应用笔记

Application Note

文档编号：**AN1093**

APM32 Arm MCU Windows 系统下 Eclipse 开发教程

版本：**V1.0**

1 引言

本应用笔记提供用户在 Windows 系统下使用 Eclipse 开发 APM32 系列 MCU 的应用指导。

目录

图索引

2 开发环境

- 开发板： APM32F103ZEMINI 开发板
- 调试器： J-Link V9/V10 或 Geehy-Link
- 操作系统： WIN10 64-bit OS
- IDE：Eclipse IDE for Embedded C/C++ Developers Version: 2022-03 (4.23.0)
- 交叉编译链： gcc-arm-none-eabi-10.3-2021.10-win32.exe
- 编译工具： xpack-windows-build-tools-4.3.0-1
- GDB 服务器： Pyocd CMSIS-DAP / J-Link GDB Server CL V7.62c

3 开发工程过程

3.1 新建工程过程

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

图 1 新建工程 C/C++ Project

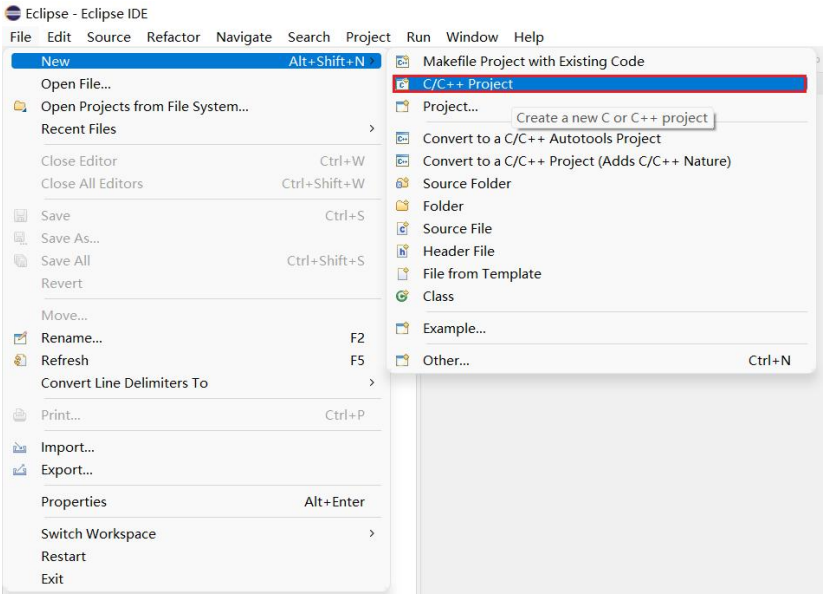
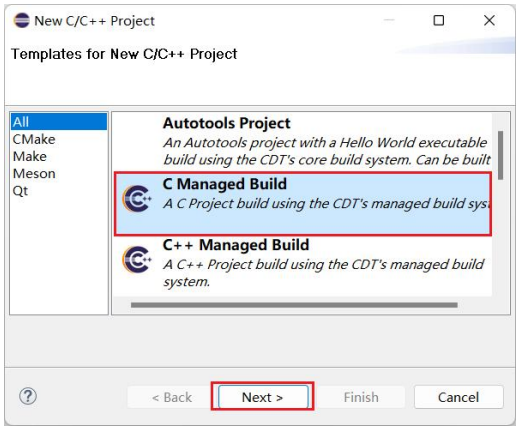
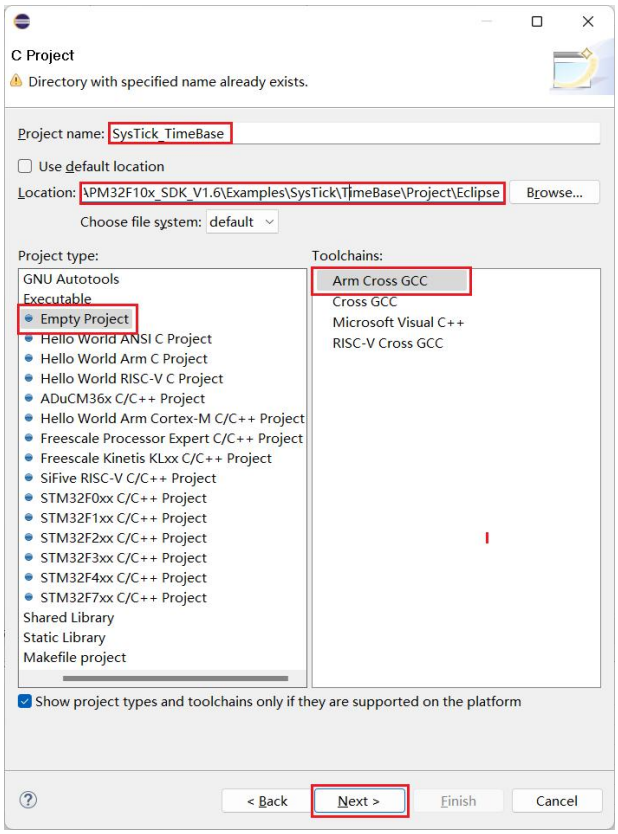


图 2 C Managed Build



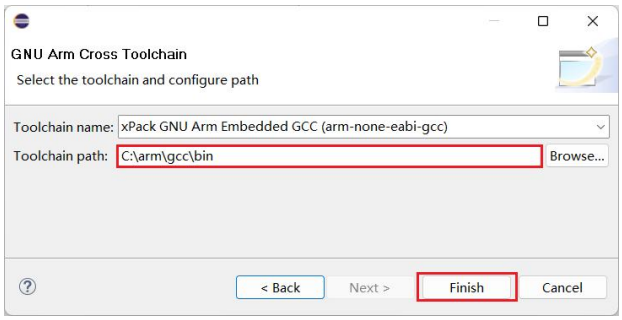
输入工程名，配置工程类型，建议将该工程放到 Project 目录下。配置编译链，编译链选择为 Arm Cross GCC。

图 3 配置工程



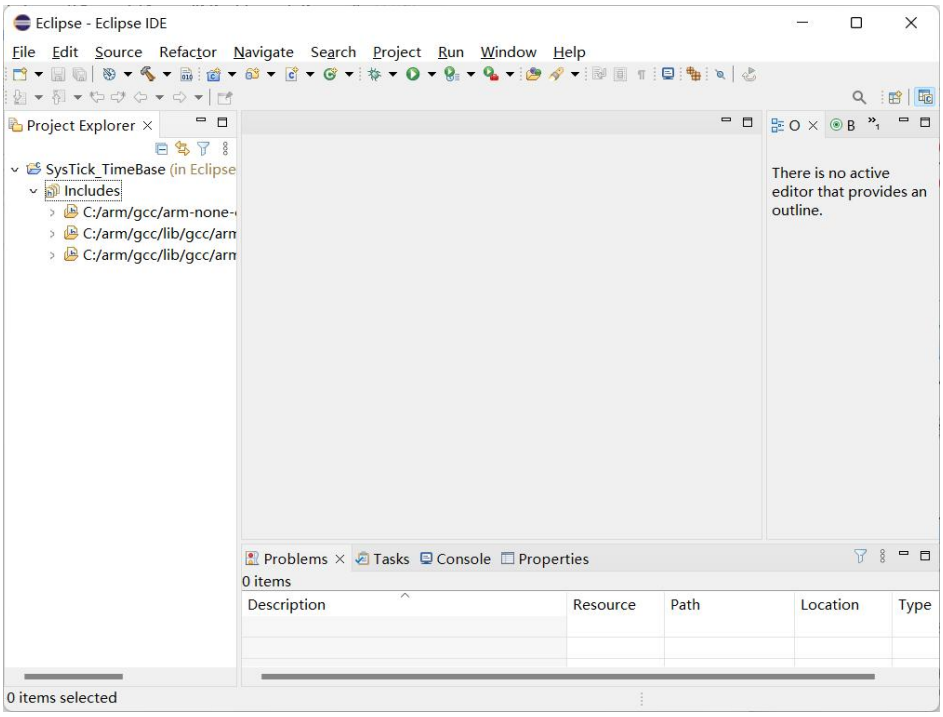
若 Eclipse IDE 的 Arm Toolchains 已正确配置，这里将会自动选择路径。若未正确配置，可点击 Browse 选择对应的绝对路径。

图 4 设置 Arm Toolchains Path



点击”Finish”，直至显示界面如图 5 视图所示。至此，完成 Project 的建立。

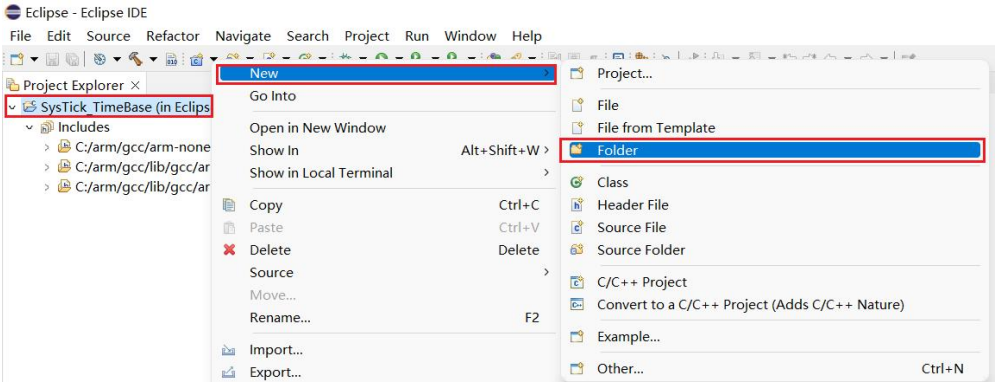
图 5 完成 Project 建立



3.2 工程文件夹及添加文件

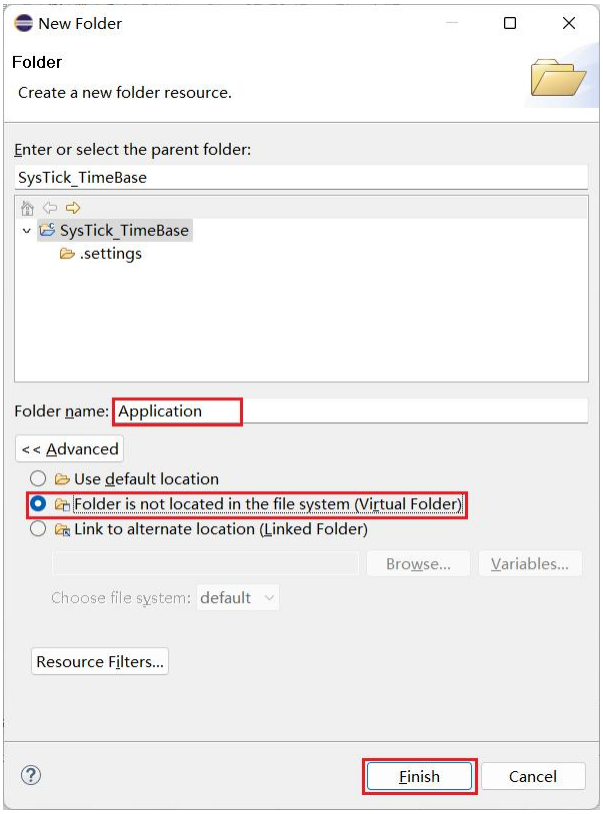
3.2.1 手动方式

图 6 添加文件夹



右键所在工程文件夹，建立虚拟文件夹 Application。

图 7 建立虚拟文件夹 Application

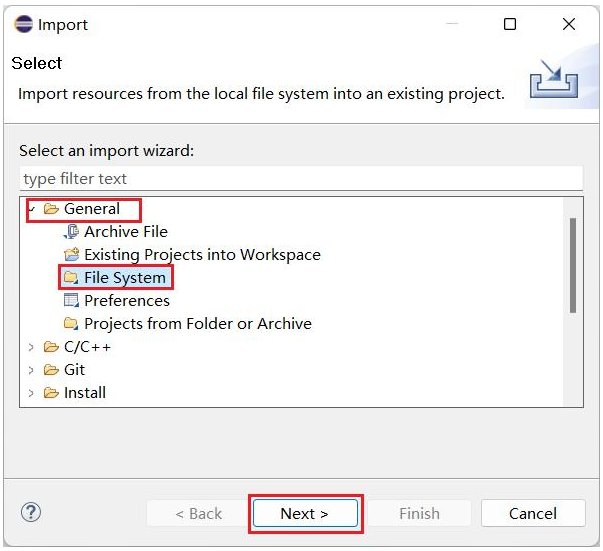


同样，在此工程目录下建立 CMSIS、Board、StdPeriphDriver 文件夹。

图 8 建立 CMSIS、Board、StdPeriphDriver 文件夹。

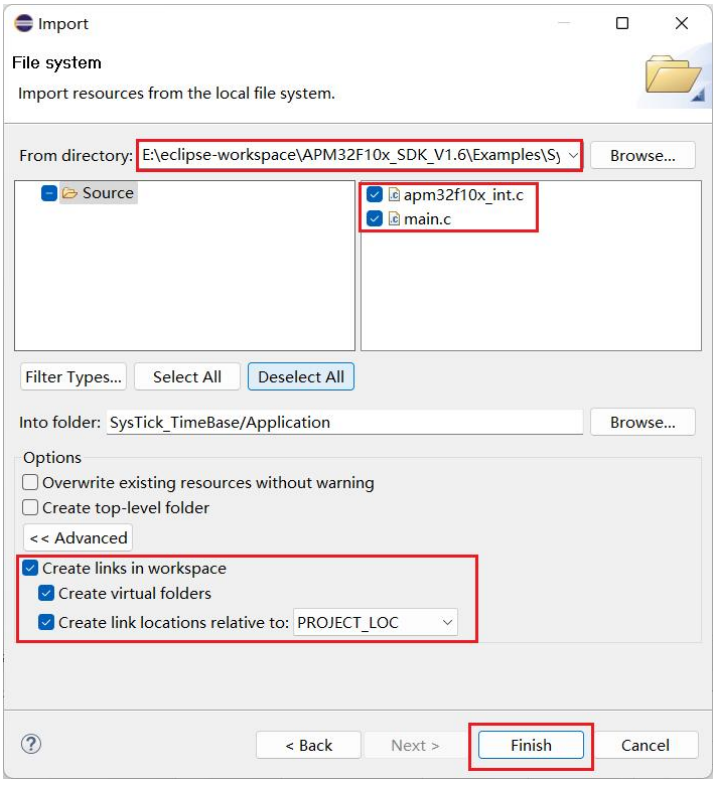
选中 Application 文件夹，右键选择 Import 选项，可直接导入文件。

