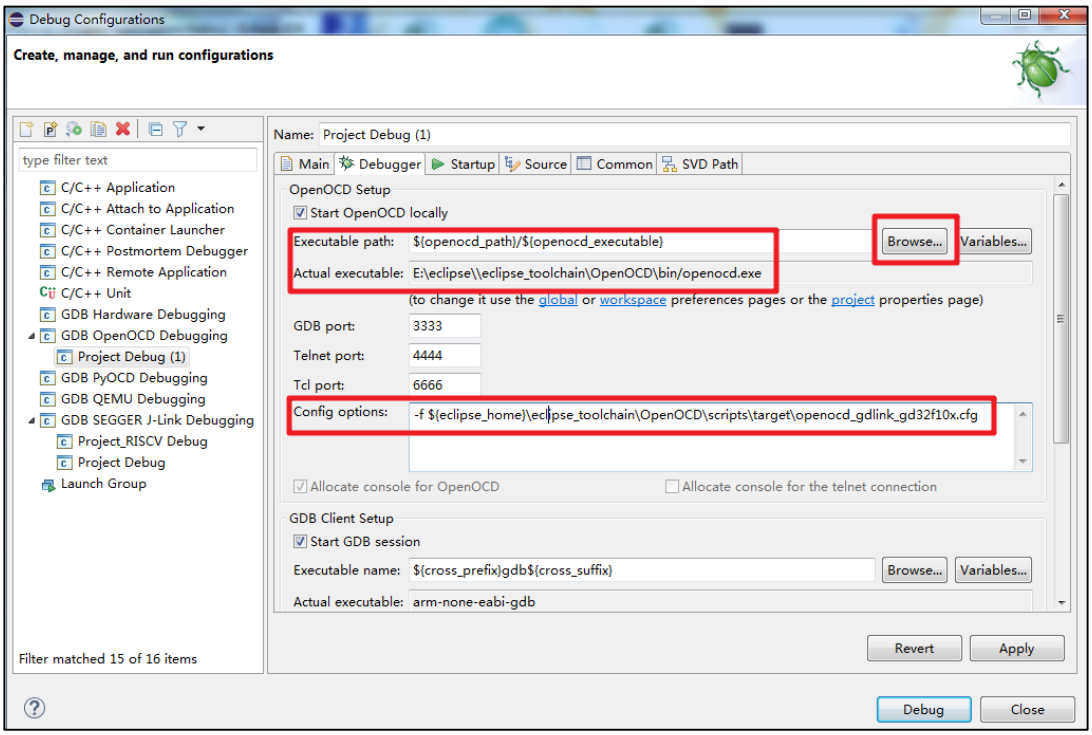
若在搭建Eclipse环境时已正确配置OpenOCD路径，这里将自动识别。如果之前没有正确配置，也可在Executable path栏，选中OpenOCD的绝对路径。

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_gd32f10x.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。

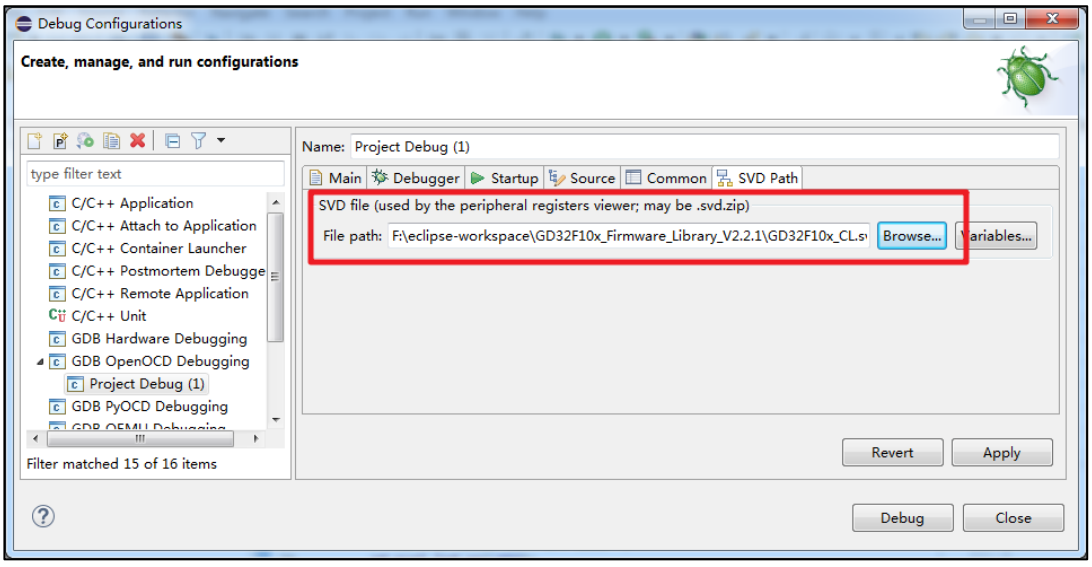
图 2-38. GDB OpenOCD Debugging-Debugger 选项卡



2.6.4. SVD Path 选项卡

在SVD Path选项卡，选择目标芯片所需的SVD文件。

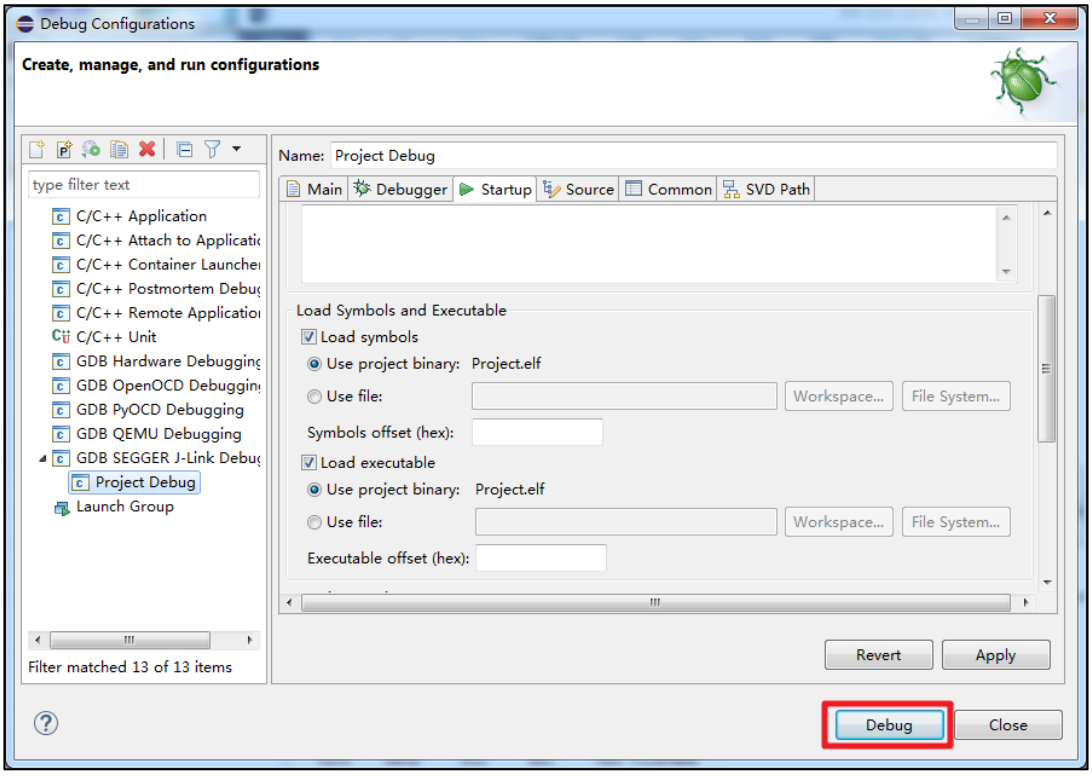
图 2-39. GDB OpenOCD Debugging-SVD Path 选项卡



