Application Note

Application Note

Document No.: AN1093

Eclipse Development Tutorial under APM32 Arm MCU Windows System

Version: V1.0

1 Introduction

This application note provides users with application guidance for using Eclipse to develop APM32 series MCU under Windows system.

Contents

Figure Index

2 Development environment

- Development board: APM32F103ZEMINI development board
- Debugger: J-Link V9/V10 or Geehy-Link
- Operating system: WIN10 64 bit OS
- IDE：Eclipse IDE for Embedded C/C++ Developers Version: 2022-03 (4.23.0)
- Cross compilation chain: gcc-arm-none-eabi-10.3-2021.10-win32.exe
- Compilation tool: xpack windows build tools 4.3.0-1
- GDB server: Pyocd CMSIS-DAP / J-Link GDB Server CL V7.62c

3 Project development process

3.1 Process of building a new project

Open the Eclipse. Under "File>New", select to create a new C/C++ Project and select C Managed Build.

Figure 1 New Project C/C++Project

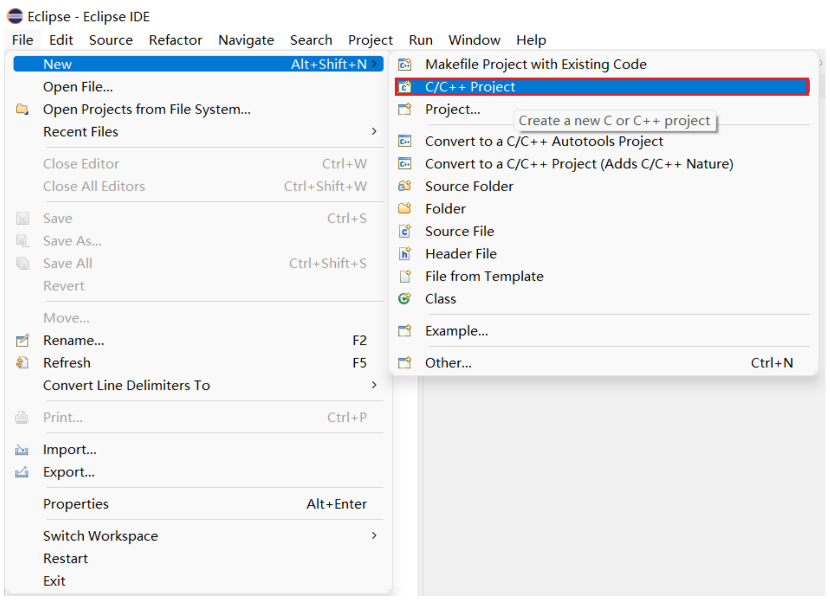
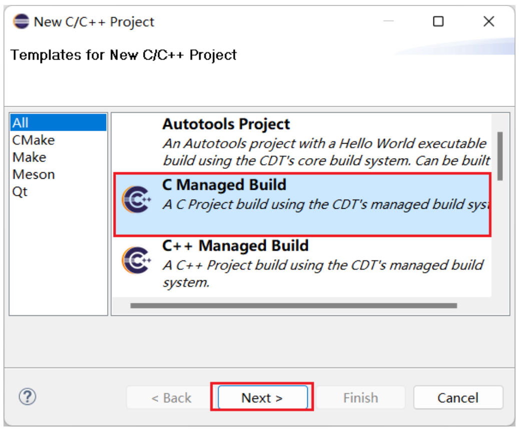


Figure 2 C Managed Build



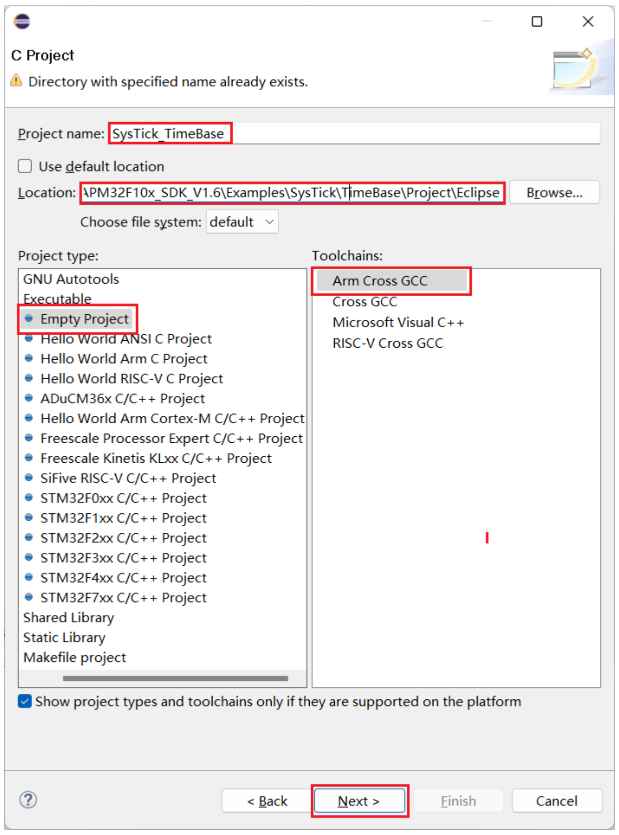
An Autotools project with a Hello World executable build using the CDT's core build system. Can be built

A C Project build using the CDT's managed build sys

A C++ Project build using the CDT's managed build system.

Enter the project name and configure the project type. It is recommended to place the project under the Project directory. Configure the compilation chain and select Arm Cross GCC as the compilation chain.

Figure 3 Configure a Project



C Project

Directory with specified name already exists.

Project name: SysTick_TimeBase

[ ] Use default location

Location: \APM32F10x_SDK_V1.6\Examples\SysTick\TimeBase\Project\Eclipse

Choose file system: default

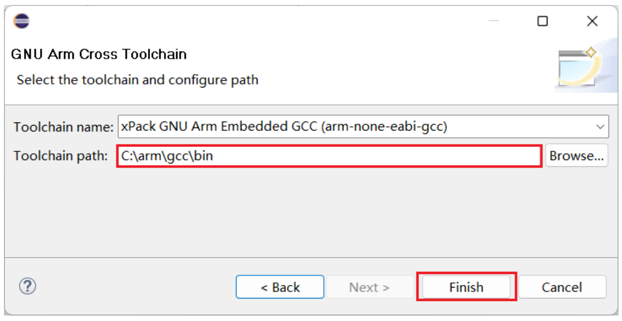
Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Hello World ANSI C Project
  - Hello World Arm C Project
  - Hello World RISC-V C Project
  - ADuCM36x C/C++ Project
  - Hello World Arm Cortex-M C/C++ Project
  - Freescale Processor Expert C/C++ Project
  - Freescale Kinetis KLxx C/C++ Project
  - SiFive RISC-V C/C++ Project
  - STM32F0xx C/C++ Project
  - STM32F1xx C/C++ Project
  - STM32F2xx C/C++ Project
  - STM32F3xx C/C++ Project
  - STM32F4xx C/C++ Project
  - STM32F7xx C/C++ Project
- Shared Library
- Static Library
- Makefile project

Toolchains:
- Arm Cross GCC
- Cross GCC
- Microsoft Visual C++
- RISC-V Cross GCC

[✓] Show project types and toolchains only if they are supported on the platform

If Arm Toolchains of the Eclipse IDE has been correctly configured, the path will be automatically selected here. If it has not been configured correctly, click Browse to select the corresponding absolute path.