图 9 导入文件



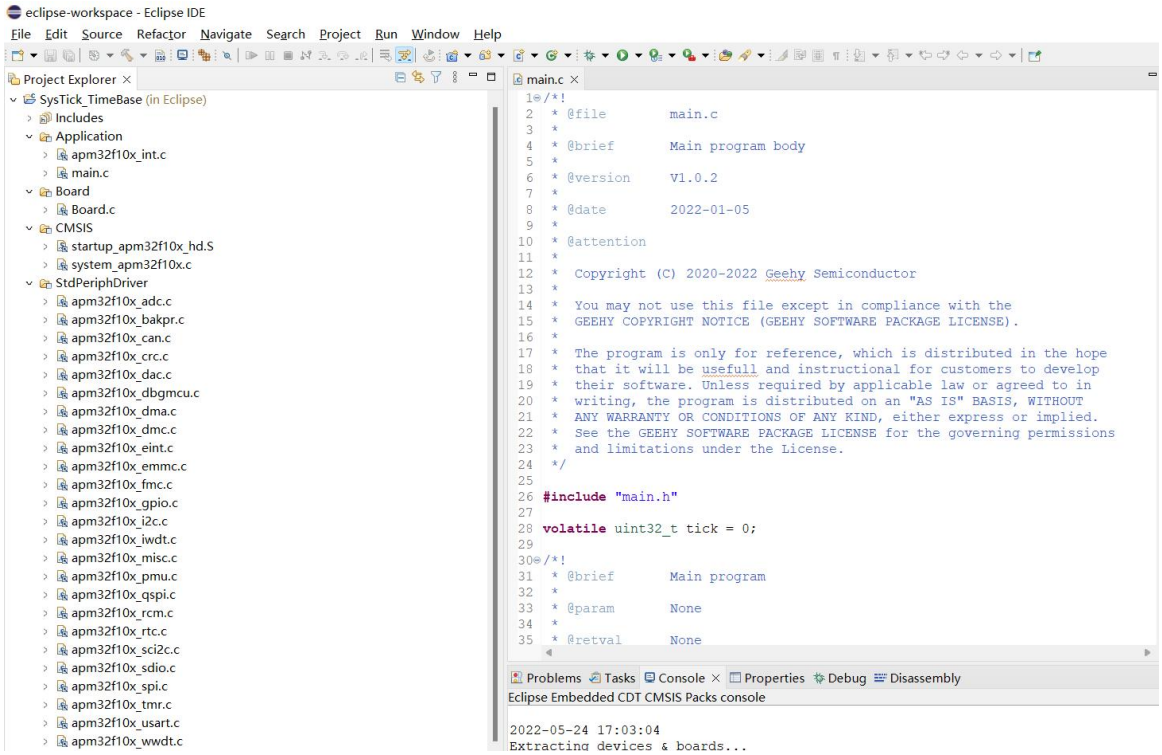
在导入文件时，选择“File System”作为导入方式，在弹出的文件路径选择对话框中，选择需要导入的文件路径，然后勾选需要导入的文件。

图 10 勾选需要导入的文件



同理，将 CMSIS、 Board、 StdPeriphDriver 文件夹的所需文件导入。

图 11 导入所有文件



3.2.2 “Refresh”方式

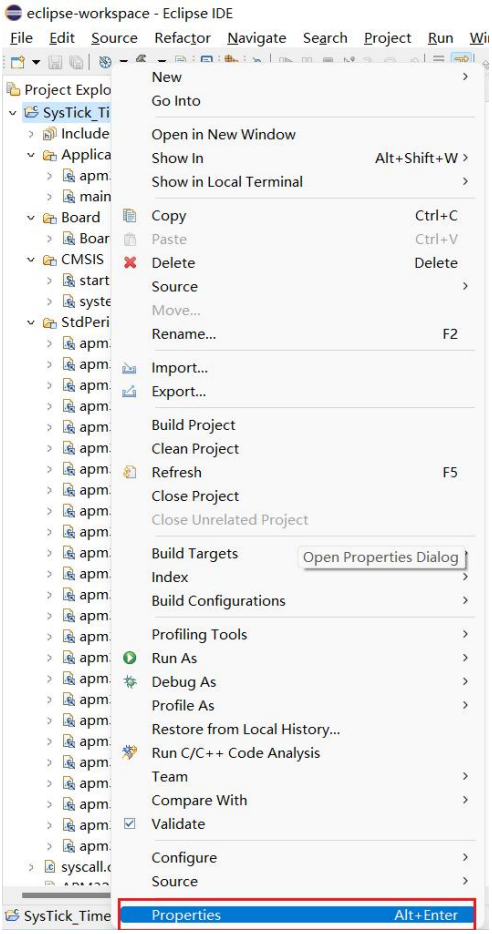
除了上述手动导入文件夹外，可直接把文件和文件夹放在.cproject 同级目录下，在 Eclipse IDE 中右键点击工程名，选择“refresh”即可将文件夹及文件导入工程。

注意：由于“refresh”建立的文件及文件夹是存储在磁盘中的，所以若在 Eclipse IDE 中删除某文件，其对应在磁盘中的文件也会删除。

3.3 工程配置过程

右键选择工程，在弹出的选项框选择 Properties。

图 12 打开 Properties 选项

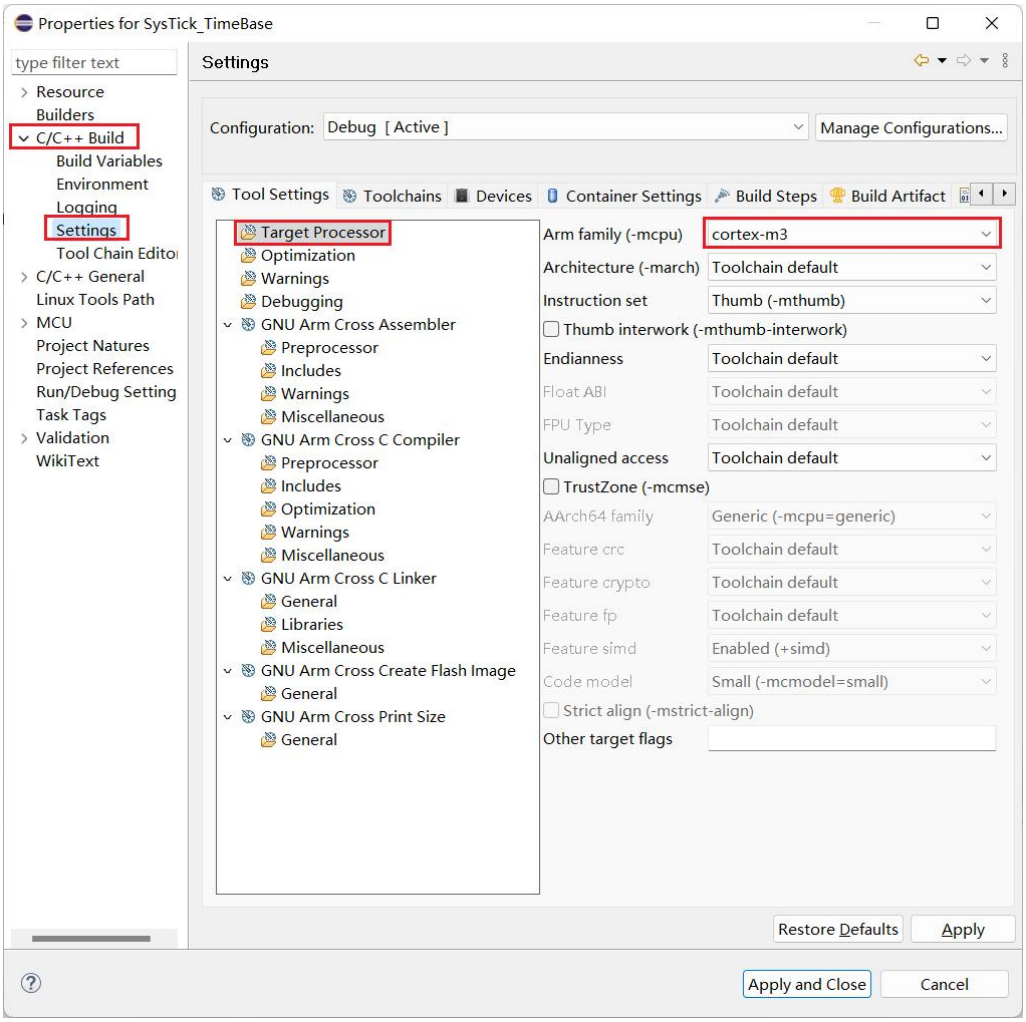


3.3.1 配置 Target Processor 选项卡

针对“C/C++ Build->Settings->Tool Settings->Target Processor”的配置如下：

根据目标芯片选择对应的内核，例如 cortex-m3、 cortex-m4、 cortex-m23 或 cortex-m33。 在本例中所使用的芯片内核为 cortex-m3。

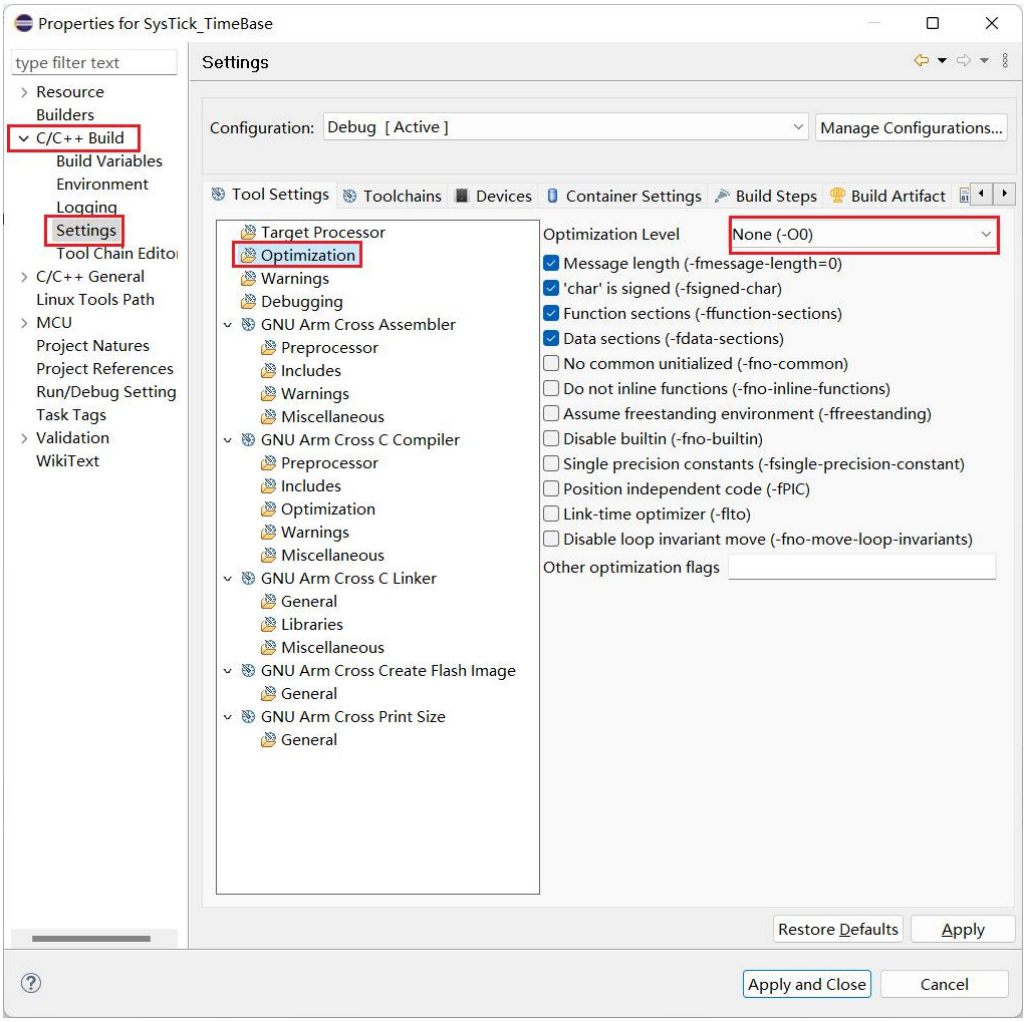
图 13 Target Processor 配置



3.3.2 配置 Optimization 选项卡

在“C/C++ Build->Settings->Tool Settings->Optimization”选项中， 可为芯片配置不同的优化等级，例如-O0、 -O1、-O2、 -O3、 -Os、 -Ofast、 -Og。