2.7. Debug 界面

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

图 2-40. 进入 Debug 视图-1



切换至Debug视图。

图 2-41. 进入 Debug 视图-2

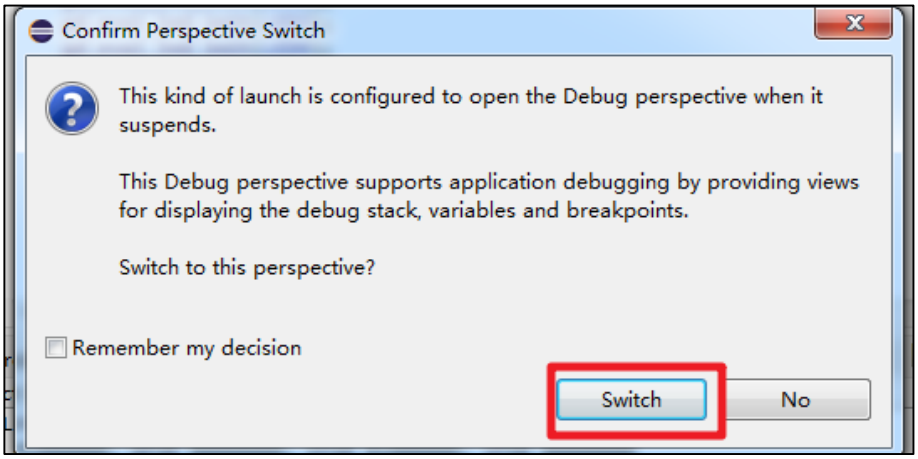
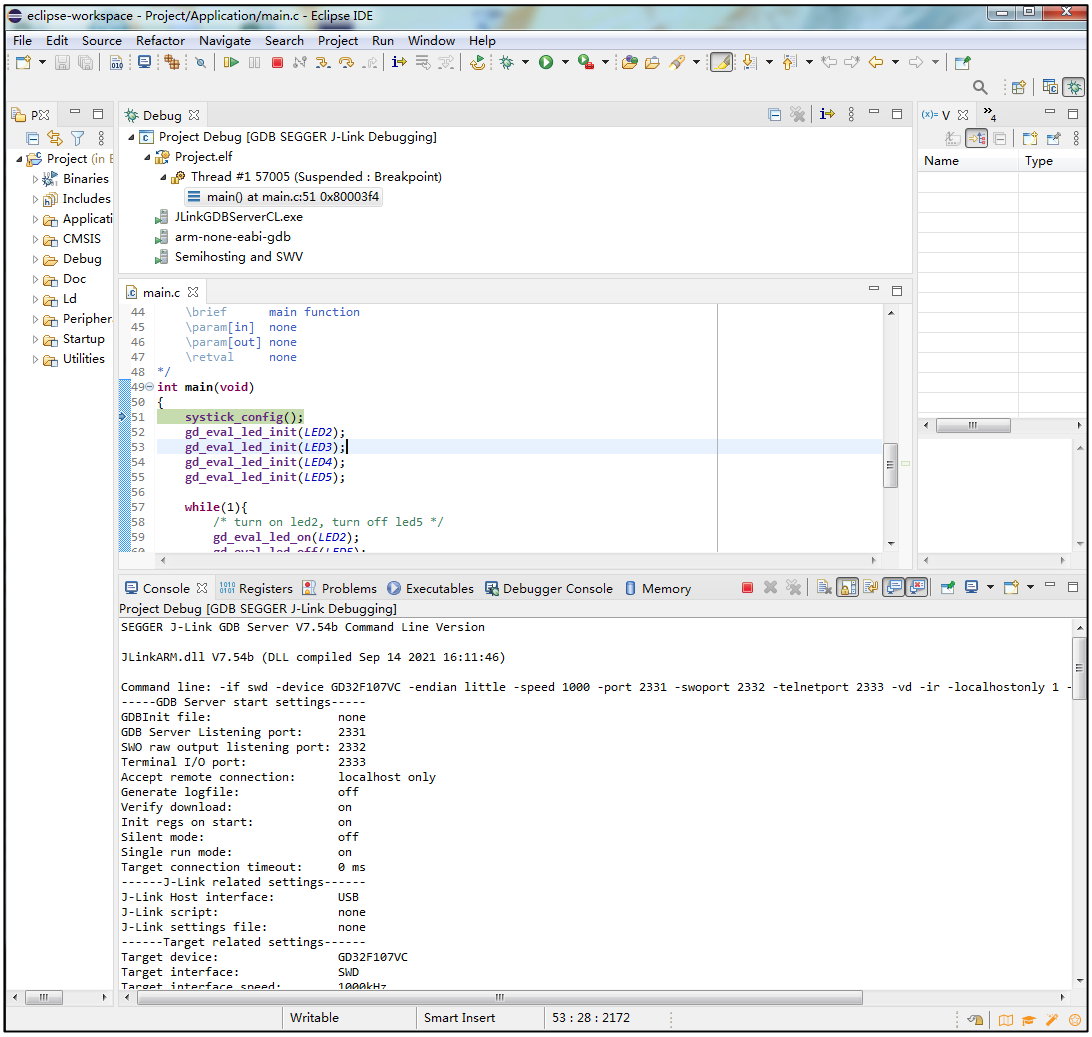


图 2-42. Debug 视图



2.7.1. 工具栏介绍

 ：resume全速运行

 ：suspend暂停

 ：terminate退出

 ：step into

 ：step over

：step out

 ：reset

2.7.2. 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 2-43. 打开 Registers 窗口