Figure 4 Set Arm Toolchains Path



GNU Arm Cross Toolchain

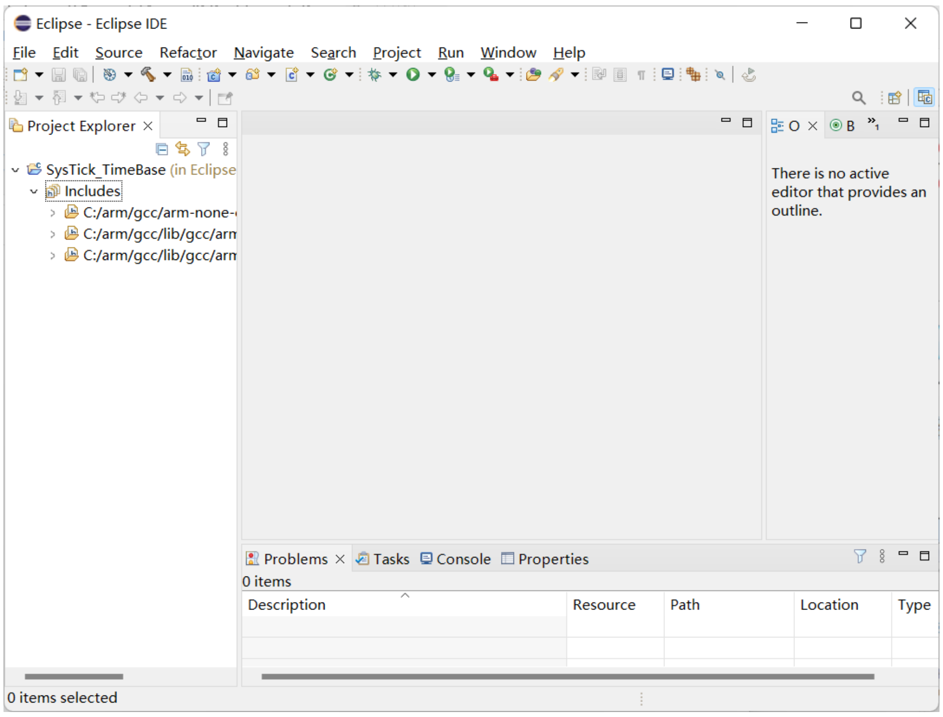
Select the toolchain and configure path

Toolchain name: xPack GNU Arm Embedded GCC (arm-none-eabi-gcc)

Toolchain path: C:\arm\gcc\bin

Click "Finish" until the displayed interface is the view as shown in Figure 5. Then the establishment of the Project is completed.

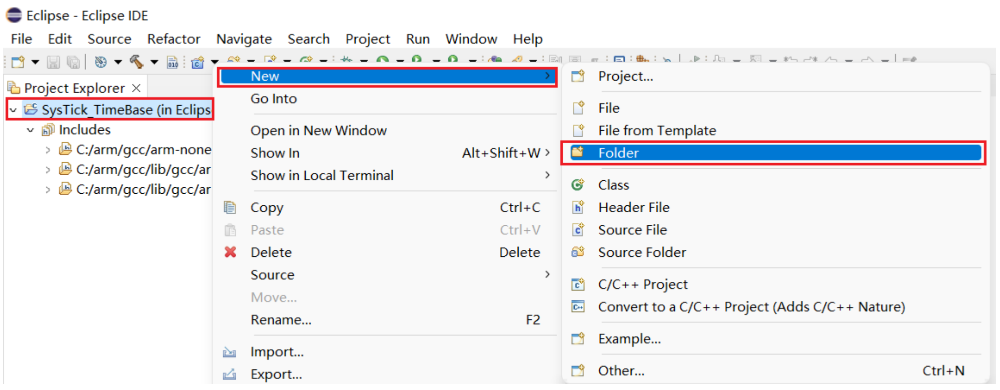
Figure 5 Complete Project Establishment



3.2 Project folder and adding a file

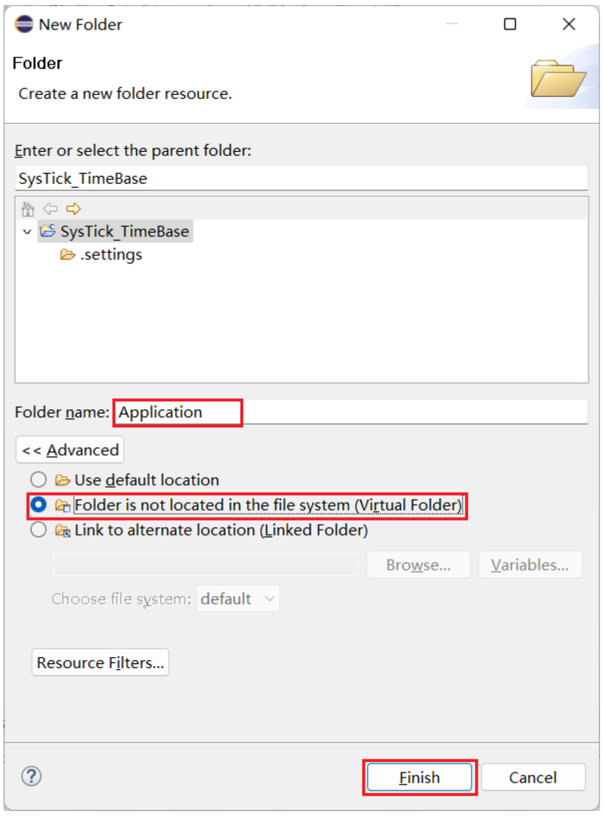
3.2.1 Manual mode

Figure 6 Add a Folder



Right-click the project folder to create the virtual folder Application.

Figure 7 Create a Virtual Folder Application



New Folder

Folder

Create a new folder resource.

Enter or select the parent folder:

SysTick_TimeBase

SysTick_TimeBase
.settings

Folder name: Application

<< Advanced

Use default location

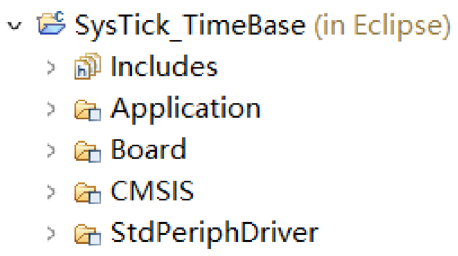
Folder is not located in the file system (Virtual Folder)

Link to alternate location (Linked Folder)

Choose file system: default

Similarly, create CMSIS, Board, and StdPeriphDriver folders under this project directory.

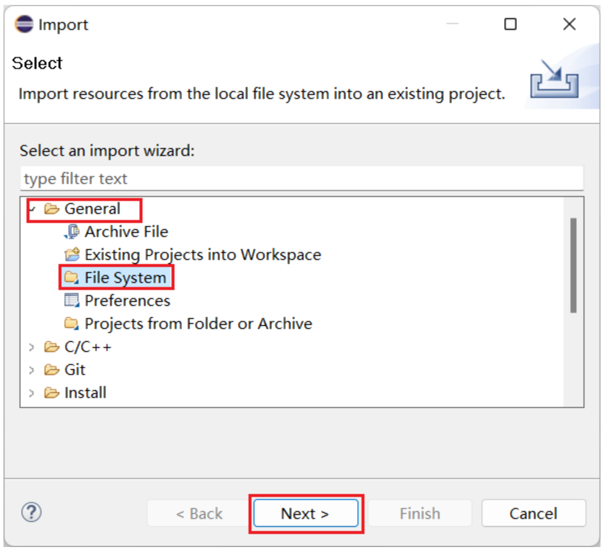
Figure 8 Create CMSIS, Board, and StdPeriphDriver Folders



SysTick_TimeBase (in Eclipse)
- Includes
- Application
- Board
- CMSIS
- StdPeriphDriver