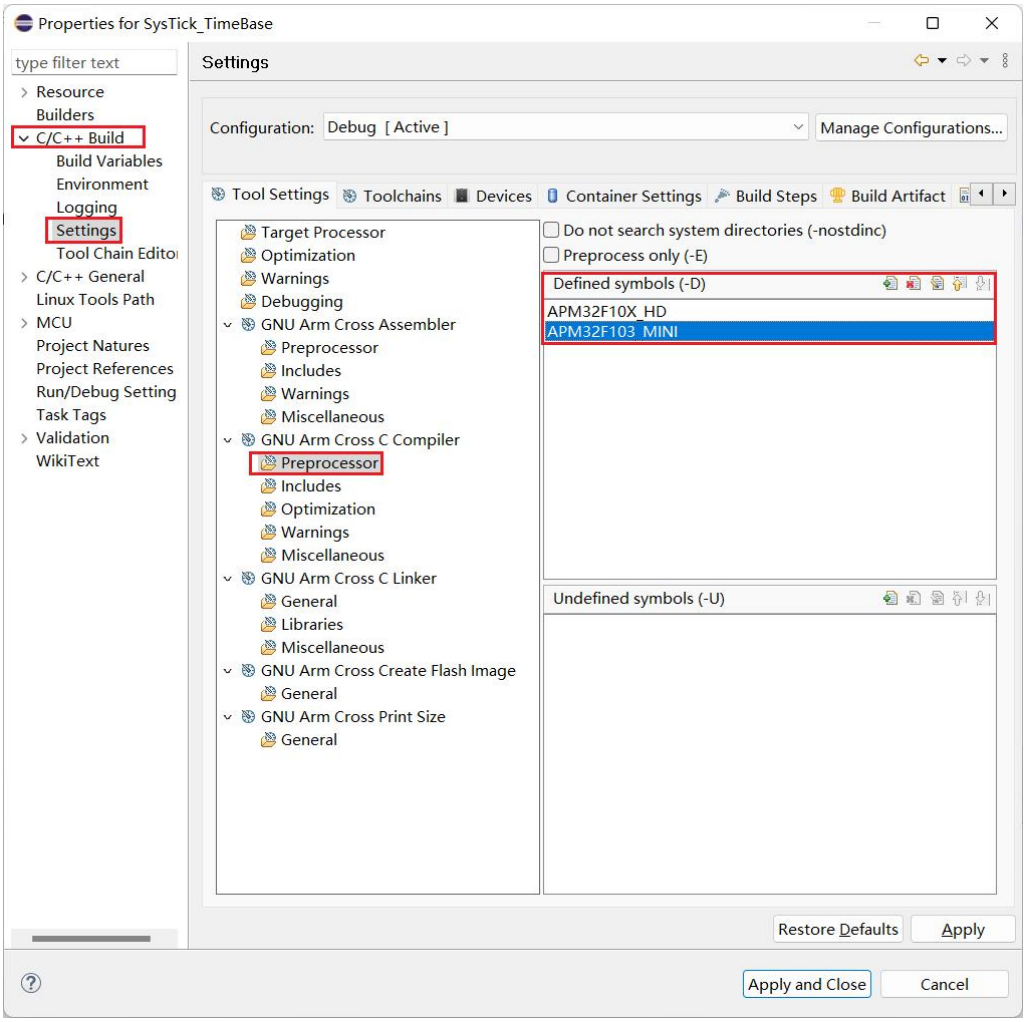
图 14 Optimization 配置



3.3.3 配置 GNU Arm Cross C Compiler 选项卡

配置“C/C++ Build->Settings->Tool Settings->GNU Arm Cross C Compiler”选项，本例中，需在‘’Preprocessor->Defined symbols”选项中添加预编译宏：APM32F10X_HD、APM32F103_MINI。

图 15 GNU Arm Cross C Compiler 配置



工程所需的头文件在”includes->Include paths”选项中添加。本例中，添加的头文件路径如下：

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

"${ProjDirPath}/../../../../../Board"

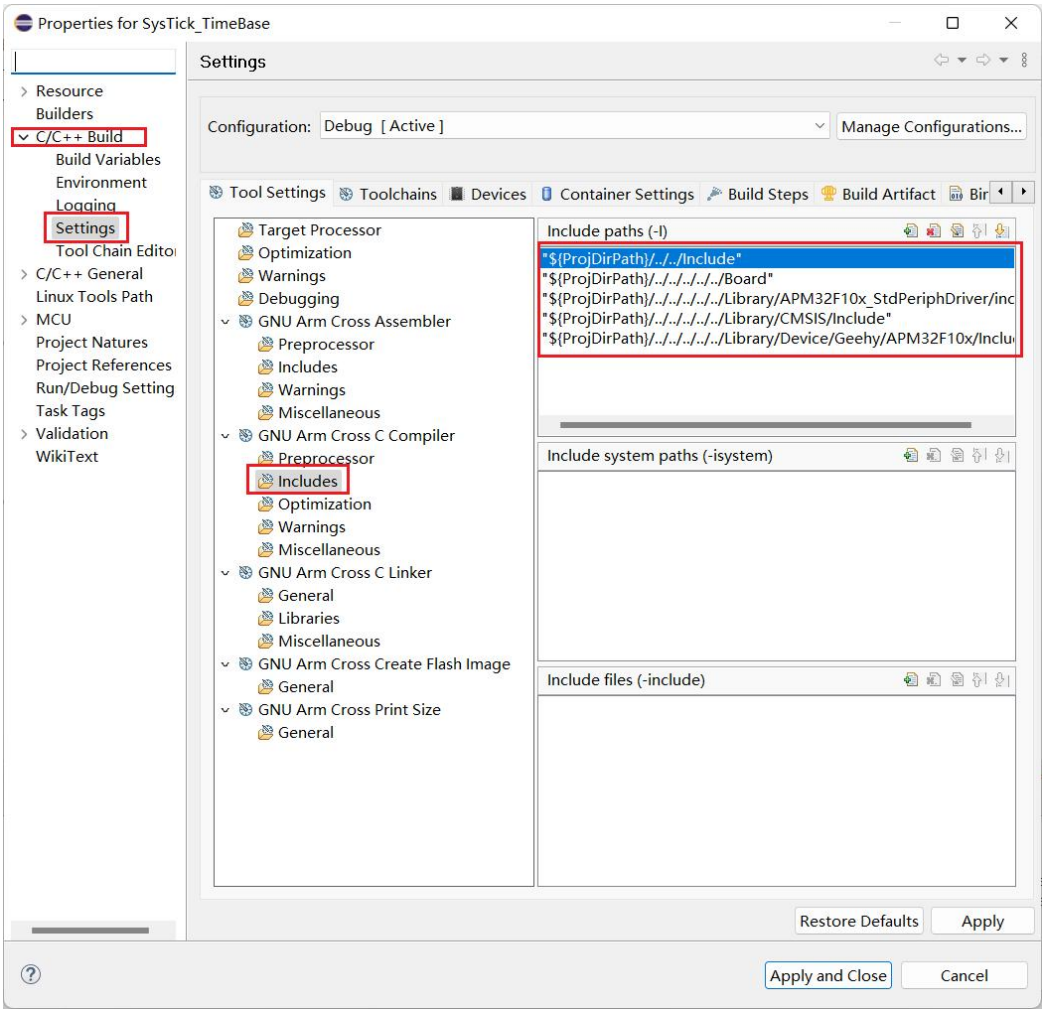
"${ProjDirPath}/../../../../../Library/APM32F10x_StdPeriphDriver/inc"

"${ProjDirPath}/../../../../../Library/CMSIS/Include"

"${ProjDirPath}/../../../../../Library/Device/Geehy/APM32F10x/Include"

注意： 路径添加有相对路径添加和绝对路径添加，本例中使用相对路径添加。

图 16 添加工程所需的头文件路径



3.3.4 配置 GNU Arm Cross C Linker 选项卡

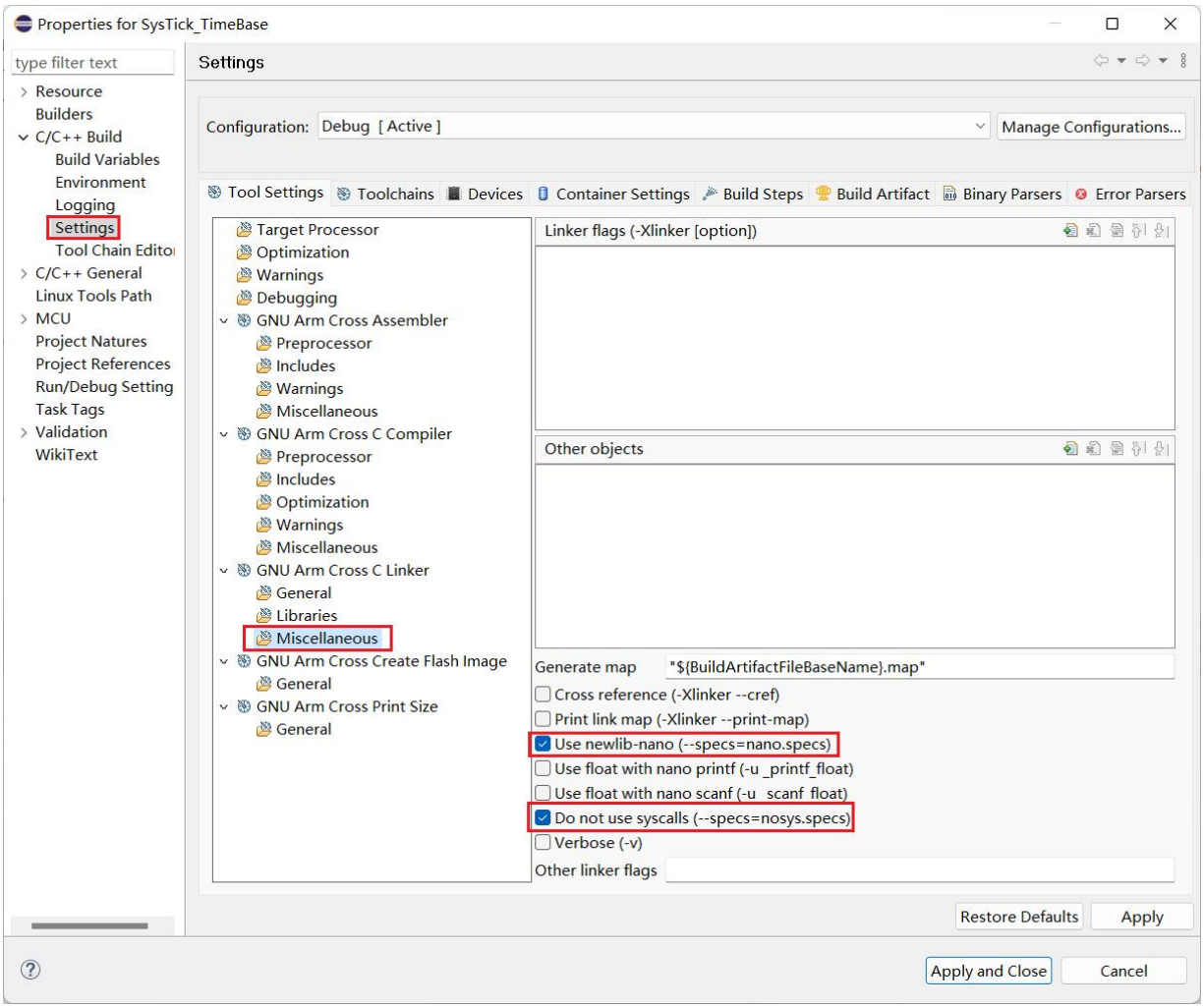
配置“C/C++ Build->Settings->Tool Settings->GNU Arm Cross C Linker”链接选项。本例中，需在“General ->Script files”选项中添加：
"${workspace_loc:/${ProjName}/gcc_APM32F10xxE.ld}"

ld 脚本是链接器 ld 使用的脚本文件，他描述了程序的内部布局，定义了链接时需要使用的符号，指定了代码段、数据段等存储区域的起始地址和大小等信息。使用的 ld 脚本应该与目标芯片的 FLASH、RAM 的大小及客户所需的内存配置一致。