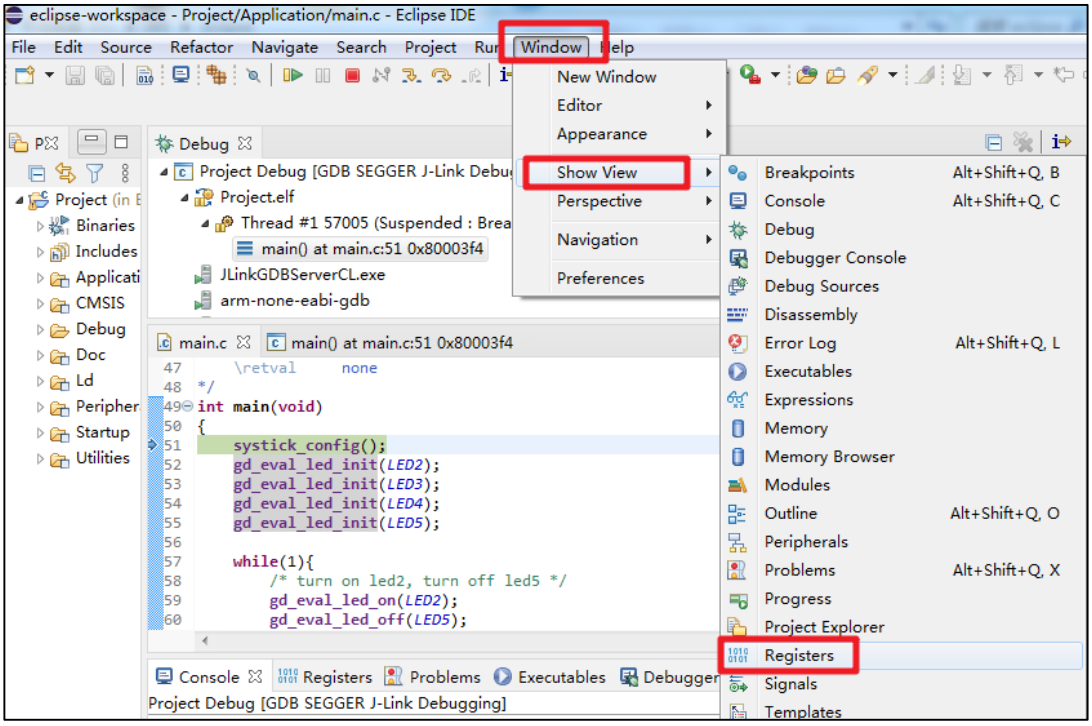
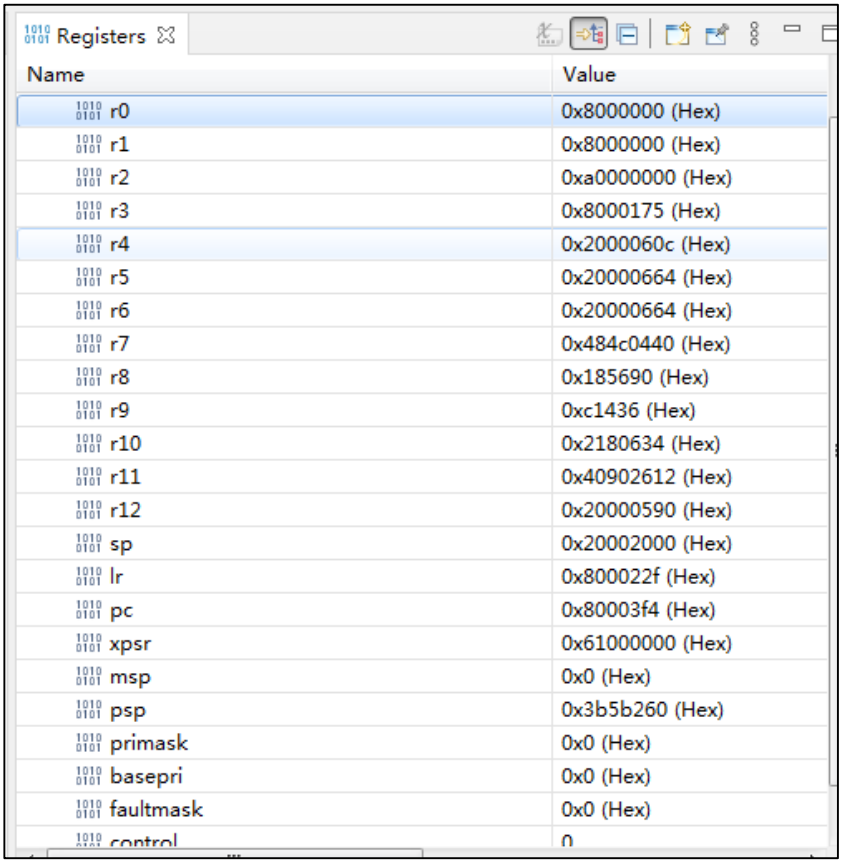


图 2-44. Registers 窗口



2.7.3. 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 2-45. 打开 Peripherals 窗口