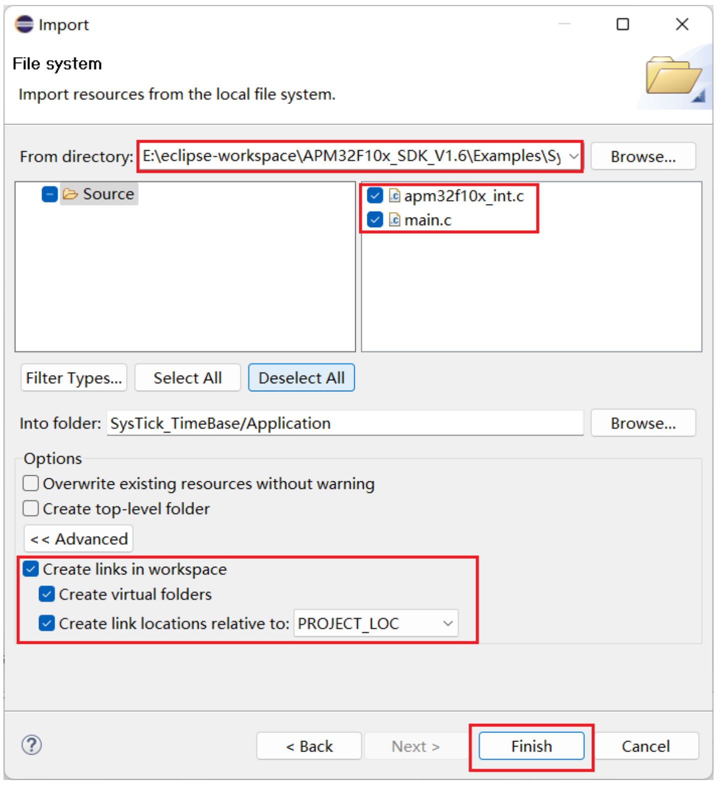
Select the Application folder, right-click to select the Import option and directly import the file.

Figure 9 Import a File


Import
Select
Import resources from the local file system into an existing project.
Select an import wizard:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
Projects from Folder or Archive
C/C++
Git
Install
< Back
Next >
Finish
Cancel

When importing a file, select "File System" as the import mode. In the pop-up file path selection dialog box, select the file path to be imported, and check the file to be imported.

Figure 10 Check the File to be Imported



From directory: E:\eclipse-workspace\APM32F10x_SDK_V1.6\Examples\S)

Source

- [✓] apm32f10x_int.c
- [✓] main.c

Into folder: SysTick_TimeBase/Application

Options

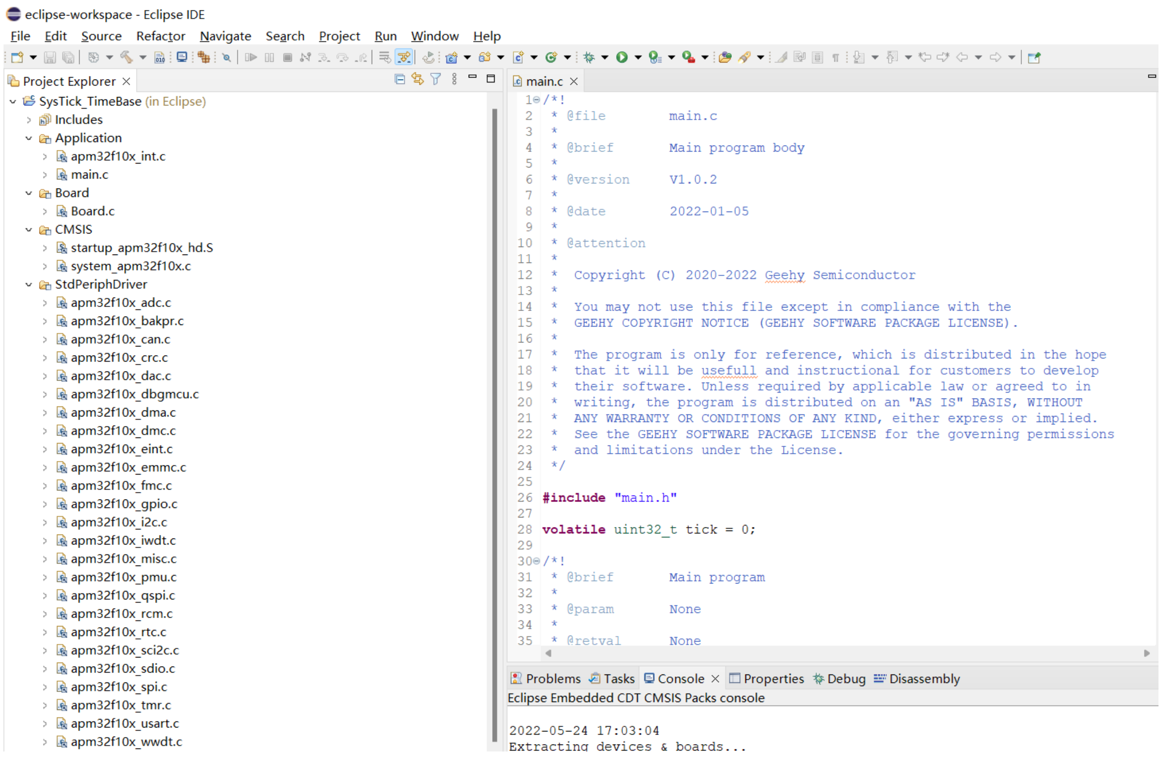
- [ ] Overwrite existing resources without warning
- [ ] Create top-level folder

<< Advanced

- [✓] Create links in workspace
  - [✓] Create virtual folders
  - [✓] Create link locations relative to: PROJECT_LOC

Similarly, import the required files of CMSIS, Board, and StdPeriphDriver folders.

Figure 11 Import All Files


eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
main.c
Problems Tasks Console Properties Debug Disassembly

```
SysTick_TimeBase (in Eclipse)
  Includes
  Application
    apm32f10x_int.c
    main.c
  Board
    Board.c
  CMSIS
    startup_apm32f10x_hd.S
    system_apm32f10x.c
  StdPeriphDriver
    apm32f10x_adc.c
    apm32f10x_bakpr.c
    apm32f10x_can.c
    apm32f10x_crc.c
    apm32f10x_dac.c
    apm32f10x_dbgmcu.c
    apm32f10x_dma.c
    apm32f10x_dmc.c
    apm32f10x_eint.c
    apm32f10x_emmc.c
    apm32f10x_fmc.c
    apm32f10x_gpio.c
    apm32f10x_i2c.c
    apm32f10x_iwdt.c
    apm32f10x_misc.c
    apm32f10x_pmu.c
    apm32f10x_qspi.c
    apm32f10x_rcm.c
    apm32f10x_rtc.c
    apm32f10x_sci2c.c
    apm32f10x_sdio.c
    apm32f10x_spi.c
    apm32f10x_tmr.c
    apm32f10x_usart.c
    apm32f10x_wwdt.c
```

```c
/*!
 * @file        main.c
 *
 * @brief       Main program body
 *
 * @version     V1.0.2
 *
 * @date        2022-01-05
 *
 * @attention
 *
 *  Copyright (C) 2020-2022 Geehy Semiconductor
 *
 *  You may not use this file except in compliance with the
 *  GEEHY COPYRIGHT NOTICE (GEEHY SOFTWARE PACKAGE LICENSE).
 *
 *  The program is only for reference, which is distributed in the hope
 *  that it will be usefull and instructional for customers to develop
 *  their software. Unless required by applicable law or agreed to in
 *  writing, the program is distributed on an "AS IS" BASIS, WITHOUT
 *  ANY WARRANTY OR CONDITIONS OF ANY KIND, either express or implied.
 *  See the GEEHY SOFTWARE PACKAGE LICENSE for the governing permissions
 *  and limitations under the License.
 */

#include "main.h"

volatile uint32_t tick = 0;

/*!
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
```

Eclipse Embedded CDT CMSIS Packs console

```
2022-05-24 17:03:04
Extracting devices & boards...
```