注意： 本例中所添加的 ld 文件，除使用上述相对路径添加外，也可直接添加绝对路径。

勾选 Miscellaneous 选项的 Use newlib-nano 及 Do not use syscalls，可对代码大小进行优化。

图 17 GNU Arm Cross C Linker 配置

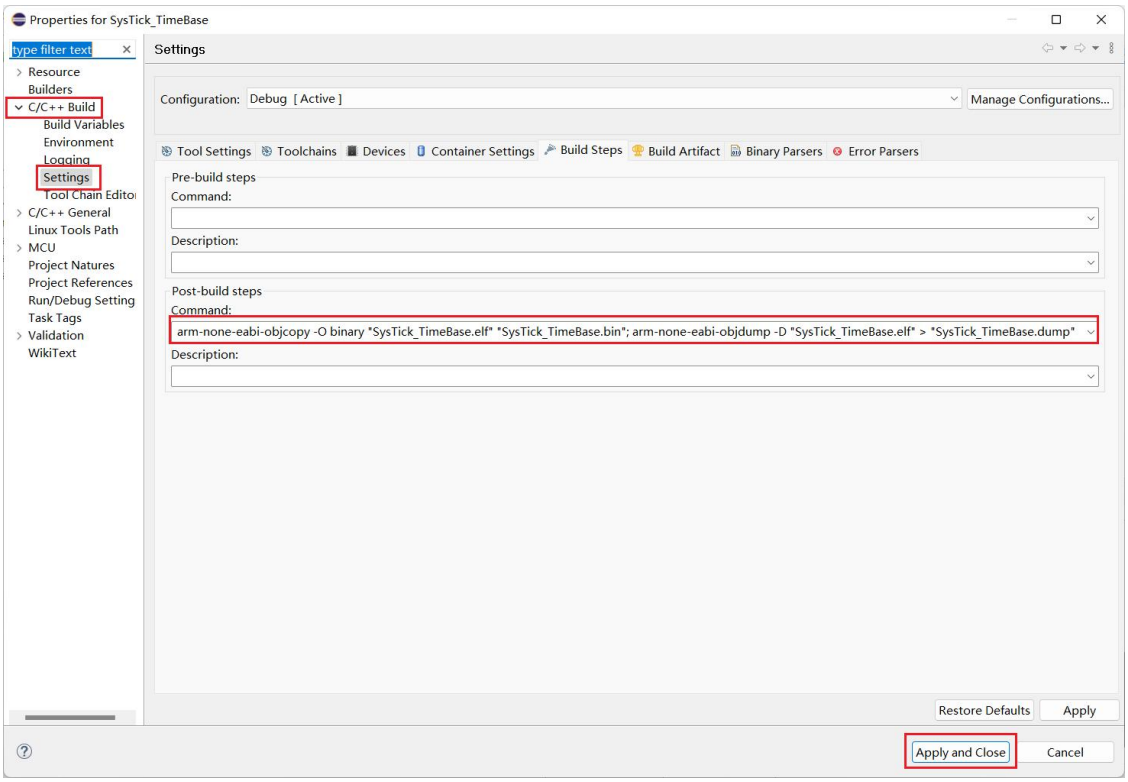


3.3.5 配置 Build Steps 选项卡-生成 bin 文件

若要生成 bin/hex 文件，需在“C/C++ Build->Settings-> Build Steps”添加相关命令。

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

图 18 配置-生成 bin 文件

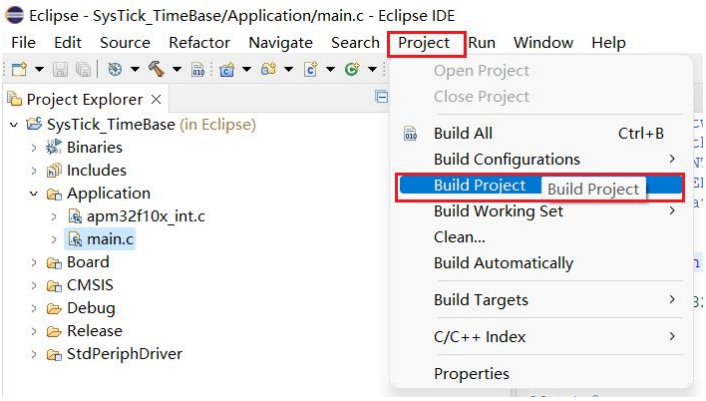


3.4 编译工程

点击 Project，选择 Build Project 可对当前工程编译。

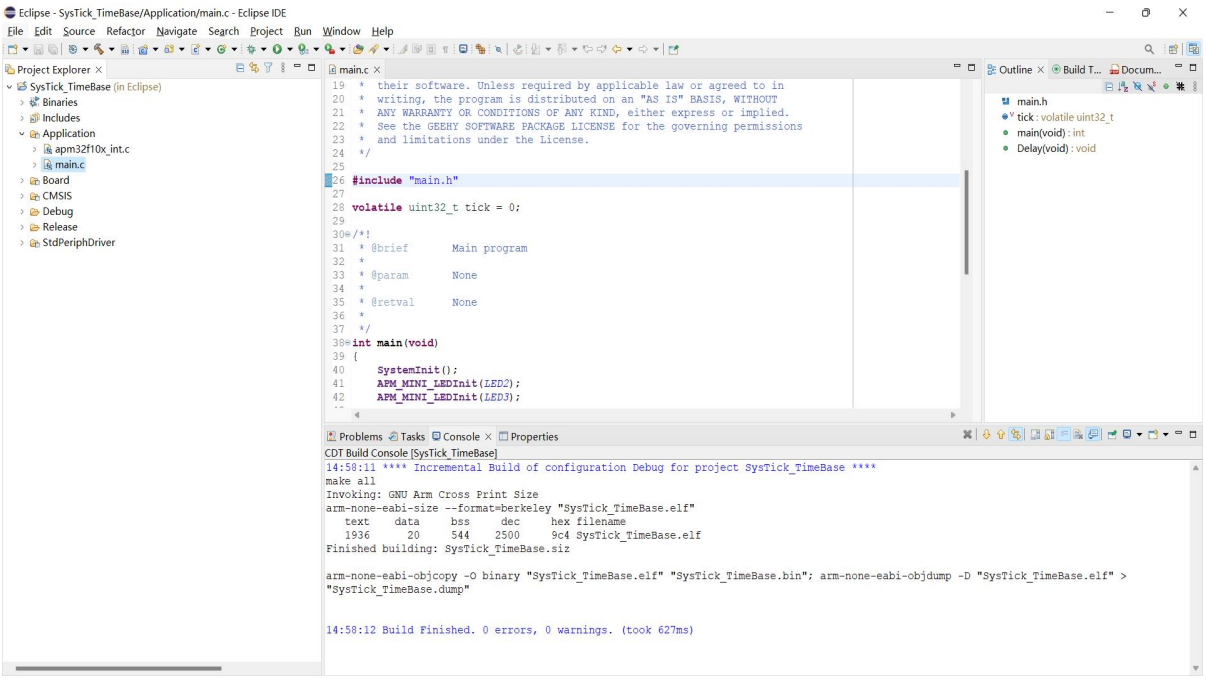
注意： 使用 Build Project 仅编译当前工程，使用 Build All 则会编译当前 workspace 所有工程。

图 19 编译工程



注意：在进行编译前，务必先保存当前工程。否则，编译的可能是上一次保存的工程，不是当前修改后的工程。为确保编译的正确性，需要在修改后进行工程 clean,然后再进行编译。编译完成后，会生成对应的 elf、hex 及 bin 文件。

图 20 编译结果

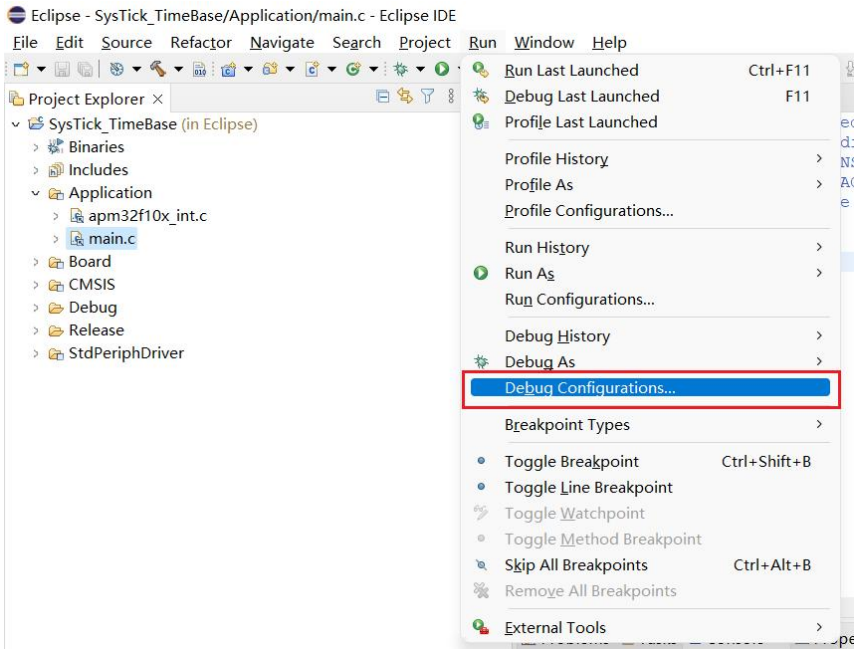


3.5 使用 J-Link 下载及调试工程

3.5.1 打开 Debug 配置界面

在工程中，点击菜单栏中的 Run，在弹出的选项框中点击 Debug Configurations，进入 Debug 配置界面。

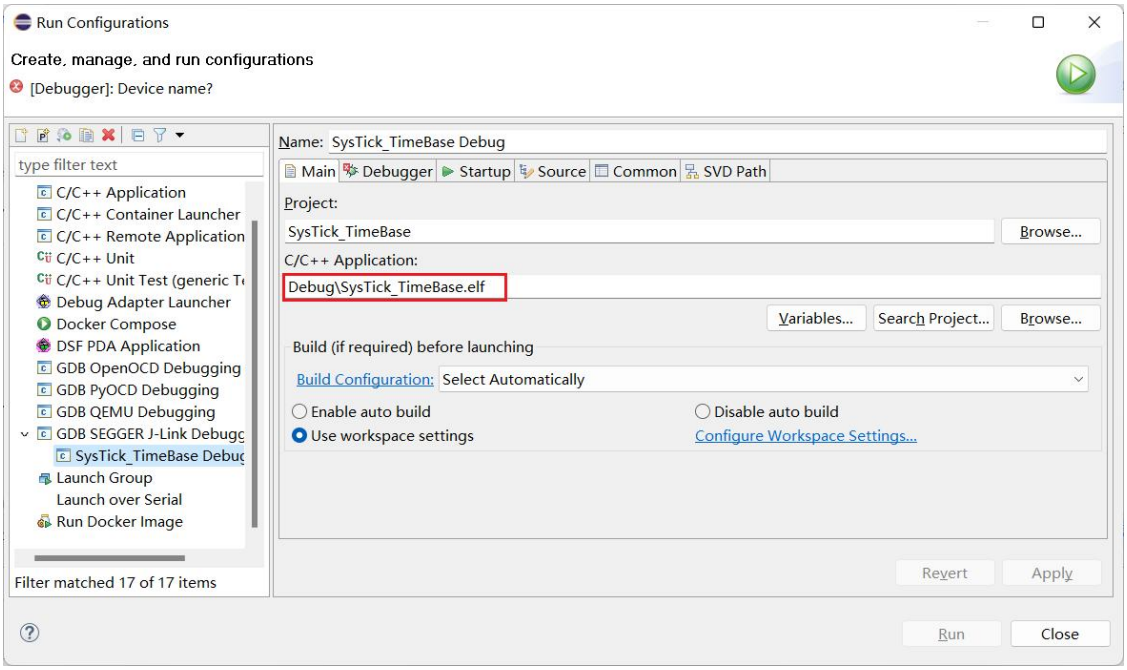
图 21 进入 Debug 配置界面



使用 GDB Server 和 GDB Client 作为 J-Link 配置选项工具，本例中，GDB Server 为 J-Link GDBServerCL，GDB Client 为 GCC 工具链中的 GDB 工具。若需新建一套 J-Link 配置选项，可双击 GDB SEGGER J-Link Debugging。

3.5.2 配置 Main 选项卡

图 22 J-Link Main 选项卡



在新建一套 J-Link 的配置选项后，main 选项卡一般会默认自动添加当前工程的 elf 文件。若没有，可点击右侧的 Browse，找到所在工程编译产生的 elf 文件，手动添加。

注意： 不同型号的芯片编译之后都会产生一个 elf 文件，若有多个芯片型号编译的需求，可对该芯片对应的不同型号都新建一套 Debug 配置。

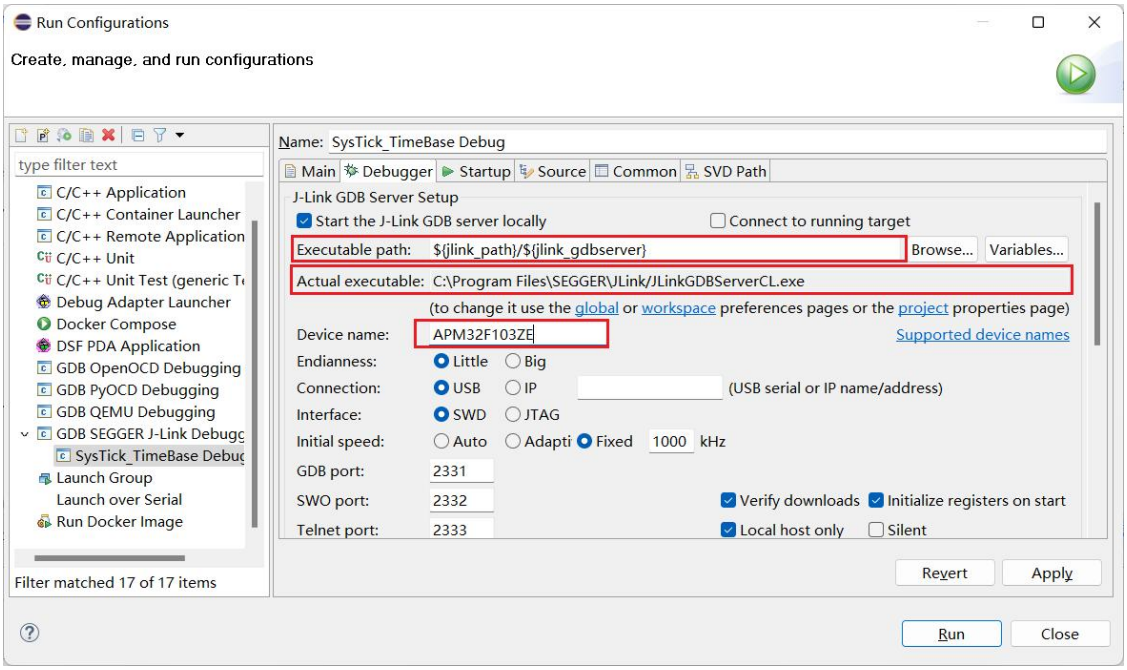
3.5.3 配置 Debugger 选项卡

在 Debugger 选项卡中，需在 Device name 中填写对应的芯片型号，本例中为 APM32F103ZE。

在搭建 Eclipse 环境时，若已正确配置 J-Link 路径，这里将自动识别。若 J-Link 路径配置失败，也可点击 Executable path 的 Browse，选择 J-Link GDBServerCL 的所在路径。