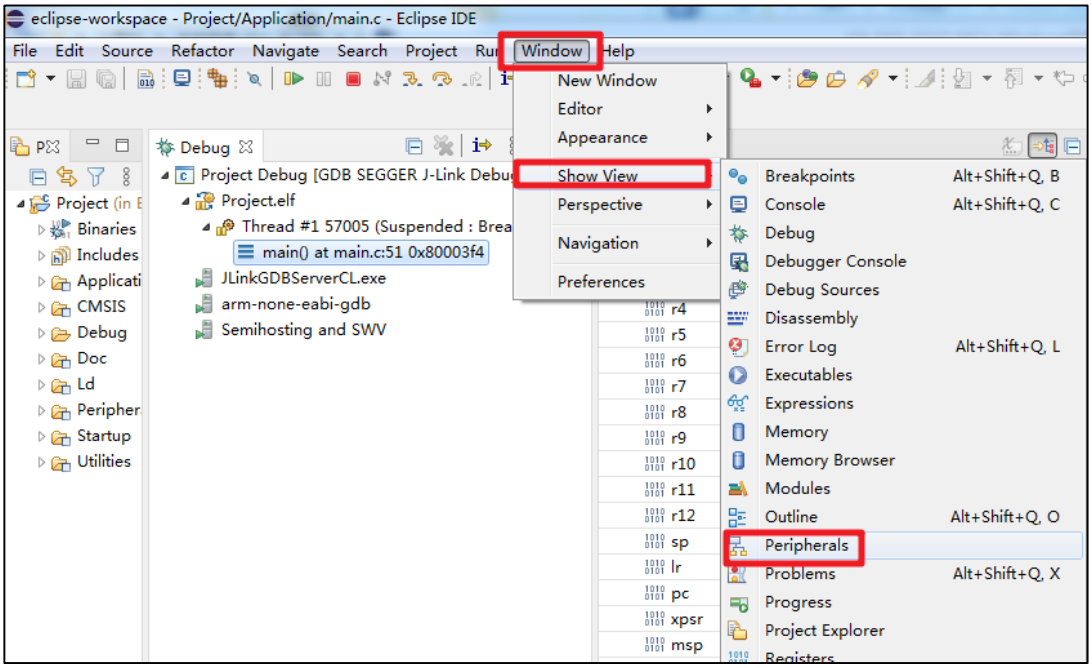
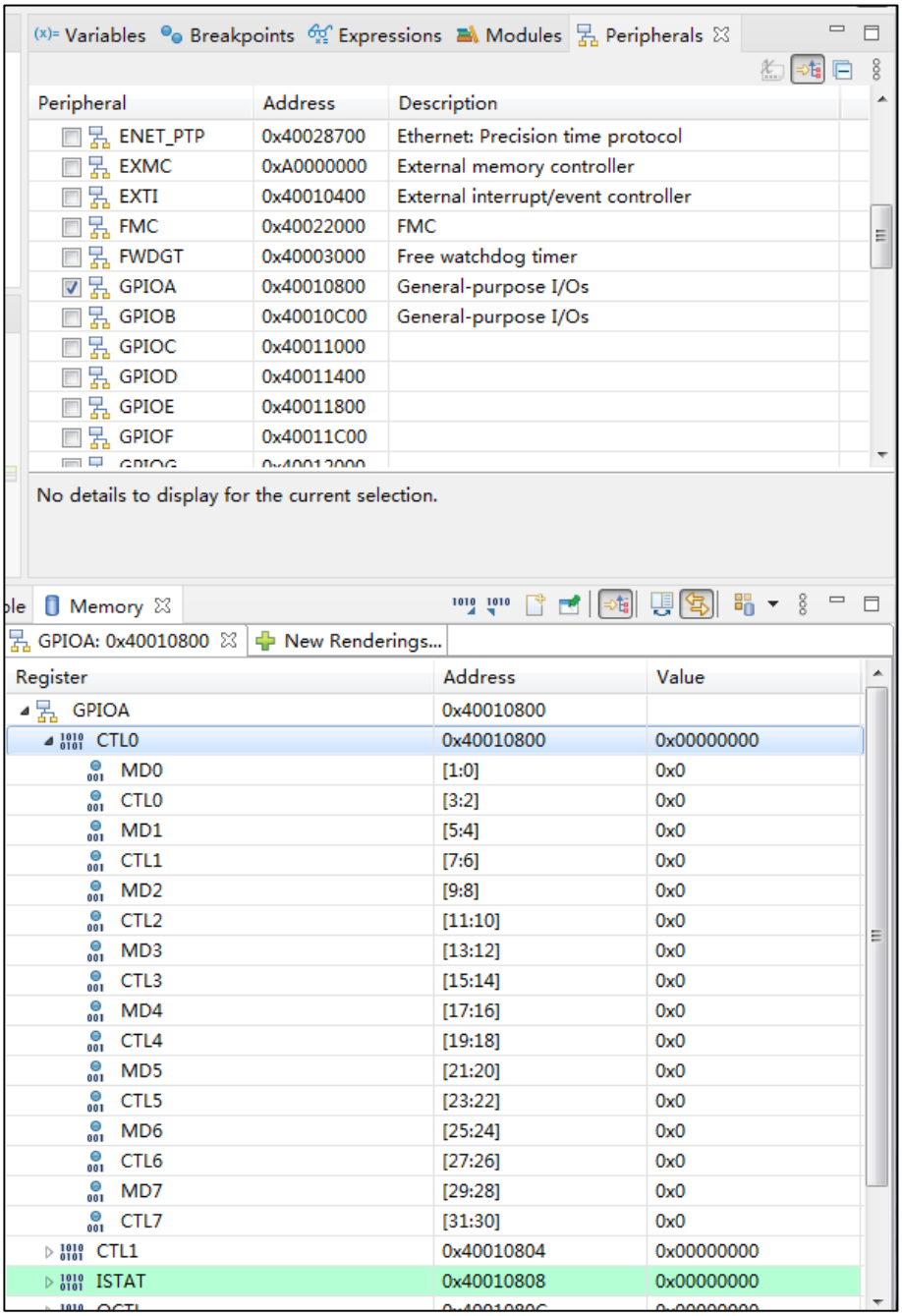


图 2-46. Peripherals 窗口



2.7.4. 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 2-47. 打开 Memory 窗口

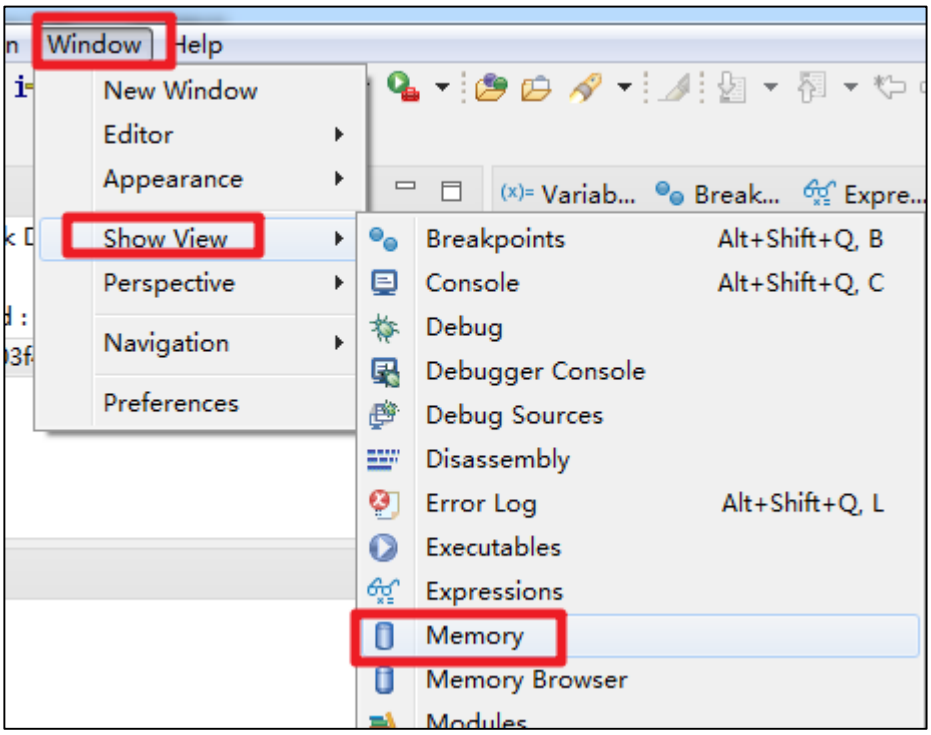
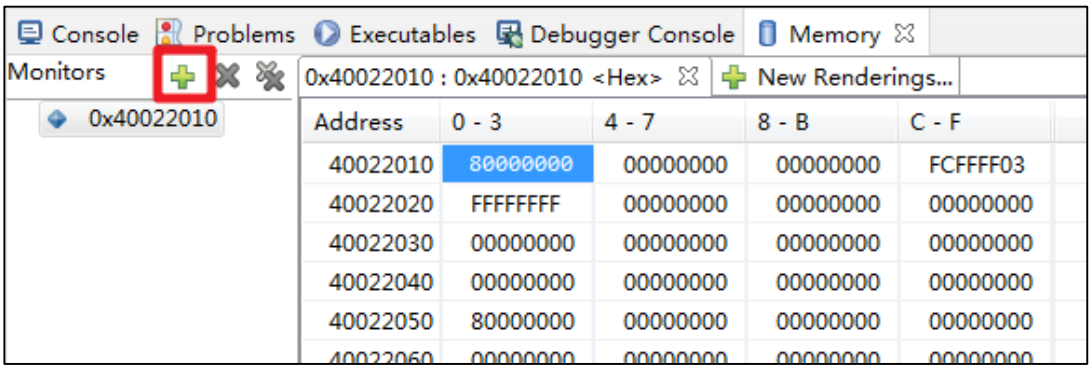