3.2.2 "Refresh" method

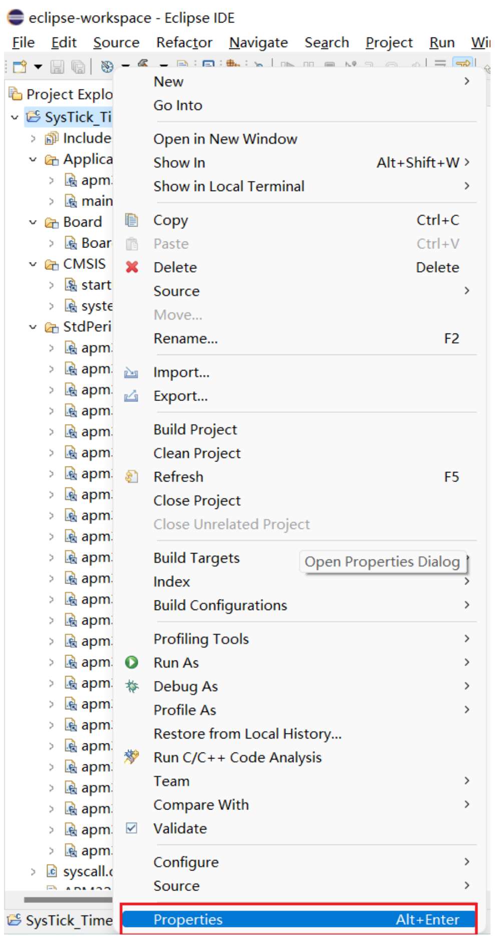
In addition to manually importing the folders mentioned above, files and folders can be directly placed in the same-level directory as the .cproject. In the Eclipse IDE, right-click the project name and select "refresh" to import the folders and files into the project.

Note: Since the files and folders created by "refresh" are stored in the disk, if a file is deleted from the Eclipse IDE, the corresponding file in the disk will also be deleted.

3.3 Project configuration process

Right-click the project and select Properties in the pop-up option box.

Figure 12 Open Properties Option

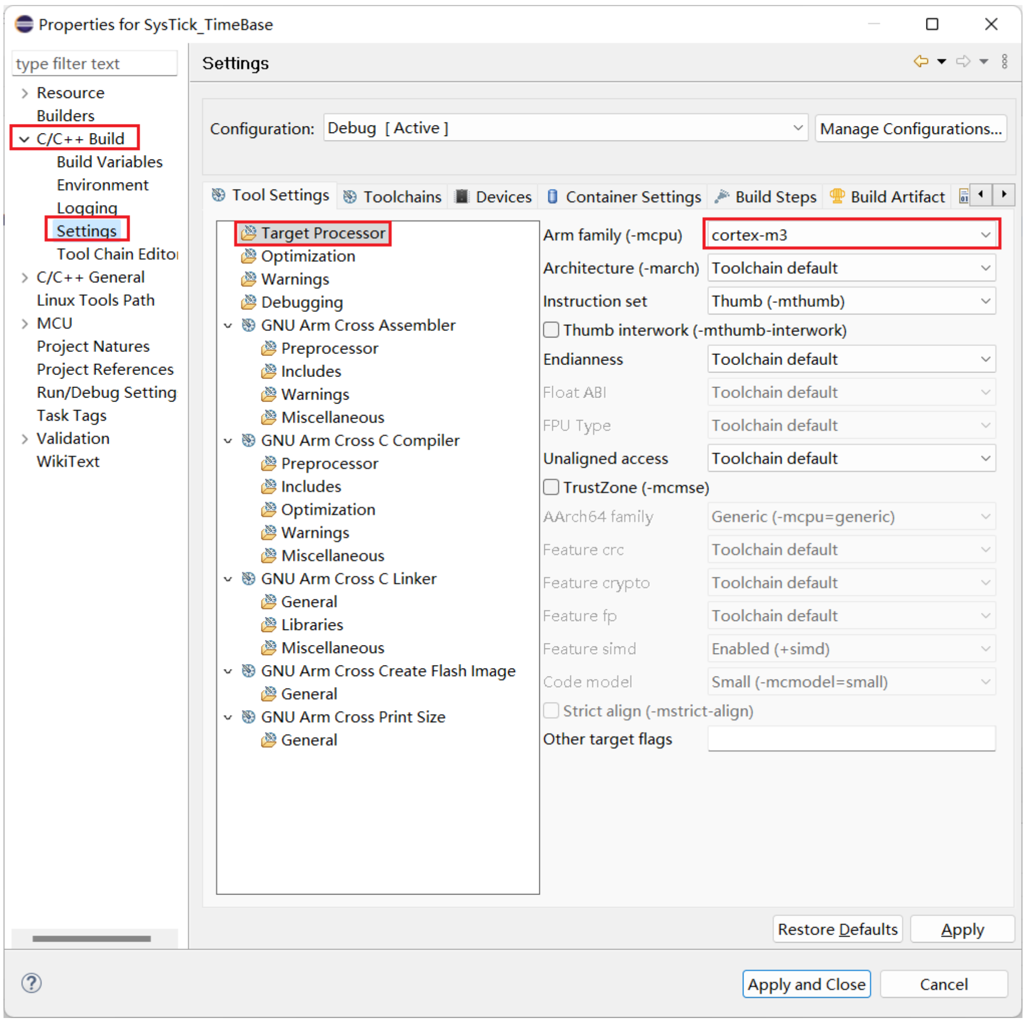


3.3.1 Configure Target Processor tab

The configuration for "C/C++ Build->Settings->Tool Settings->Target Processor" is as follows:

Select the corresponding core according to the target chip, such as cortex-m3, cortex-m4, cortex-m23 or cortex-m33. The chip core used in this example is cortex-m3.

Figure 13 Target Processor Configuration



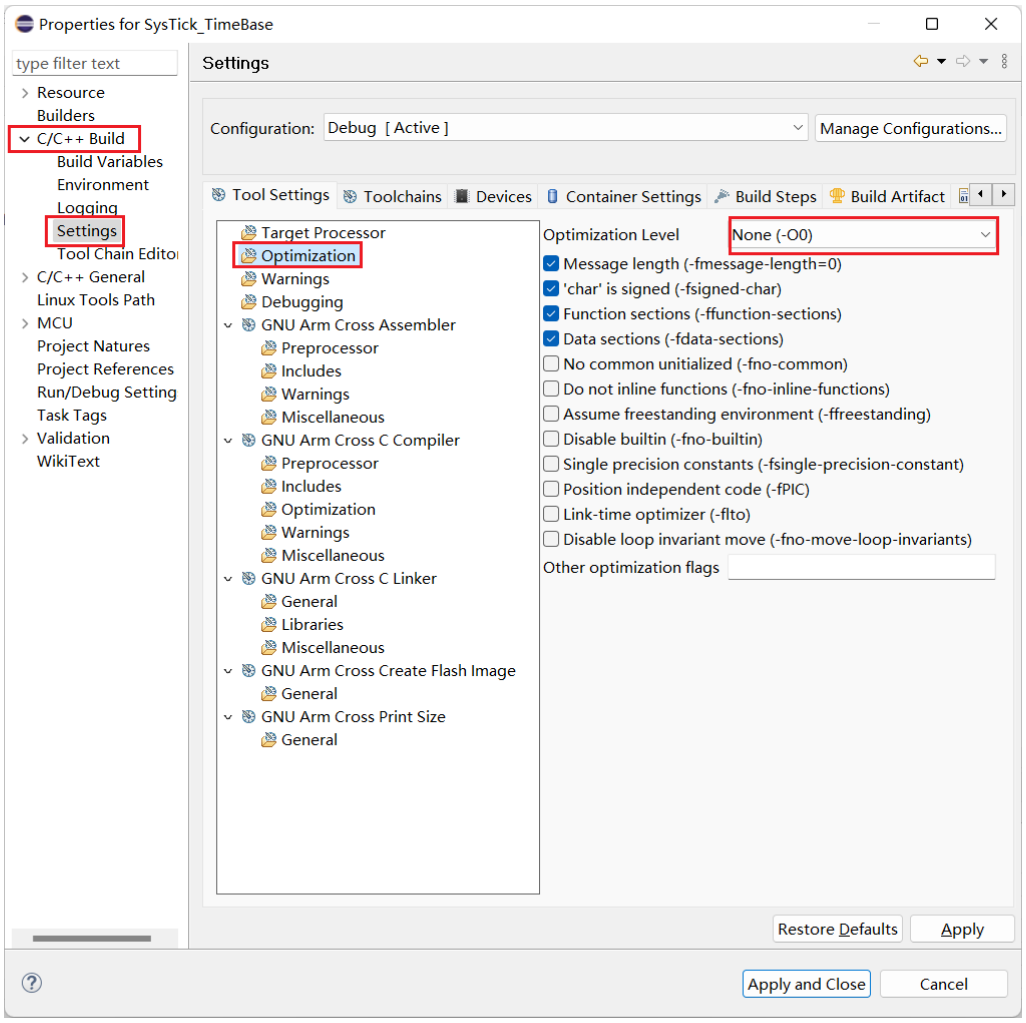
Configuration: Debug [Active]