注意： 所配置的 J-Link 驱动中必须支持填写的芯片型号。

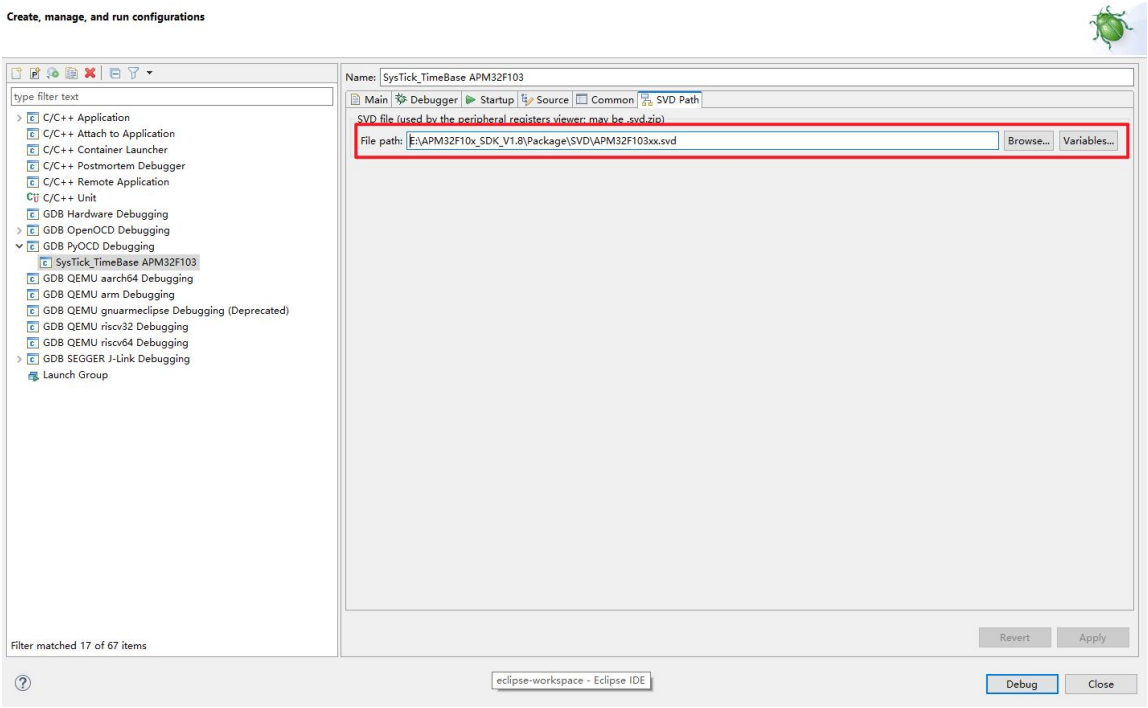
图 23 J-Link Debugger 选项卡



3.5.4 配置 SVD Path 选项卡

在 SVD Path 选项卡，选择目标芯片的 SVD 文件。 本例中，SVD 文件为 APM32F103xx.svd，默认路径为：APM32F103_SDK\Packages\SVD

图 24 J-Link SVD Path 选项卡

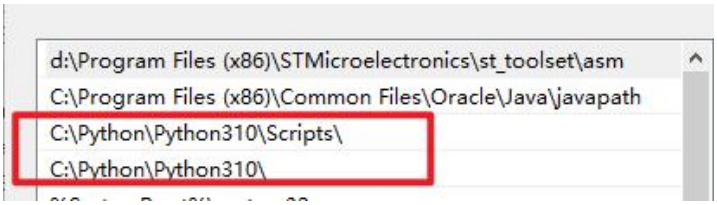


3.6 使用 Geehy-Link 下载及调试工程

3.6.1 配置 Pyocd 环境

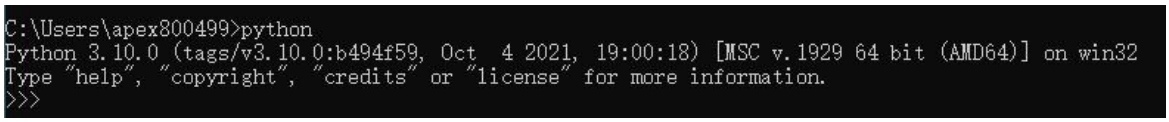
安装 python，官网下载地址 Welcome to Python.org，下载完后设置系统环境变量。

图 25 配置系统环境变量



打开 Windows 的 cmd，输入 python，若出现如下信息，则说明 python 安装成功。

图 26 python 安装成功

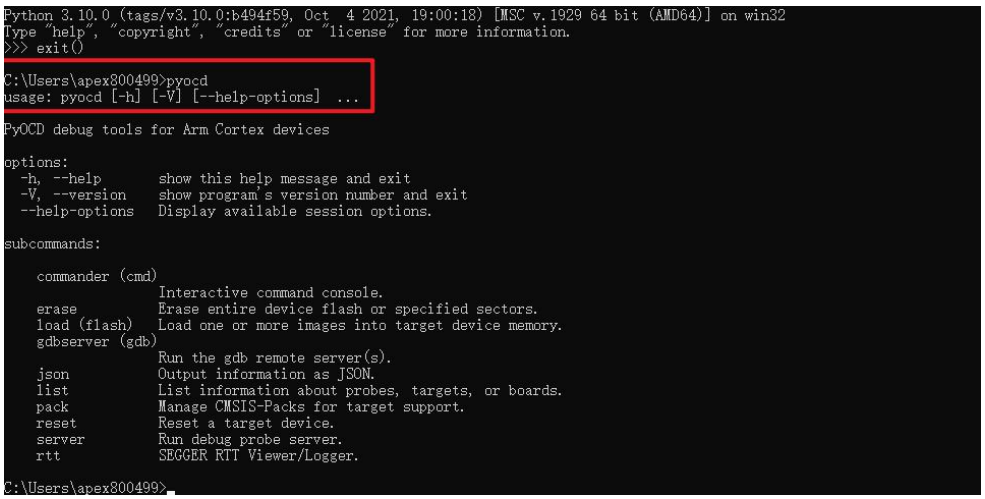


Python 安装成功，安装 pyocd 以及 usb 驱动库。

1. 退出 python cmd 输入 exit()
2. 安装 pyocd cmd 输入 pipinstall –U pyocd
3. 安装 libsub 库 cmd 输入 pip install –U libusb

Cmd 输入 pyocd，出现如下图信息则说明安装成功。

图 27 pyocd 安装成功



3.6.2 安装 pack 包

连接 Geehy-link 仿真器，cmd 输入 pyocd list，若出现以下内容，则说明连接成功。

图 28 连接 Geehy-link 仿真器

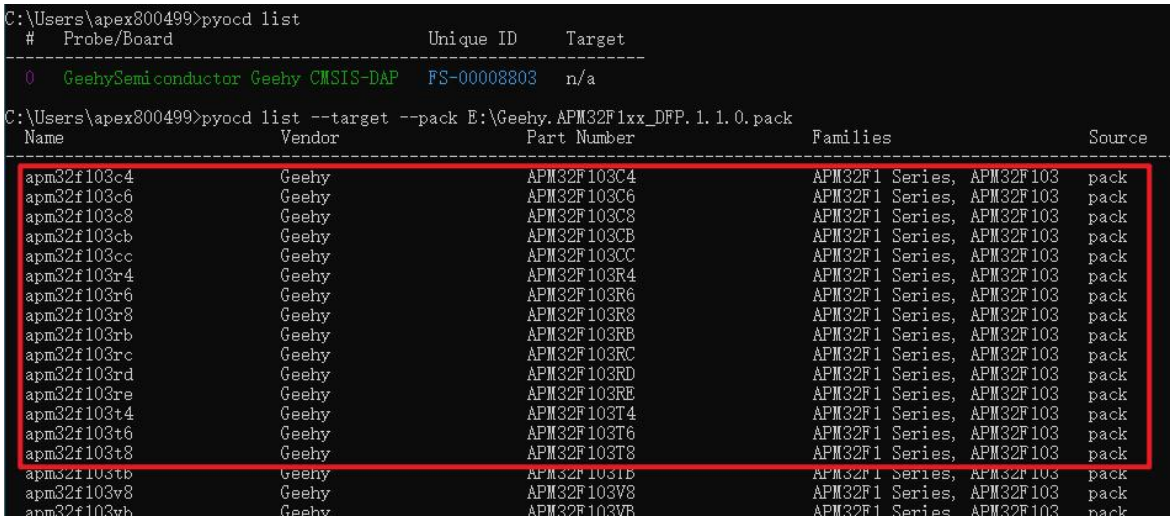
去 keil 官网（https://www.keil.arm.com/devices）或 geehy 官网（https://geehy.com/MCU）下载对应 apm32f1 系列的 pack 包。

Cmd 输入 pyocd list --target -–pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack，查看 apm32f1 系列 pack 包中支持的所有芯片型号。

图 29 查看 apm32f1 系列 pack 包

命令执行成功，出现以下信息，则说明信息获取成功。

图 30 apm32f103 芯片型号

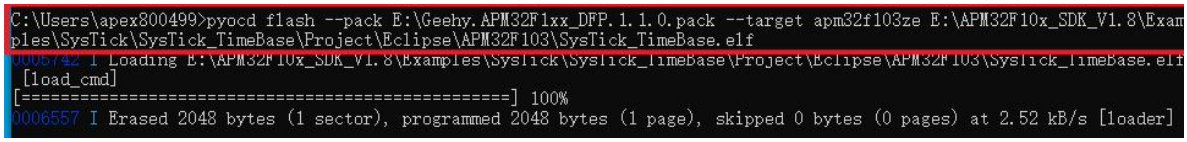


仿真器连接开发板后，Cmd 输入 pyocd erase --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack –t apm32f103ze --chip，可对芯片进行擦除。

图 31 apm32f103ze 芯片擦除

芯片擦除成功后，cmd 输入 pyocd flash --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack --target apm32f103ze +对应 hex(或 elf,bin)文件，对芯片进行代码烧录。

图 32 apm32f103ze 芯片烧录



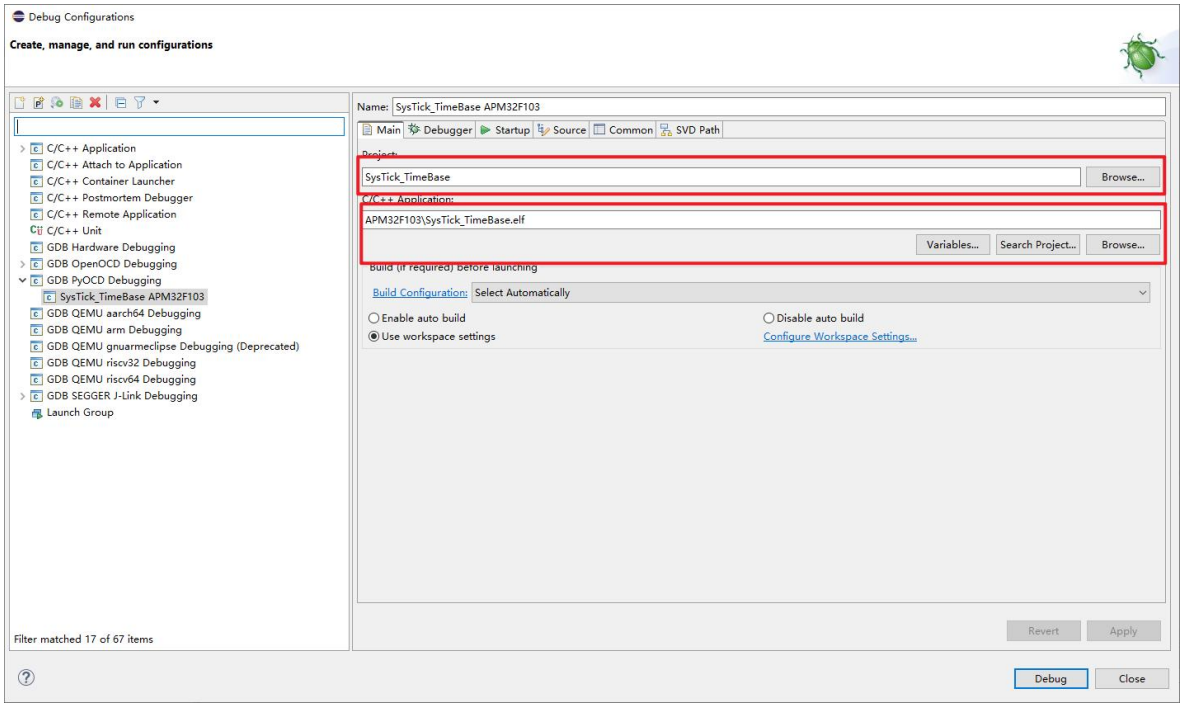
3.6.3 打开 Debug 配置界面

在菜单栏中，点击 Run，在弹出的选择框中选择 Debug Configurations，进入 Debug 配置界面。

使用 PyOCD 和 GDB Client 作为 Geehy-Link 配置选项工具，本例中，GDB Server 为 PyOCD，GDB Client 为 GCC 工具链中的 GDB 工具。若需新建一套 Geehy-Link 配置选项，可双击 GDB PyOCD Debugging。