图 2-48. Memory 窗口



2.7.5. 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 2-49. 打开 Expressions 窗口

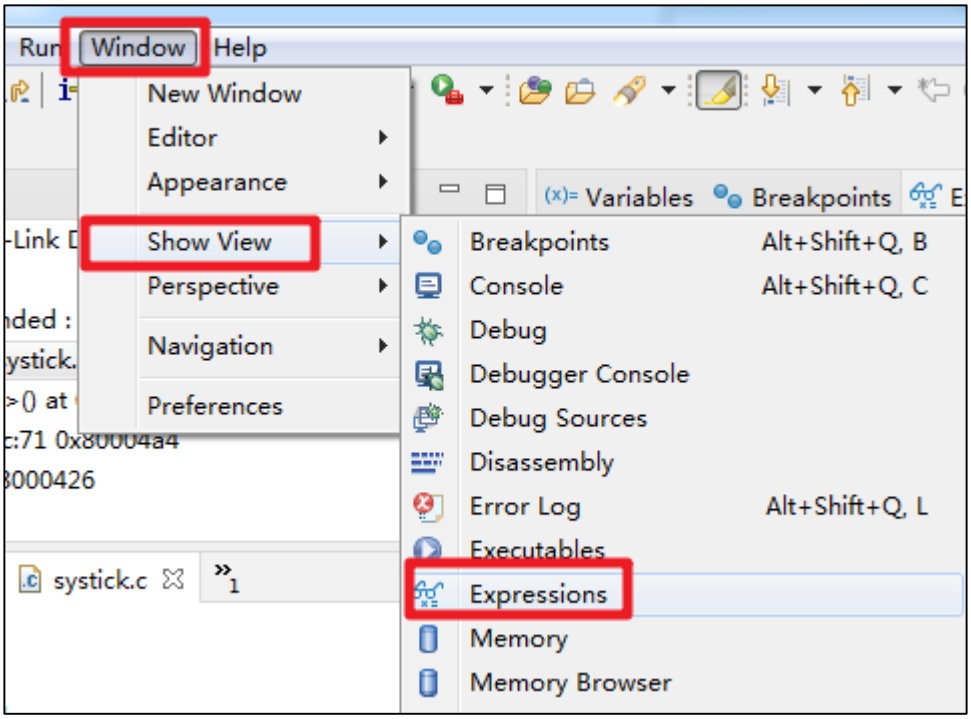
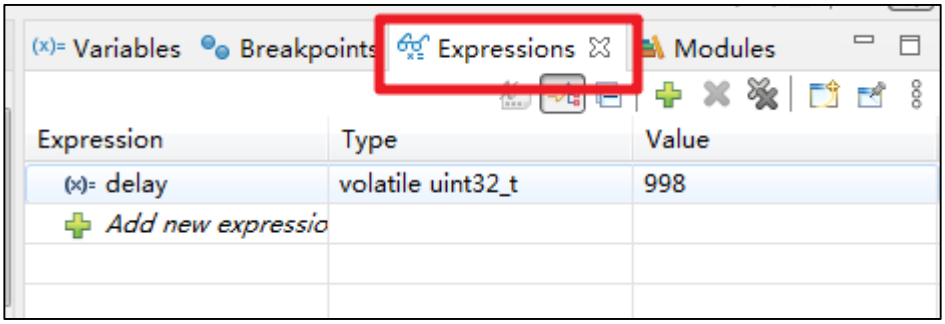


图 2-50. Expressions 窗口

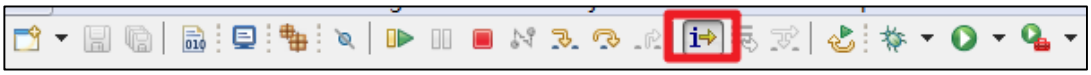


注意：Ecplise只有在代码停止运行的状态下才可以查看变量的值，暂时无法实时更新变量的值。

2.7.6. 反汇编窗口

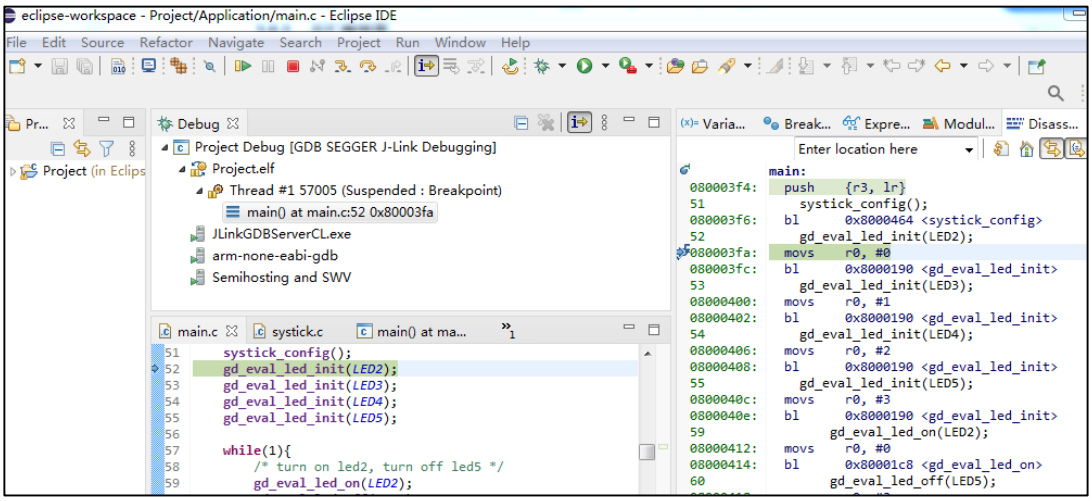
在调试工具栏选择Instruction Stepping Mode按键，打开反汇编窗口。