Arm family (-mcpu)	cortex-m3
Architecture (-march)	Toolchain default
Instruction set	Thumb (-mthumb)
☐ Thumb interwork (-mthumb-interwork)	
Endianness	Toolchain default
Float ABI	Toolchain default
FPU Type	Toolchain default
Unaligned access	Toolchain default
☐ TrustZone (-mcmse)	
AArch64 family	Generic (-mcpu=generic)
Feature crc	Toolchain default
Feature crypto	Toolchain default
Feature fp	Toolchain default
Feature simd	Enabled (+simd)
Code model	Small (-mcmodel=small)
☐ Strict align (-mstrict-align)	
Other target flags	

3.3.2 Configure Optimization tab

In the "C/C++ Build->Settings->Tool Settings->Optimization" option, different optimization levels can be configured for the chip, such as -O0, -O1, -O2, -O3, -Os, -Ofast and -Og.

Figure 14 Optimization Configuration



Optimization Level: None (-O0)

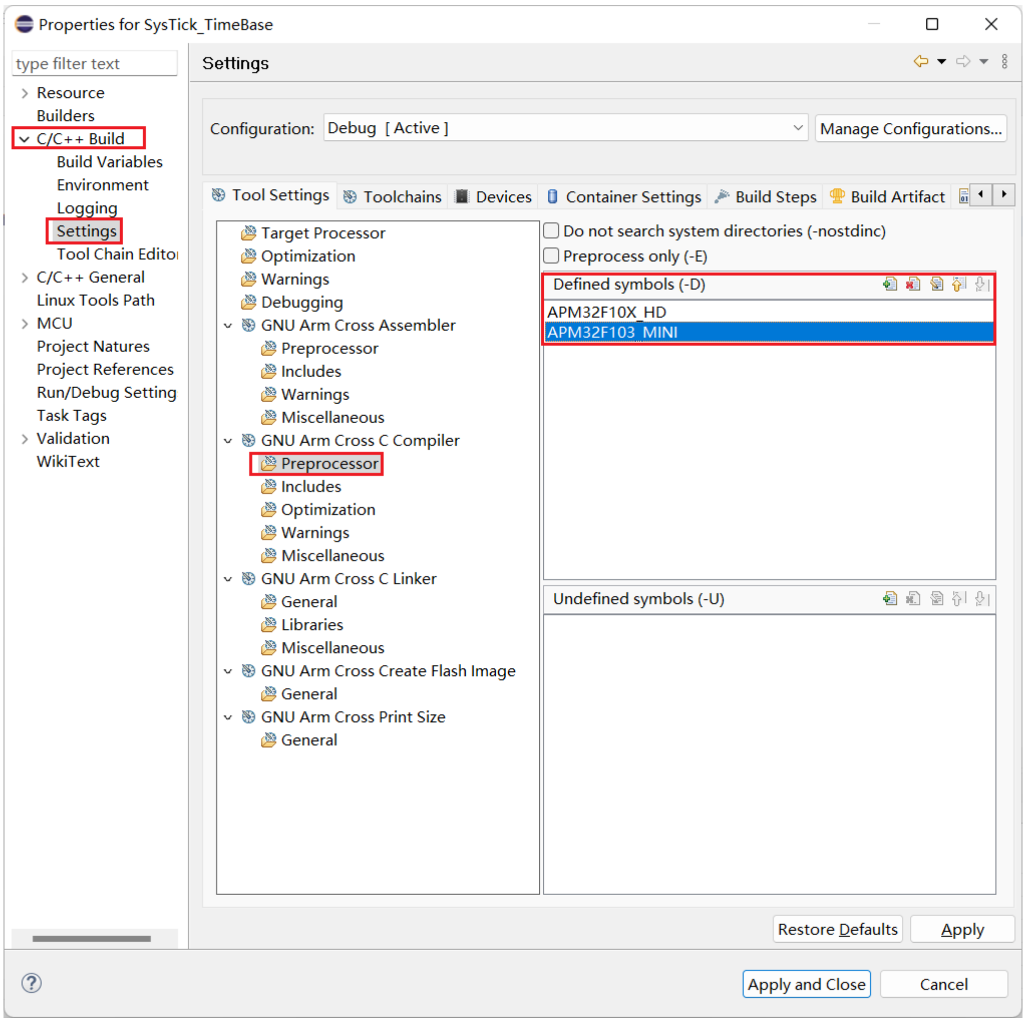
- [x] Message length (-fmessage-length=0)
- [x] 'char' is signed (-fsigned-char)
- [x] Function sections (-ffunction-sections)
- [x] Data sections (-fdata-sections)
- [ ] No common unitialized (-fno-common)
- [ ] Do not inline functions (-fno-inline-functions)
- [ ] Assume freestanding environment (-ffreestanding)
- [ ] Disable builtin (-fno-builtin)
- [ ] Single precision constants (-fsingle-precision-constant)
- [ ] Position independent code (-fPIC)
- [ ] Link-time optimizer (-flto)
- [ ] Disable loop invariant move (-fno-move-loop-invariants)

Other optimization flags

3.3.3 Configure GNU Arm Cross C Compiler tab

Configure “C/C++ Build->Settings->Tool Settings->GNU Arm Cross C Compiler” option. In this example, add pre-compile macro: APM32F10X_HD、APM32F103_MINI in “Preprocessor->Defined symbols” option.

Figure 15 GNU Arm Cross C Compiler Configuration



Configuration: Debug [Active]

Do not search system directories (-nostdinc)

Preprocess only (-E)

Defined symbols (-D)

APM32F10X_HD

APM32F103_MINI

Undefined symbols (-U)

The header file required by the project is added in the "includes->Include paths" option. In this example, the path of the added header file is as follows:

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

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

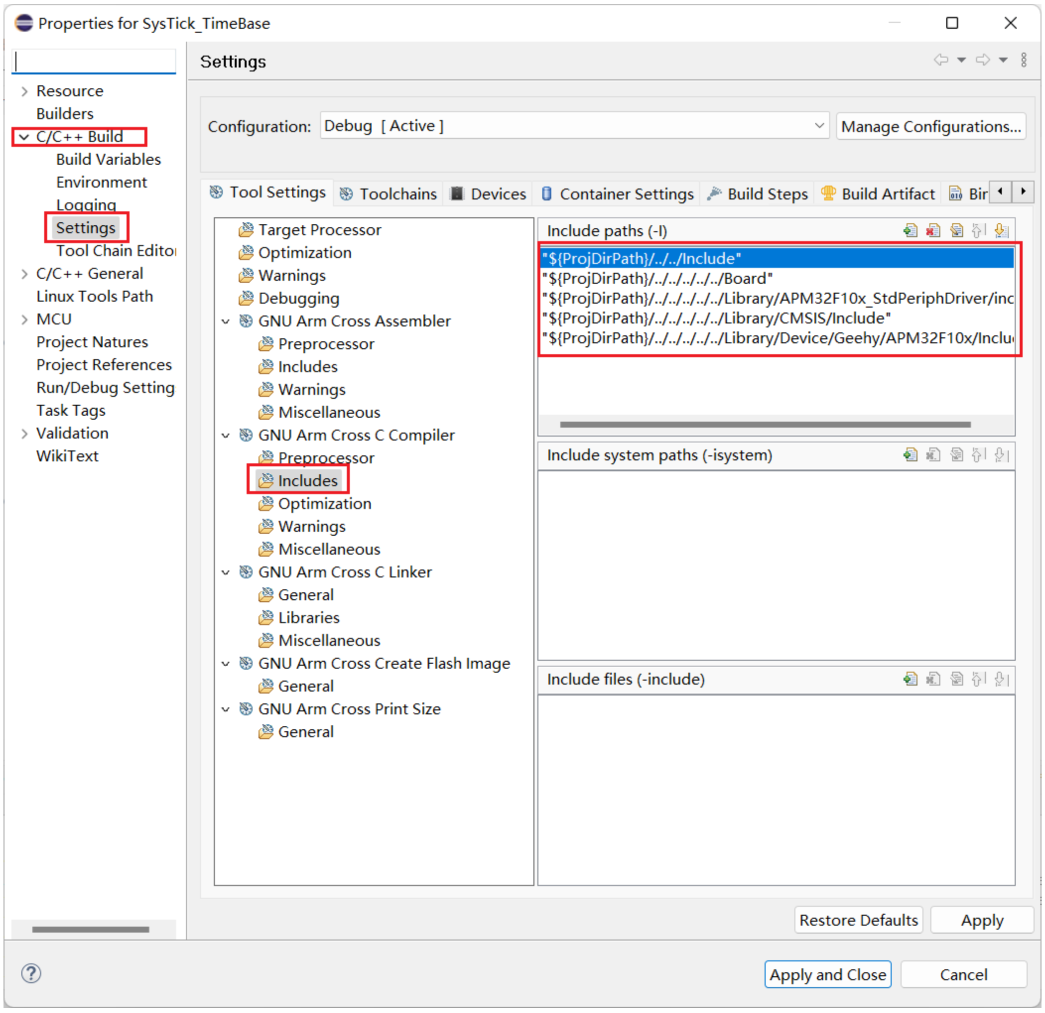
"${ProjDirPath}/../../../../../Library/APM32F10x_StdPeriphDriver/inc"

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

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

Note: Path addition includes relative path addition and absolute path addition. In this example, relative path addition is used.

Figure 16 Add the Header File Path Required by the Project



Configuration: Debug [Active]

Include paths (-I):

```
"${ProjDirPath}/../../Include"
"${ProjDirPath}/../../../../Board"
"${ProjDirPath}/../../../../../Library/APM32F10x_StdPeriphDriver/inc
"${ProjDirPath}/../../../../../Library/CMSIS/Include"
"${ProjDirPath}/../../../../../Library/Device/Geehy/APM32F10x/Inclu
```

3.3.4 Configure GNU Arm Cross C Linker tab

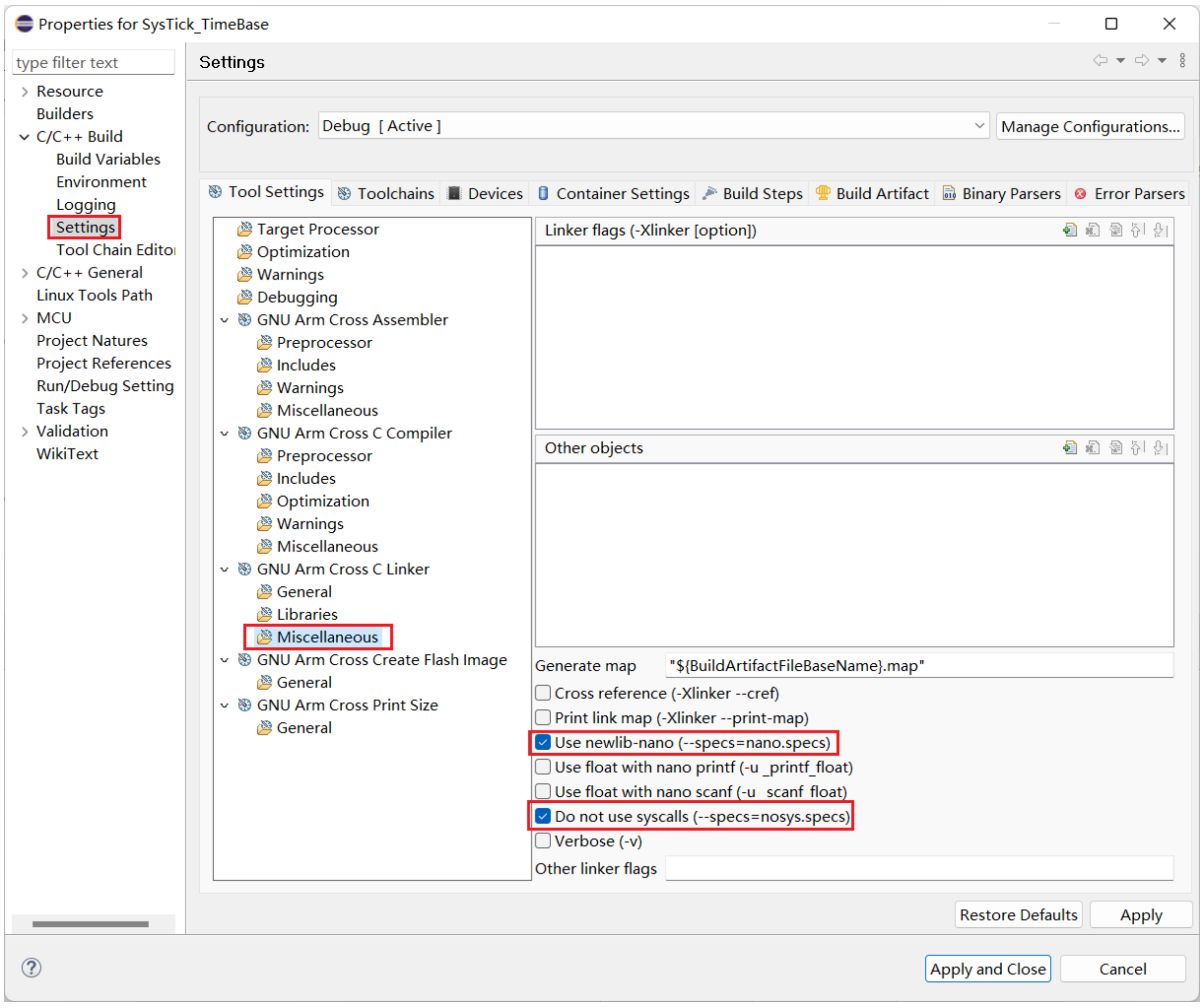
Configure "C/C++ Build->Settings->Tool Settings->GNU Arm Cross C Linker" chain option. In this example, in “General ->Script files” option, add: "${workspace_loc:/${ProjName}/gcc_APM32F10xxE.ld}"

ld script is a script file used by linker ld, which describes the internal layout of the program, defines the symbols for linking, and specifies the start address and size of storage areas such as code segment and data segment. The ld script used shall be consistent with the FLASH and RAM size of the target chip, and the memory configuration required by the customer.

Note: For the ld file added in this example, not only can the above relative path addition be used, but also the absolute path can be directly added.

Check the Use newlib-nano and Do not use syscalls of Miscellaneous option to optimize the code size.

Figure 17 GNU Arm Cross C Linker Configuration



Properties for SysTick_TimeBase

type filter text

- Resource
- Builders
- C/C++ Build
  - Build Variables
  - Environment
  - Logging
  - Settings
  - Tool Chain Editor
- C/C++ General
- Linux Tools Path
- MCU
- Project Natures
- Project References
- Run/Debug Setting
- Task Tags
- Validation
- WikiText

Settings

Configuration: Debug [Active]   Manage Configurations...