3.6.4 配置 Main 选项卡

图 33 Geehy-Link Main 选项卡



在新建一套 Geehy-Link 的配置选项后，main 选项卡一般会默认自动添加当前工程的 elf 文件。若没有，可点击右侧的 Browse，找到所在工程编译产生的 elf 文件，手动添加。

注意： 不同型号的芯片编译之后都会产生一个 elf 文件，若有多个芯片型号编译的需求，可对该芯片对应的不同型号都新建一套 Debug 配置。

3.6.5 配置 Debugger 选项卡

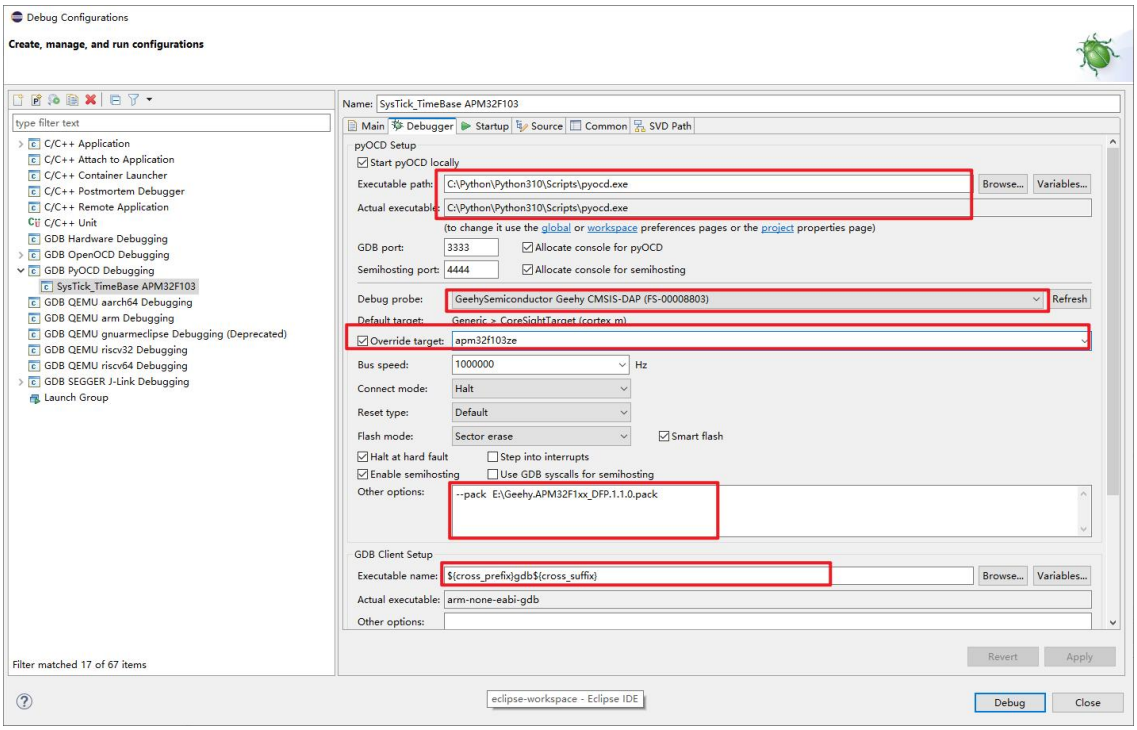
在搭建 Eclipse 环境时，若已正确配置 PyOCD 路径，这里将自动识别。若 PyOCD 路径配置失败，也可点击 Executable path 的 Browse，选择 PyOCD 的所在路径。在 Debug probe 栏中，选中所使用的 CMSIS-DAP 仿真器。

勾选 override target，选择对应的芯片型号，打开 Cmd 中输入以下命令进行查看支持的型号，这里使用的是 apm32f103ze。

pyocd list --target --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack

在 other options 添加 pack 包路径，--pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack。

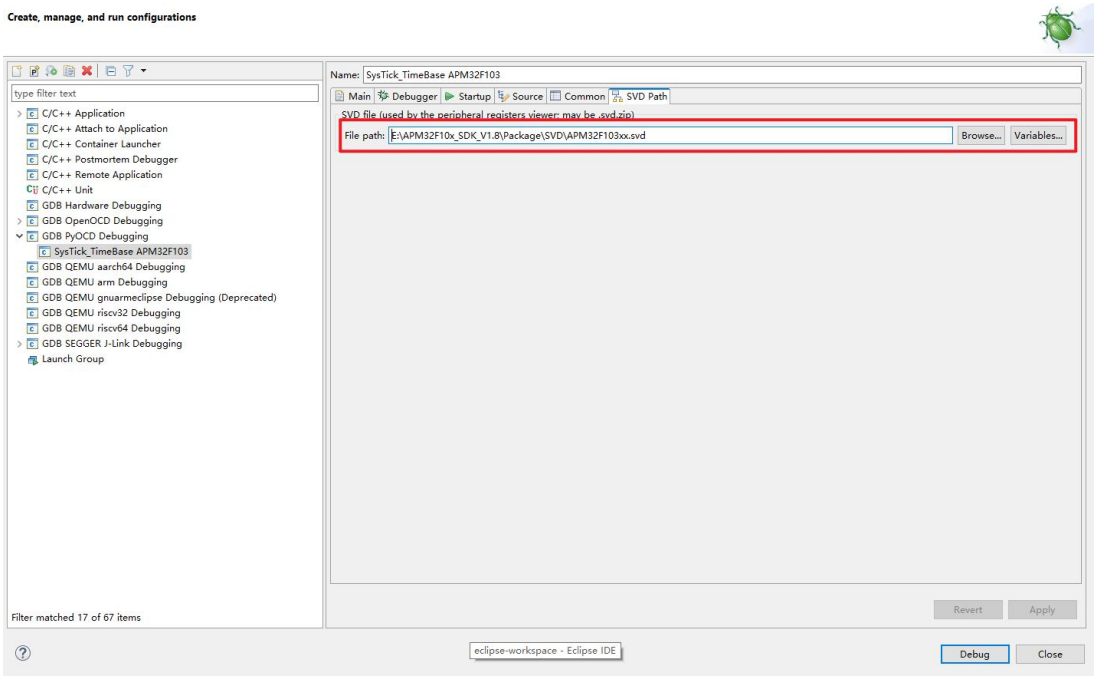
图 34 Geehy-Link Debugger 选项卡



3.6.6 配置 SVD Path 选项卡

在 SVD Path 选项卡，选择芯片所需的 SVD 文件。 本例中，使用 APM32F103xx.svd，其 SVD 默认文件路径为 APM32F10x_SDK\Package\SVD。

图 35 Geehy-Link SVD Path 选项卡



3.7 Debug 界面

若 Debug Configurations 配置完成，可点击 Debug 进行调试，出现如下图后，点击 Switch，进入 Debug 视图。

图 36 进入 Debug 视图

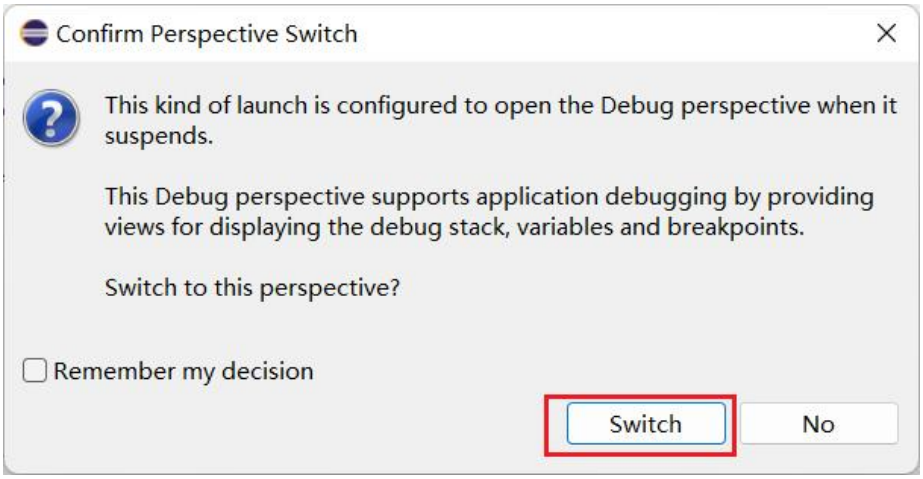
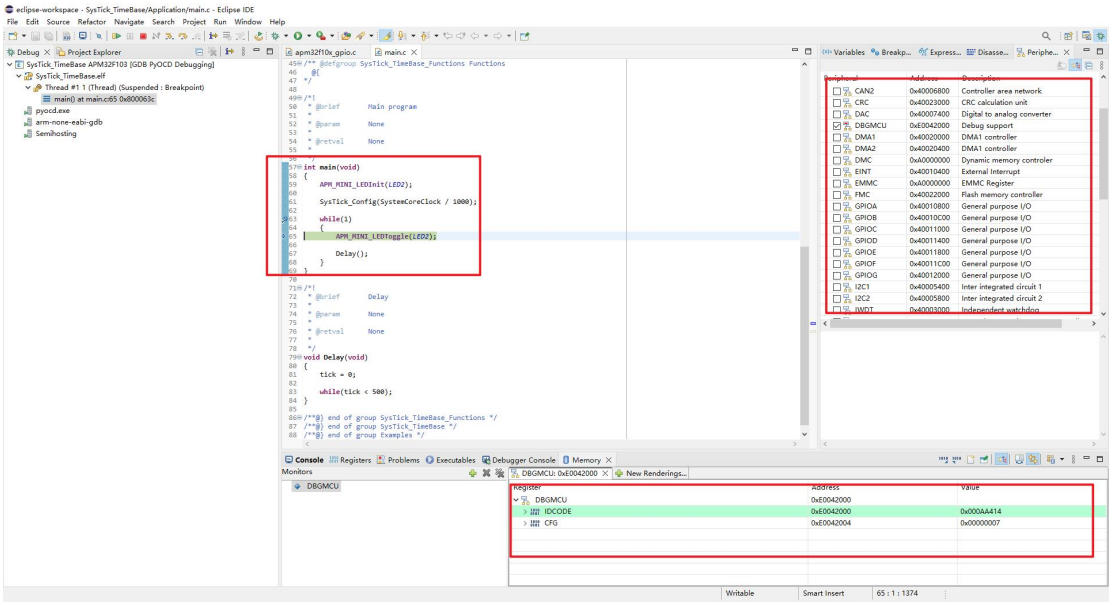


图 37 Debug 视图



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

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

图 38 Registers 选项

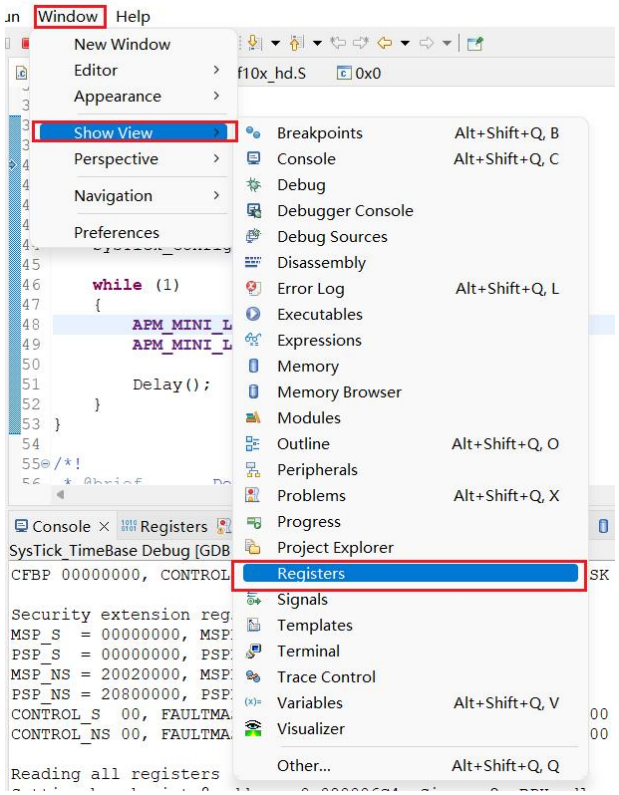
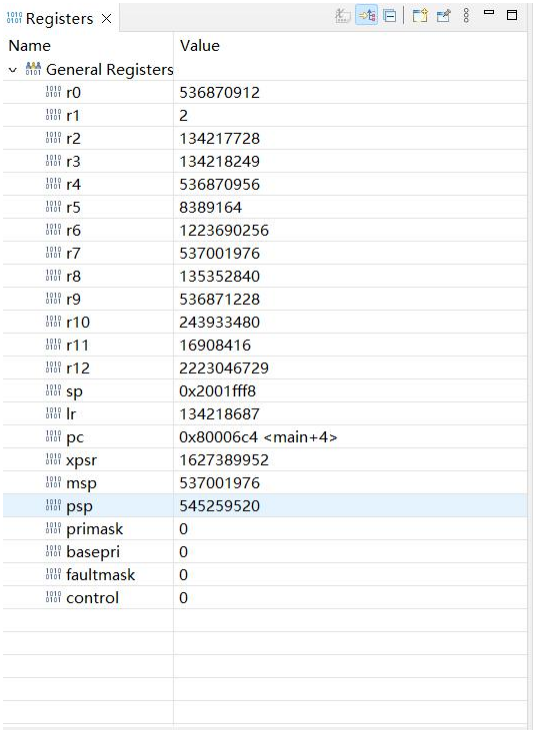