图 2-51. 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。

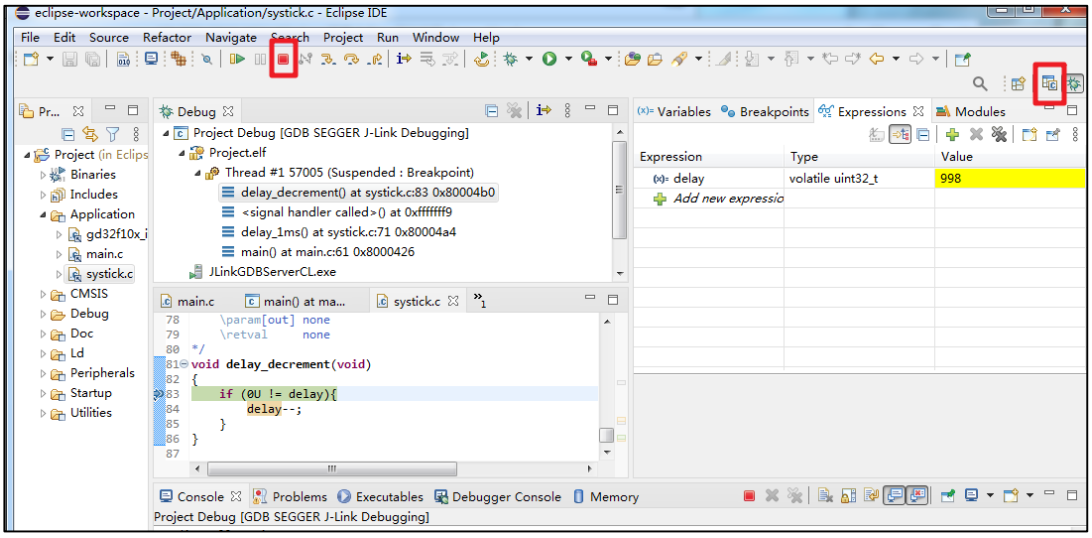
图 2-52. 反汇编窗口



2.7.7. 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。

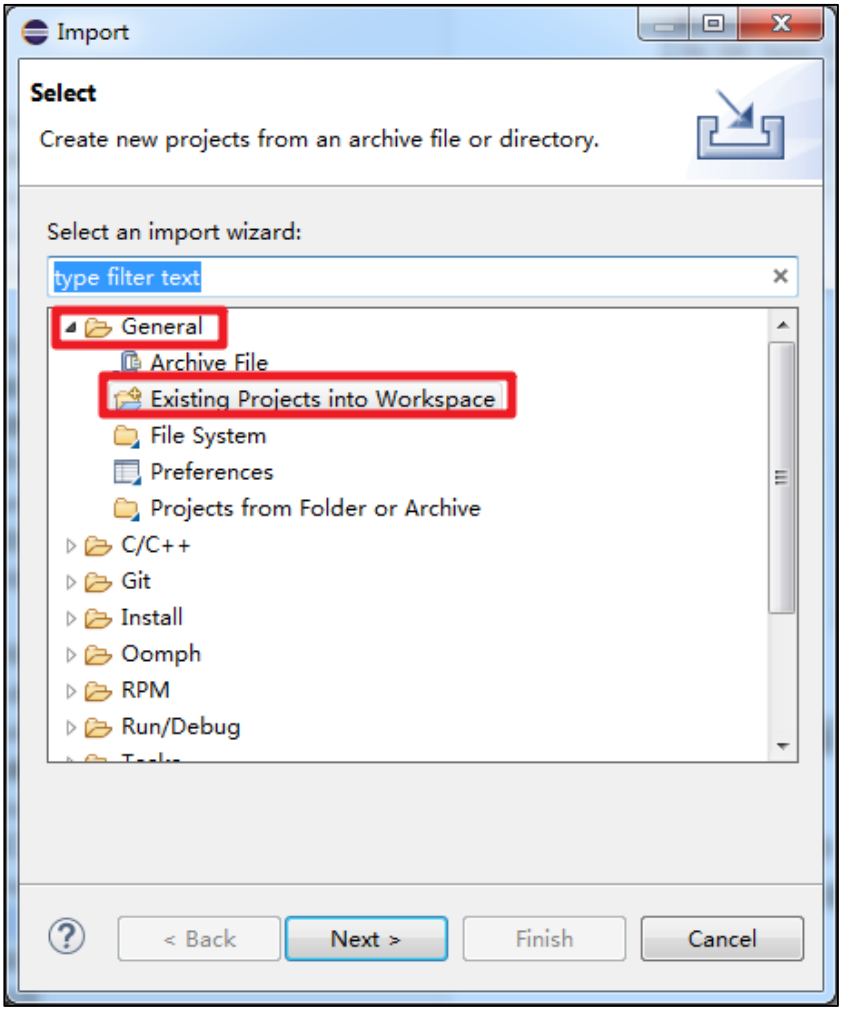
图 2-53. 退出 Debug 视图



3. 导入现有工程

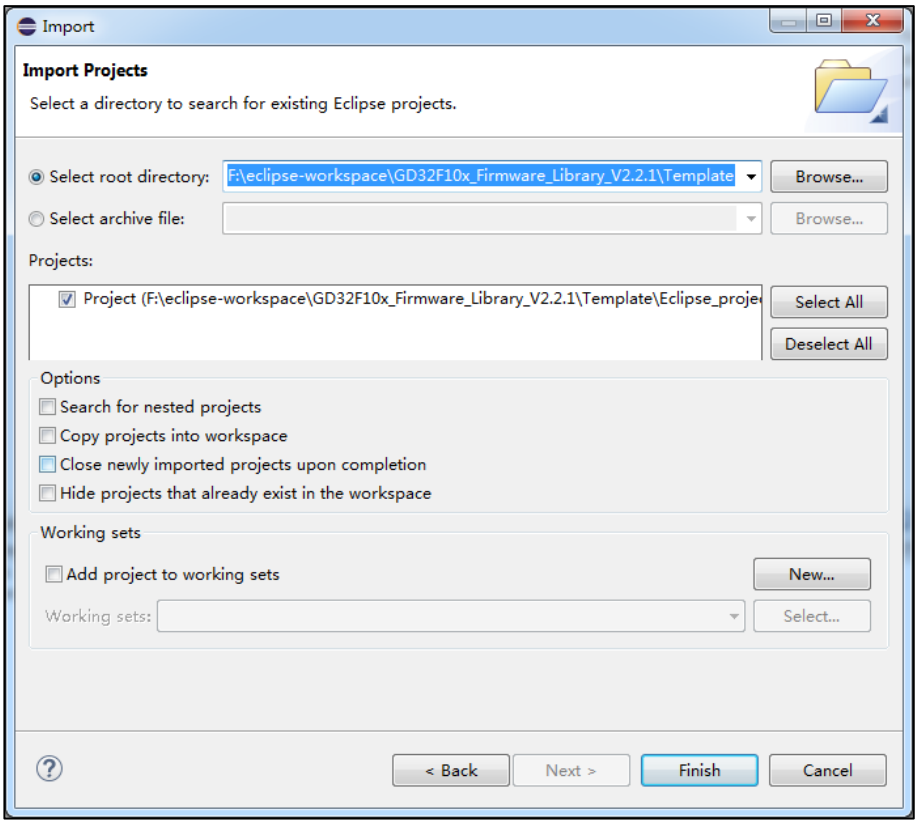
除新建工程外，也可直接导入已有Eclipse工程。在菜单栏中，点击File->Import，选择General->Exisiting Projects into Workspace导入已有工程，点击“Next”。