Tool Settings | Toolchains | Devices | Container Settings | Build Steps | Build Artifact | Binary Parsers | Error Parsers

- Target Processor
- Optimization
- Warnings
- Debugging
- GNU Arm Cross Assembler
  - Preprocessor
  - Includes
  - Warnings
  - Miscellaneous
- GNU Arm Cross C Compiler
  - Preprocessor
  - Includes
  - Optimization
  - Warnings
  - Miscellaneous
- GNU Arm Cross C Linker
  - General
  - Libraries
  - Miscellaneous
- GNU Arm Cross Create Flash Image
  - General
- GNU Arm Cross Print Size
  - General

Linker flags (-Xlinker [option])

Other objects

Generate map: "${BuildArtifactFileBaseName}.map"

- [ ] Cross reference (-Xlinker --cref)
- [ ] Print link map (-Xlinker --print-map)
- [✓] Use newlib-nano (--specs=nano.specs)
- [ ] Use float with nano printf (-u _printf_float)
- [ ] Use float with nano scanf (-u _scanf_float)
- [✓] Do not use syscalls (--specs=nosys.specs)
- [ ] Verbose (-v)

Other linker flags

Restore Defaults | Apply

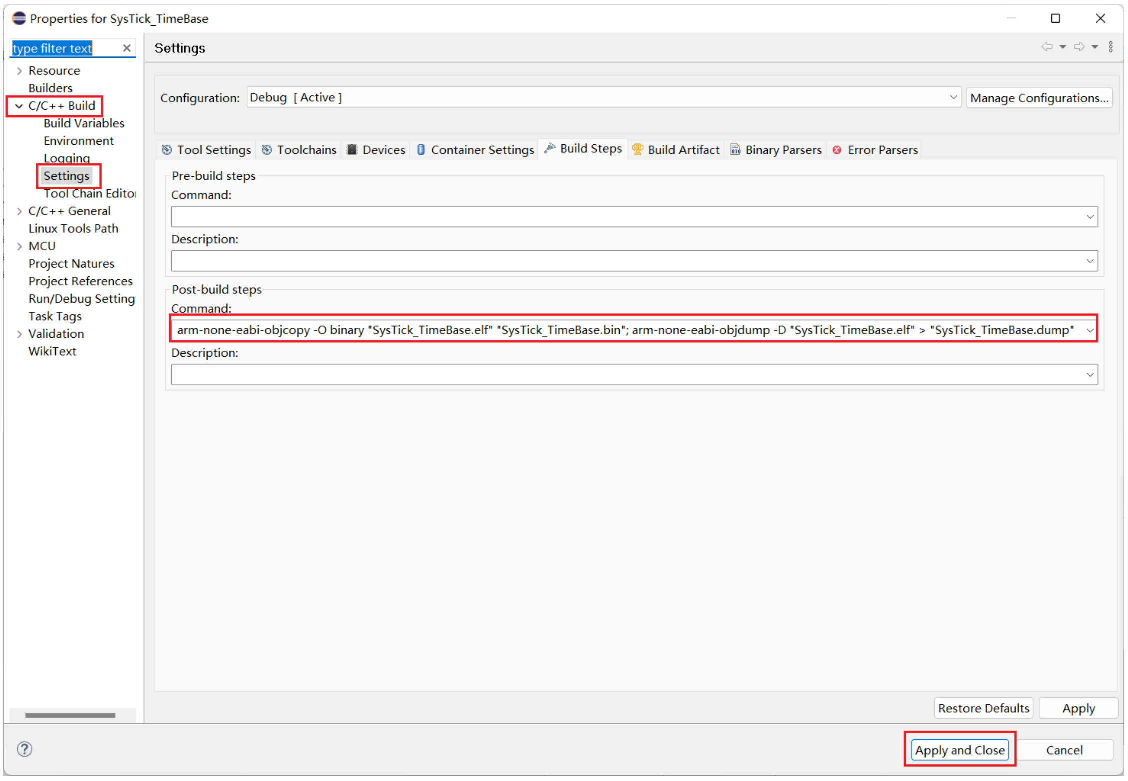
Apply and Close | Cancel

3.3.5 Configure Build Steps tab - Generate bin file

To generate a bin/hex file, add relevant command in "C/C++ Build->Settings-> Build Steps".

In this example, add: arm-none-eabi-objcopy -O binary "SysTick_TimeBase.elf" "SysTick_TimeBase.bin"; arm-none-eabi-objdump -D "SysTick_TimeBase.elf" > "SysTick_TimeBase.dump"

Figure 18 Configuration - Generate bin File



Configuration: Debug [Active]

Post-build steps Command:

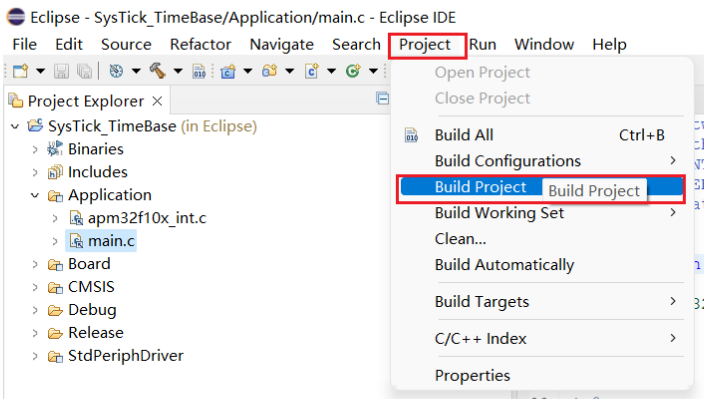
```
arm-none-eabi-objcopy -O binary "SysTick_TimeBase.elf" "SysTick_TimeBase.bin"; arm-none-eabi-objdump -D "SysTick_TimeBase.elf" > "SysTick_TimeBase.dump"
```

3.4 Compile Project

Click Project and select Build Project to compile the current project.

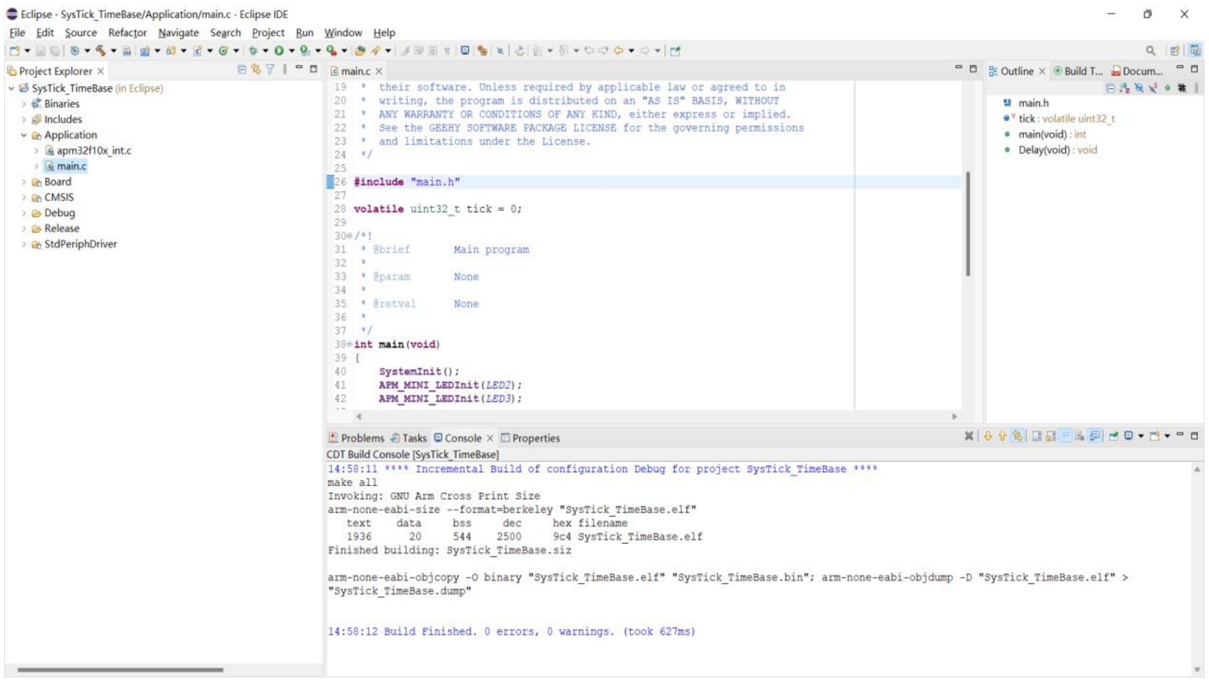
Note: Using Build Project will only compile the current project, while using Build All will compile all projects in the current workspace.