图 39 Registers 窗口



3.7.3 Peripherals 窗口

在菜单栏，若要查看外设寄存器的值，可点击 Window->Show view->Peripherals 选项。

图 40 打开 Peripherals 窗口

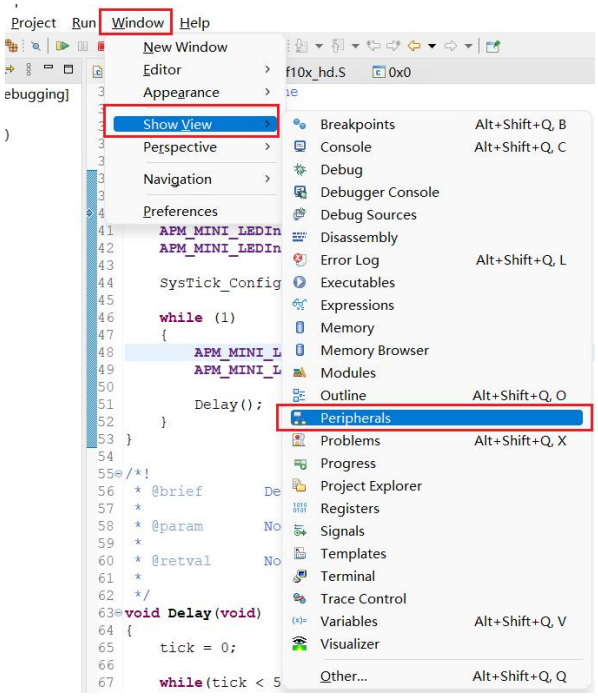
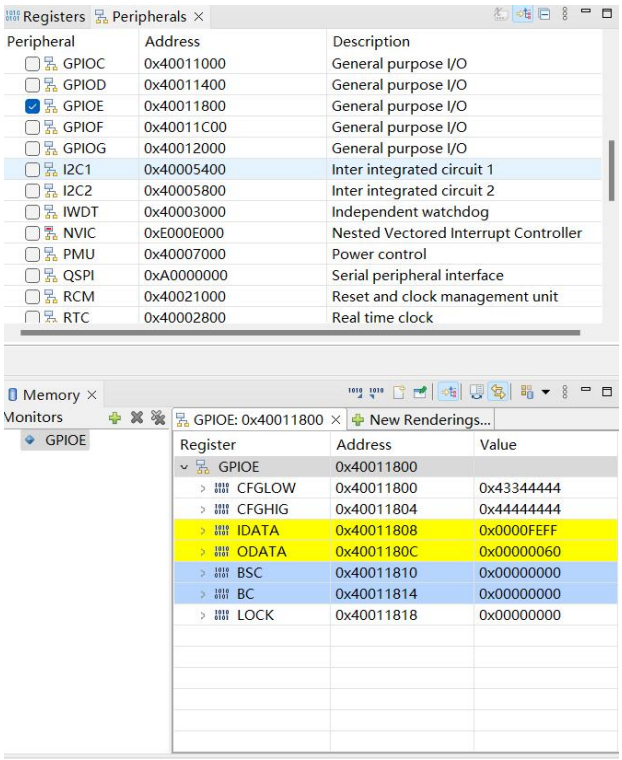


图 41 查看外设寄存器



3.7.4 Memory 窗口

若要查看寄存器或变量对应的 Memory 地址，可在菜单栏选择 “Window->Show View->Memory”，点击 Memory 窗口上方的“+”号，即可查看当前寄存器或变量所对应的 Memory 地址。

图 42 打开 Memory 窗口

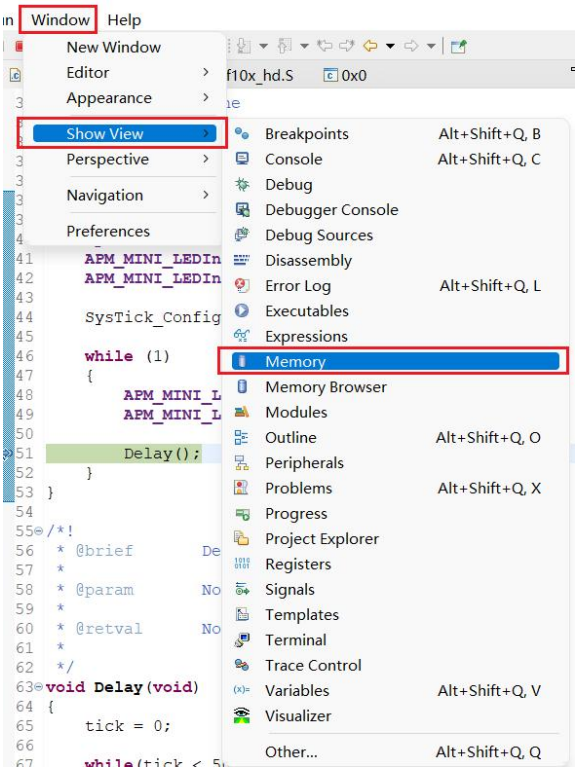
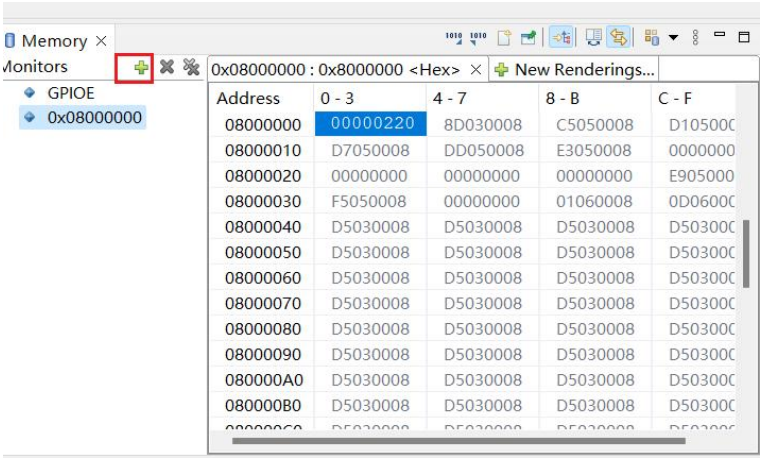


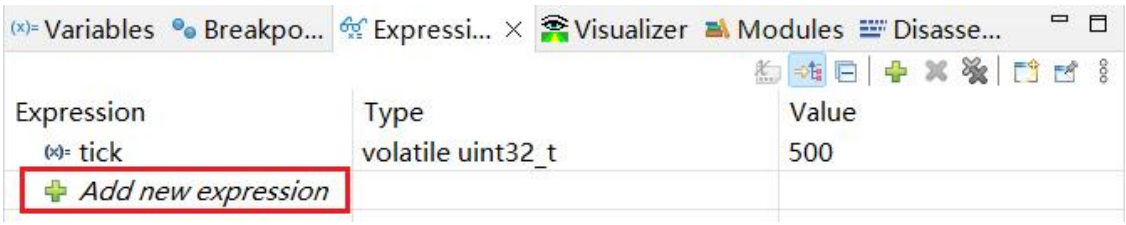
图 43 Memory 窗口



3.7.5 Expressions 窗口

若要查看相应变量的值，可在菜单栏选择 “Window->Show View->Expressions”， 点击 Expressions 窗口内“+”号，添加即可查看该变量。

图 44 Expressions 窗口

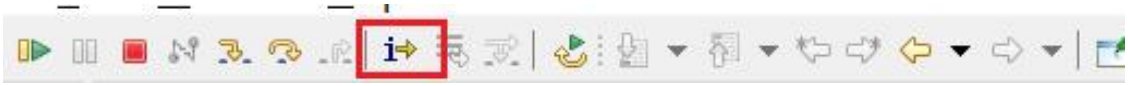


注意： 在 Eclipse 中，若程序全速运行，对应变量的值是无法实时更新的，只有在代码停止运行时才可查看该变量值的变化。

3.7.6 反汇编窗口

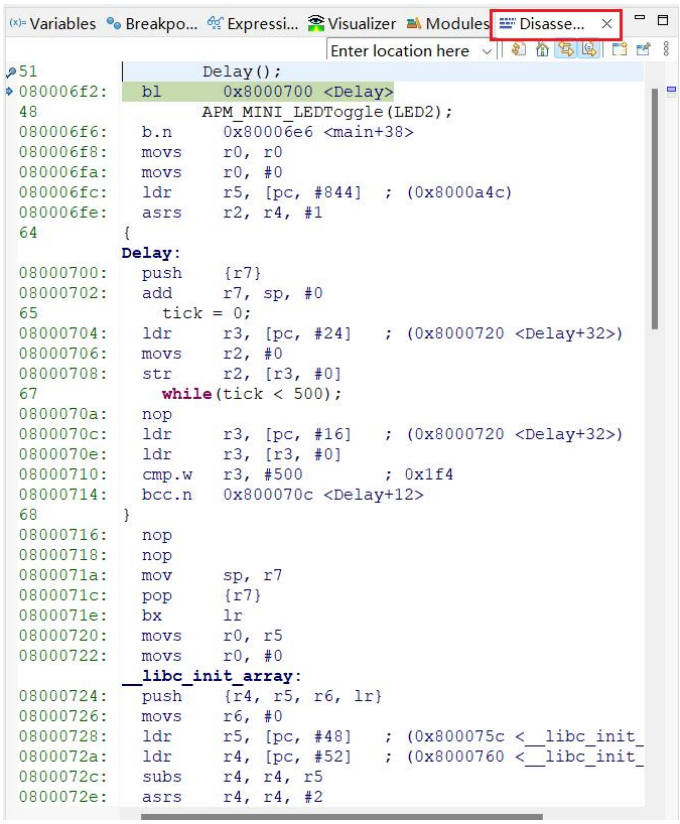
Debug 模式下，点击 Instruction Stepping Mode，打开反汇编窗口。

图 45 打开反汇编窗口



在反汇编窗口，可在其打断点，进行汇编指令的单步调试等功能。

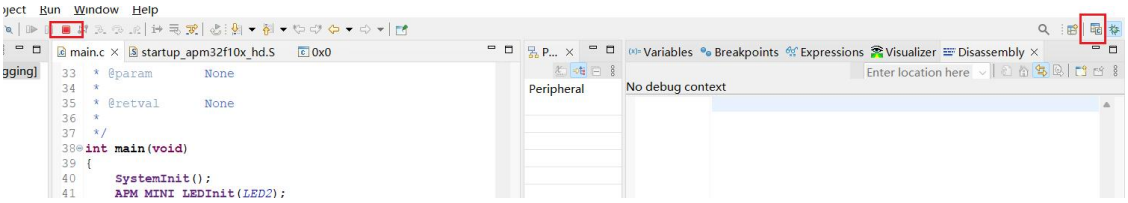
图 46 反汇编窗口



3.7.7 退出 Debug 视图

点击“停止调试”按钮，即可退出 Debug 视图， 若要进入工程视图，再点击右上角的“C/C++”。

图 47 进入工程视图



4 如何导入现有工程

Eclipse 可直接导入现有工程，在工程视图中，点击 File->Import，在弹出的选择视图中选择 General->Exisiting Projects into Workspace，点击“Next”，选择 Eclipse 工程路径，点击 finish，即可导入已有工程。