图 3-1. 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 3-2. 导入现有工程-2


Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory:
F:\eclipse-workspace\GD32F10x_Firmware_Library_V2.2.1\Template
Browse...
Select archive file:
Browse...
Projects:
Project (F:\eclipse-workspace\GD32F10x_Firmware_Library_V2.2.1\Template\Eclipse_proje
Select All
Deselect All
Options
Search for nested projects
Copy projects into workspace
Close newly imported projects upon completion
Hide projects that already exist in the workspace
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Finish
Cancel

4. 在 RAM 中调试

步骤1：修改链接脚本，例如修改如下图。

图 4-1. 在 RAM 中调试时 ld 文件 memory map

```
/* memory map */
MEMORY
{
    FLASH (rx)      : ORIGIN = 0x20000000, LENGTH = 8K
    RAM (xrw)       : ORIGIN = 0x20002000, LENGTH = 8K
}
```

步骤2：将中断向量表重定位到SRAM。完成步骤1和步骤2后重新编译工程。

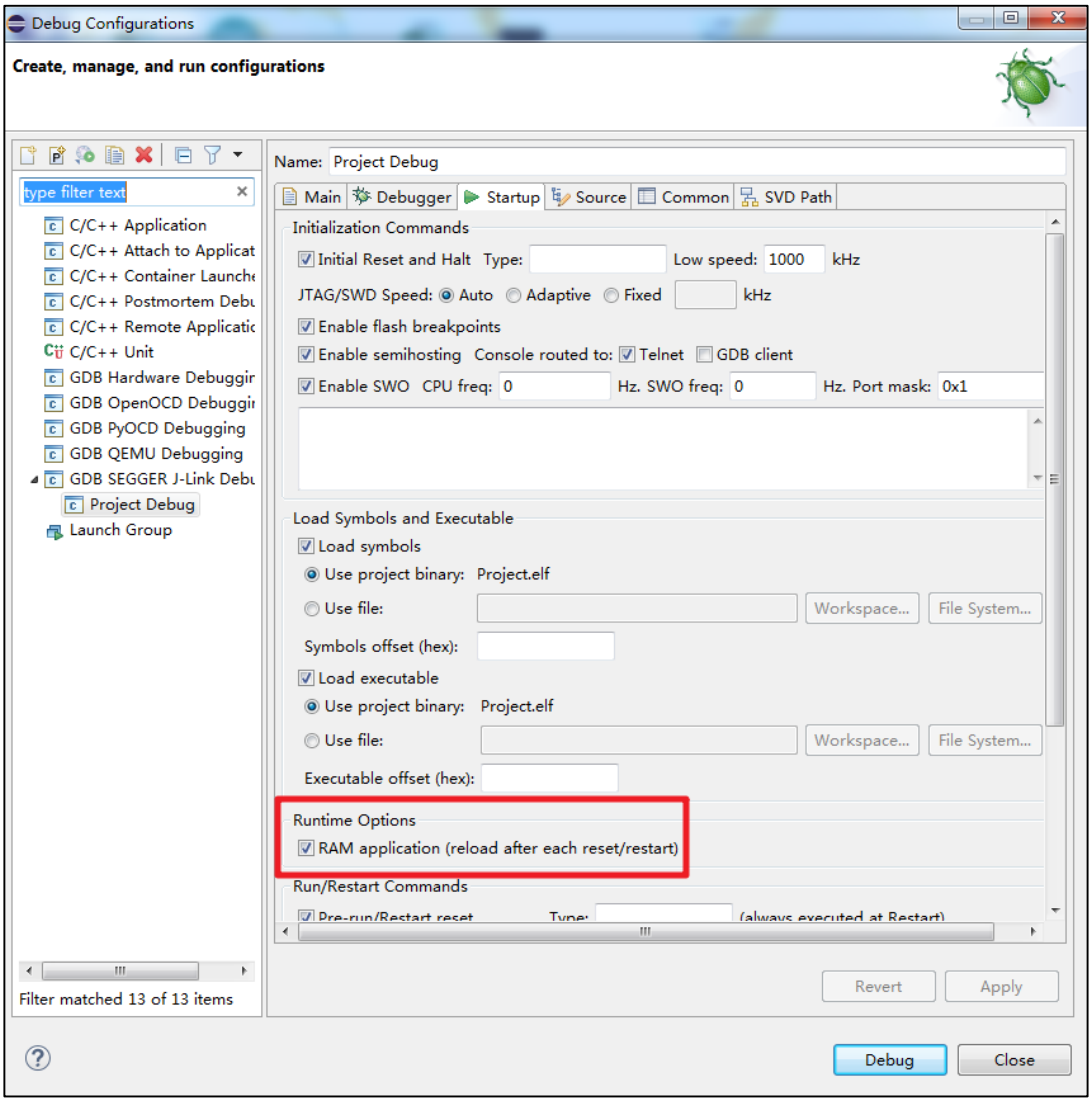
```
    nvic_vector_table_set(NVIC_VECTTAB_RAM, 0);
```

图 4-2. 在 RAM 中调试时重定位中断向量表

```
45      \param[in]  none
46      \param[out] none
47      \retval     none
48  */
49  int main(void)
50  {
51      nvic_vector_table_set(NVIC_VECTTAB_RAM, 0);
52      systick_config();
53      gd_eval_led_init(LED2);
54      gd_eval_led_init(LED3);
55      gd_eval_led_init(LED4);
56      gd_eval_led_init(LED5);
57
58      while(1){
```

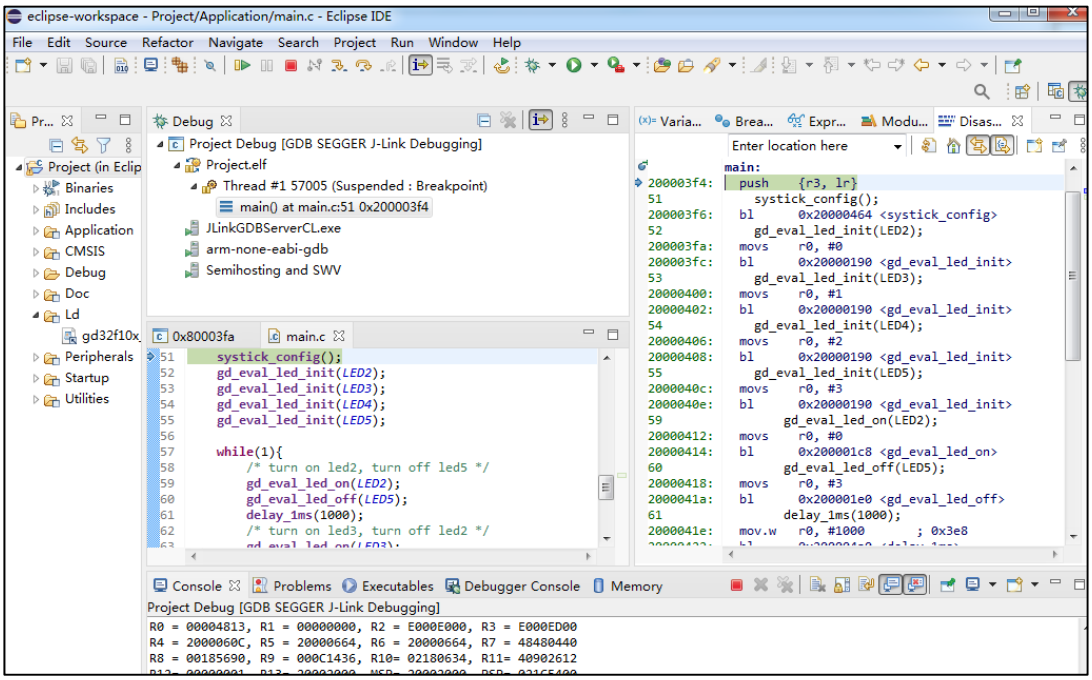
步骤3：在Debug Configurations->Startup选项中，勾选RAM application。

图 4-3. 在 RAM 中调试时 Debug Configurations



步骤4：进入RAM中调试时的Debug视图，如下图。

图 4-4. 在 RAM 中调试时 Debug 视图



5. 使用 printf 打印

5.1. 使用步骤

步骤1：添加syscall.c文件，文件中添加如下_write函数定义。

```
int _write(int file, char *ptr, int len)
{
      int DataIdx;

           for (DataIdx = 0; DataIdx < len; DataIdx++)
           {
               __io_putchar( *ptr++ );
           }
      return len;

}
```

步骤2：将usart重定向到__io_putchar函数。

```
int __io_putchar(int ch)
{
       usart_data_transmit(EVAL_COM0, (uint8_t) ch );
      while(RESET == usart_flag_get(EVAL_COM0, USART_FLAG_TBE)){
      };

      return ch;

}
```

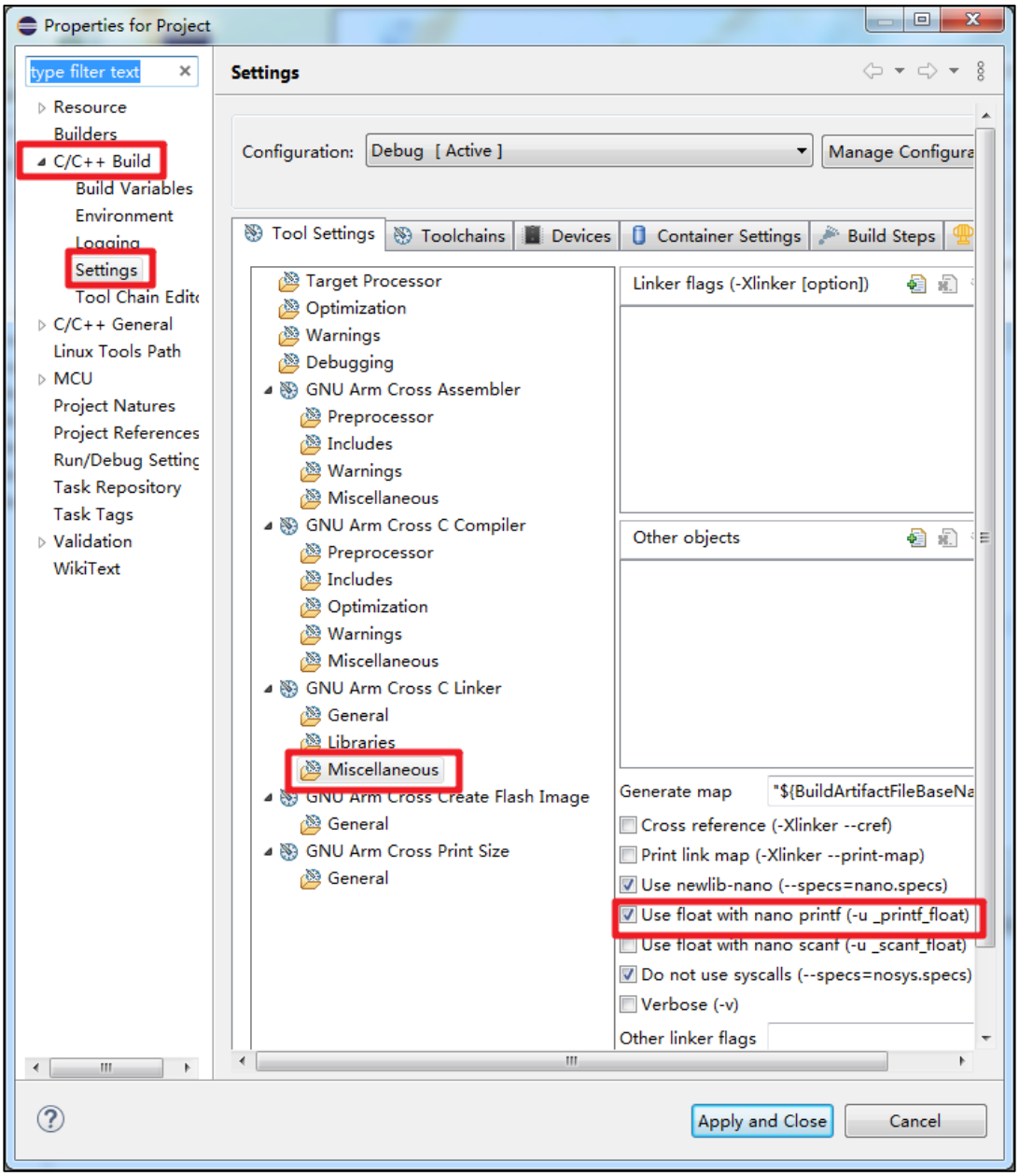
步骤3：使用printf函数，即可正常打印。

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

5.2. 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 Arm Cross C Linker->Miscellaneous中勾选-u _prinft_float选项。

图 5-1. 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。

6. 版本历史

表 6-1. 版本历史

版本号	描述	日期
1.0	初稿发布	2021 年 11 月 30 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 it’s 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 it’s 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.