图 48 导入现有工程

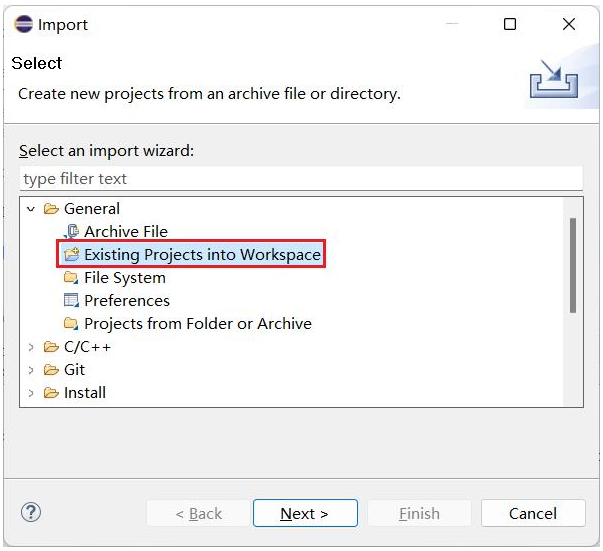
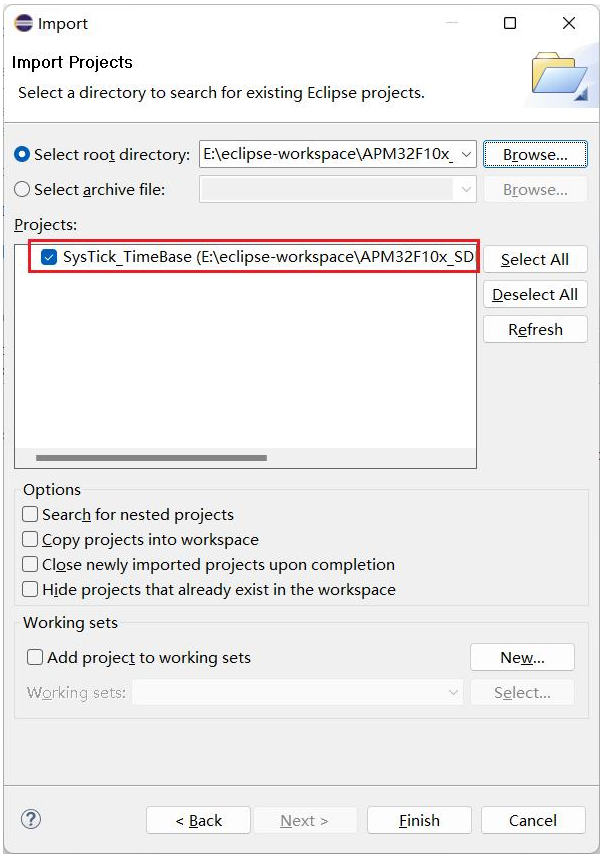


图 49 选择现有工程



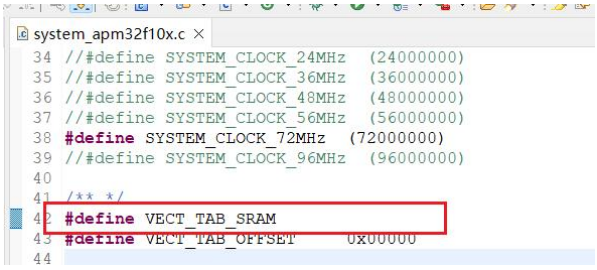
5 如何在 RAM 中调试

步骤 1：修改工程链接脚本，如下为修改的一个实例。

```
/* Specify the memory areas */
MEMORY
{
  FLASH (rx)      : ORIGIN = 0x20000000, LENGTH = 20K
  RAM (xrw)       : ORIGIN = 0x20005000, LENGTH = 108K
}
```

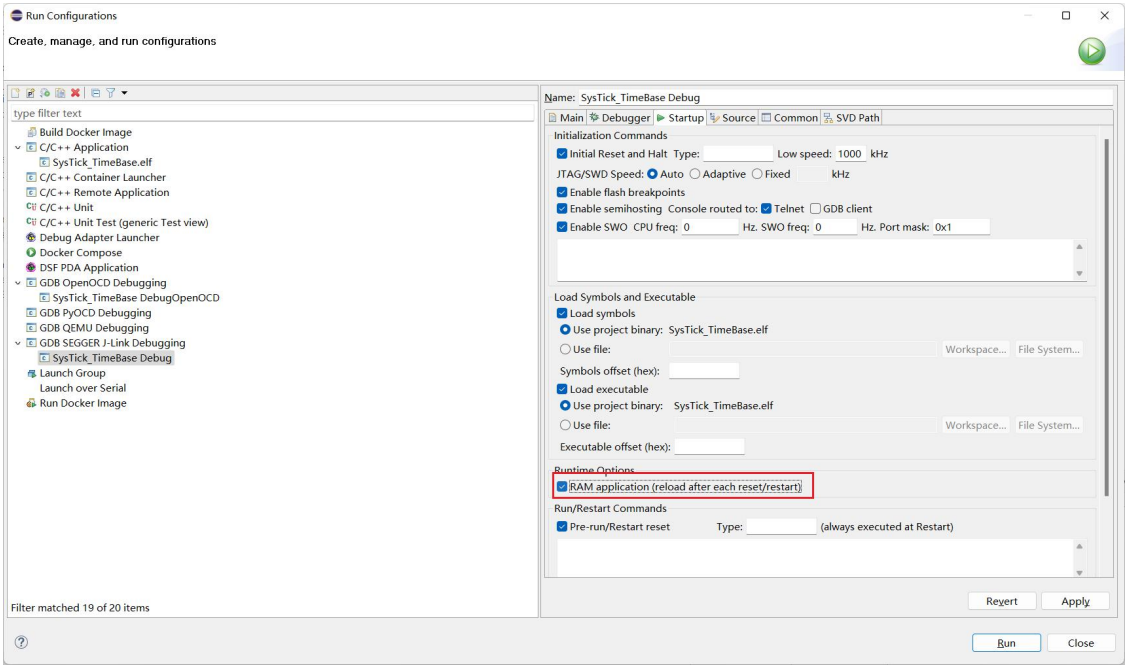
步骤 2：修改中断向量表，重定位到 SRAM。完成步骤 1 和步骤 2 后重新编译工程。

图 50 改变中断向量地址



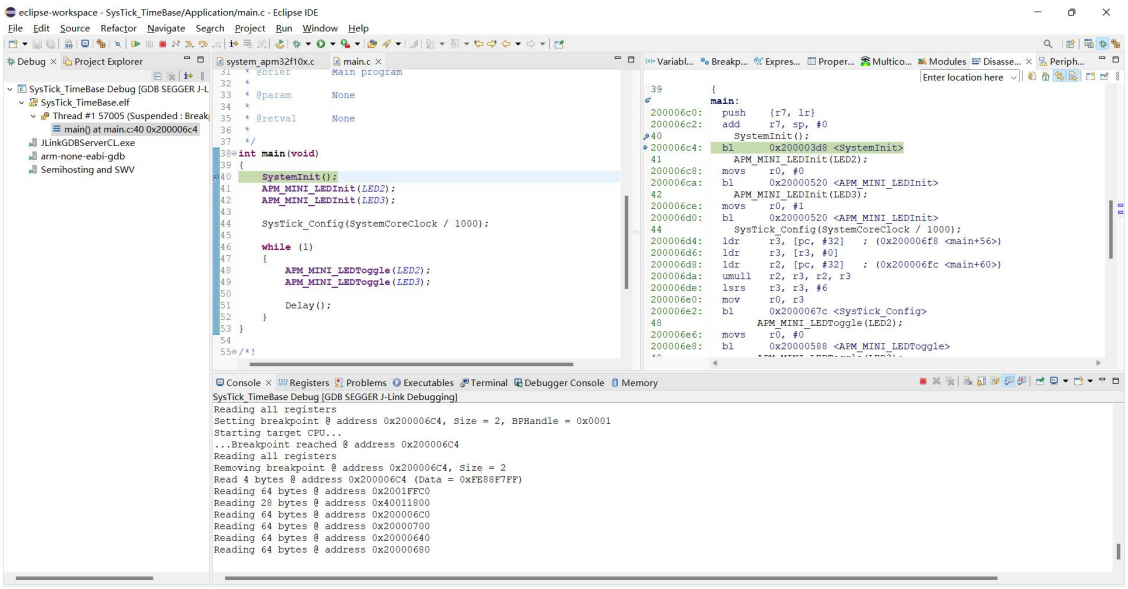
步骤 3：右键工程，选中 properties，在弹出的选项框中选择 Debug Configurations->Startup，勾选 RAM application。

图 51 勾选 RAM application



步骤 4： Debug 调试。下图为进入 RAM 中调试时的 Debug 视图。

图 52 RAM 中调试时的 Debug 视图



6 如何使用 printf 打印

6.1 使用步骤

步骤 1： 添加 syscall.c 文件，文件中添加如下代码，以添加_write 函数定义 ,将 usart 重定向到 __io_putchar 函数。

```
#include "apm32f10x_usart.h"

int _write(int file, char *ptr, int len)
{
    int DataIdx;
    for (DataIdx = 0; DataIdx < len; DataIdx++)
    {
        __io_putchar(*ptr++);
    }
    return len;
}

int __io_putchar(int ch)
{
    while (USART_ReadStatusFlag(USART1, USART_FLAG_TXBE) == RESET);
    USART_TxData(USART1, ch);
    return ch;
}
```

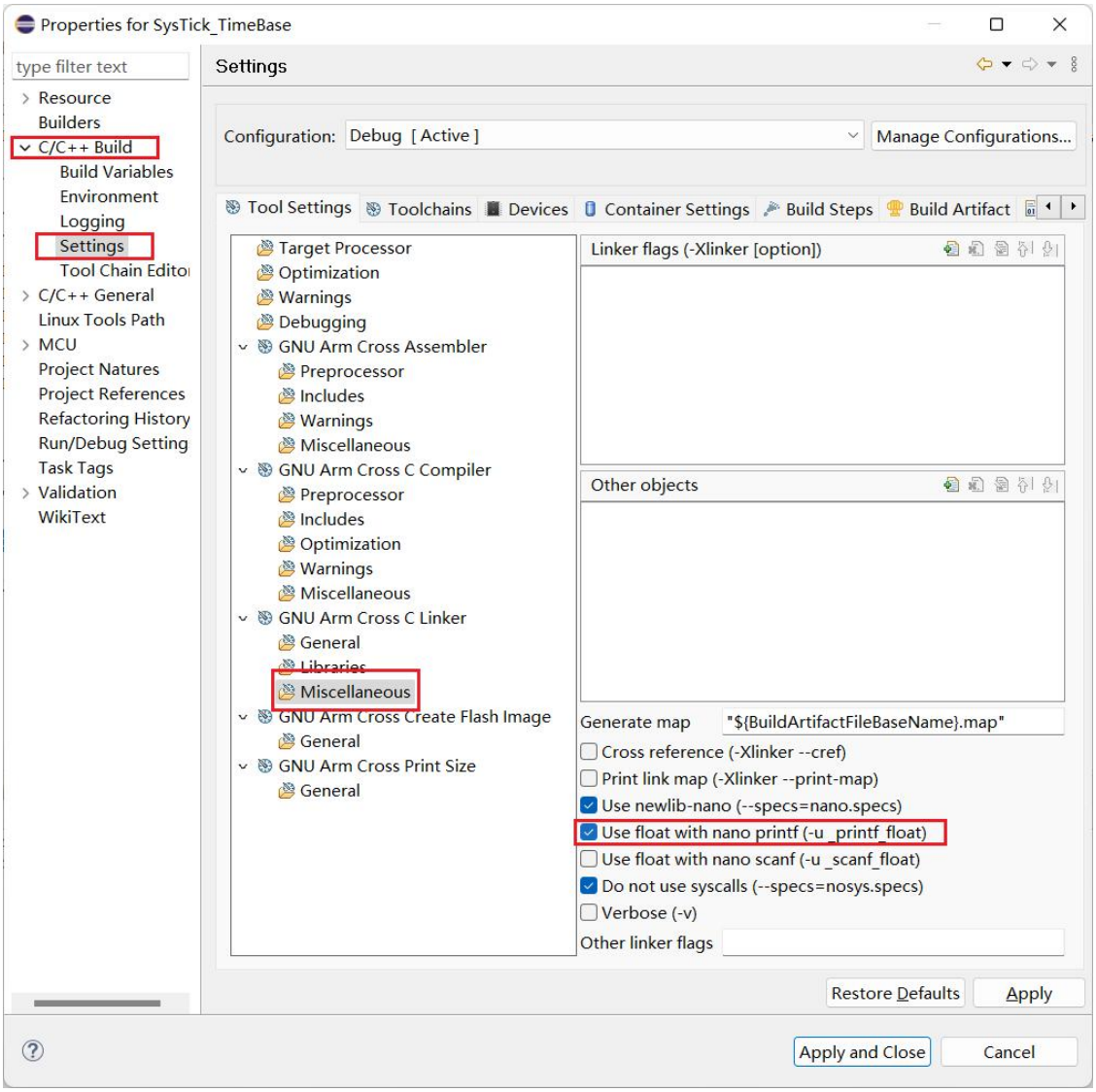
步骤 2： 使用 printf 函数， 即可正常打印。

```
printf("Runing \r\n");
```

6.2 打印浮点数据配置

在工程视图中，选中 File->Properties 选择项，在 C/C++ Build->Settings->Tool Settings->GNU Arm Cross C Linker->Miscellaneous 中，勾选-u _prinft_float 选项。

图 53 勾选-u _prinft_float 选项



注意：

1. 在使用 printf 打印时，打印内容的末尾需添加\r\n，例如 printf("Running!\r\n")。

2. 如果对代码的大小要求较高，不推荐使用 printf，因为 GCC 中使用 printf 会大大增加代码的大小。

7 修订历史

表格 1 文件修订历史

日期	修订	变化
2023.06.13	1.0	初稿发布

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“TM”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，除非在极海销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及／或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册相关信息而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失）。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。