Figure 19 Compile a Project



Note: Before compiling, be sure to save the current project. Otherwise, the compiled project may be the last saved project, rather than the current modified project. To ensure the correctness of the compilation, it is necessary to perform project clean after the modifications, and then compile. After compilation, corresponding elf, hex, and bin files will be generated.

Figure 20 Compilation Results



```
 * their software. Unless required by applicable law or agreed to in
 * writing, the program is distributed on an "AS IS" BASIS, WITHOUT
 * ANY WARRANTY OR CONDITIONS OF ANY KIND, either express or implied.
 * See the GEEHY SOFTWARE PACKAGE LICENSE for the governing permissions
 * and limitations under the License.
 */

#include "main.h"

volatile uint32_t tick = 0;

/*!
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
 */
int main(void)
{
    SystemInit();
    APM_MINI_LEDInit(LED2);
    APM_MINI_LEDInit(LED3);
```

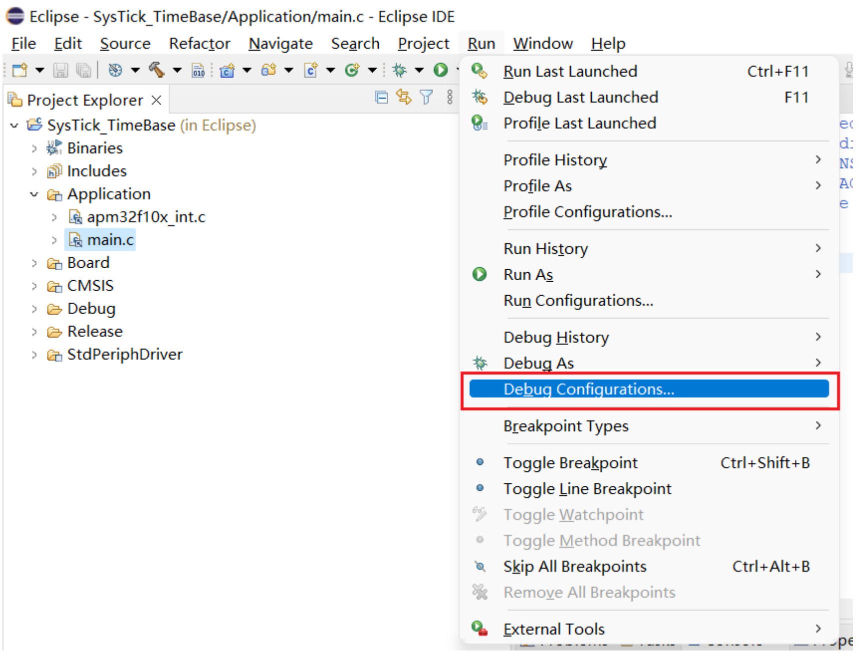
```
14:58:11 **** Incremental Build of configuration Debug for project SysTick_TimeBase ****
make all
Invoking: GNU Arm Cross Print Size
arm-none-eabi-size --format=berkeley "SysTick_TimeBase.elf"
   text    data     bss     dec     hex filename
   1936      20     544    2500     9c4 SysTick_TimeBase.elf
Finished building: SysTick_TimeBase.siz

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


14:58:12 Build Finished. 0 errors, 0 warnings. (took 627ms)
```

3.5 Download and debug the project with J-Link

3.5.1 Open Debug configuration interface

In the project, click Run in the menu bar and click Debug Configurations in the pop-up option box to enter the Debug configuration interface.

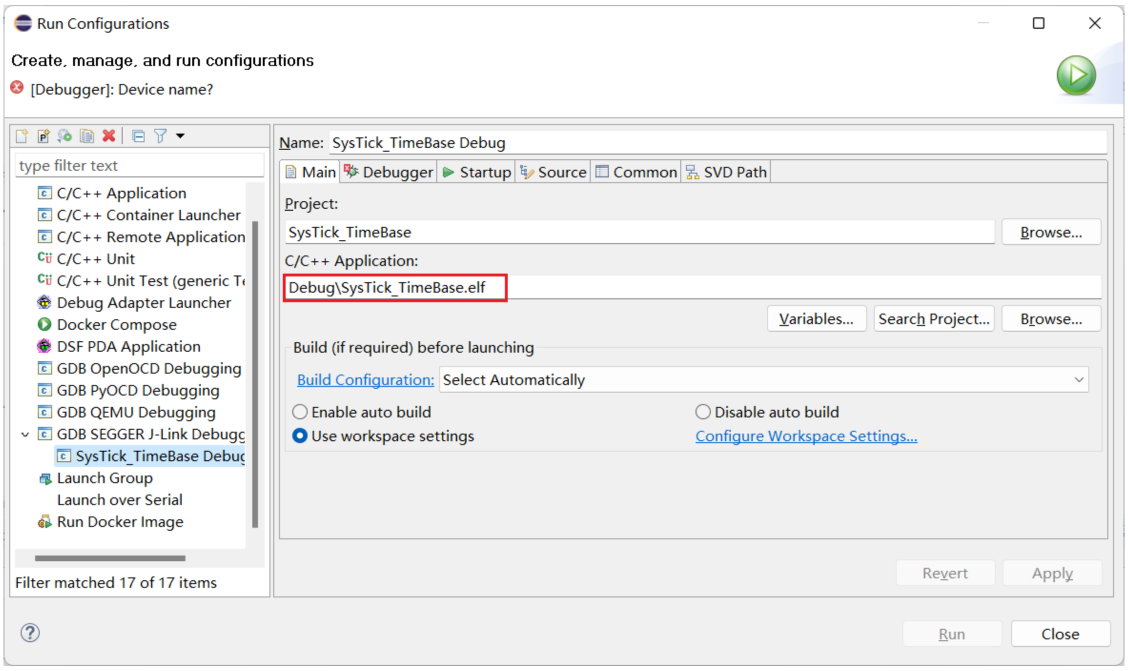
Figure 21 Enter Debug Configuration Interface



Use GDB Server and GDB Client as J-Link configuration option tools. In this example, GDB Server is J-Link GDBServerCL, and GDB Client is GDB tool in GCC tool chain. To create a new set of J-Link configuration options, double-click GDB SEGGER J-Link Debugging.

3.5.2 Configure Main tab

Figure 22 J-Link Main Tab



Run Configurations

Create, manage, and run configurations

[Debugger]: Device name?

Name: SysTick_TimeBase Debug

Project: SysTick_TimeBase

C/C++ Application: Debug\SysTick_TimeBase.elf

Build (if required) before launching

Build Configuration: Select Automatically

Use workspace settings

Filter matched 17 of 17 items

After a new set of J-Link configuration options is created, the main tab generally will automatically add the elf file of the current project by default. If not, click Browse on the right, find the elf file generated by the project compilation and manually add it.

Note: After different chip models are compiled, an elf file will be generated. If there is the need to compile multiple chip models, a new set of Debug configuration can be created for different models of the chip.

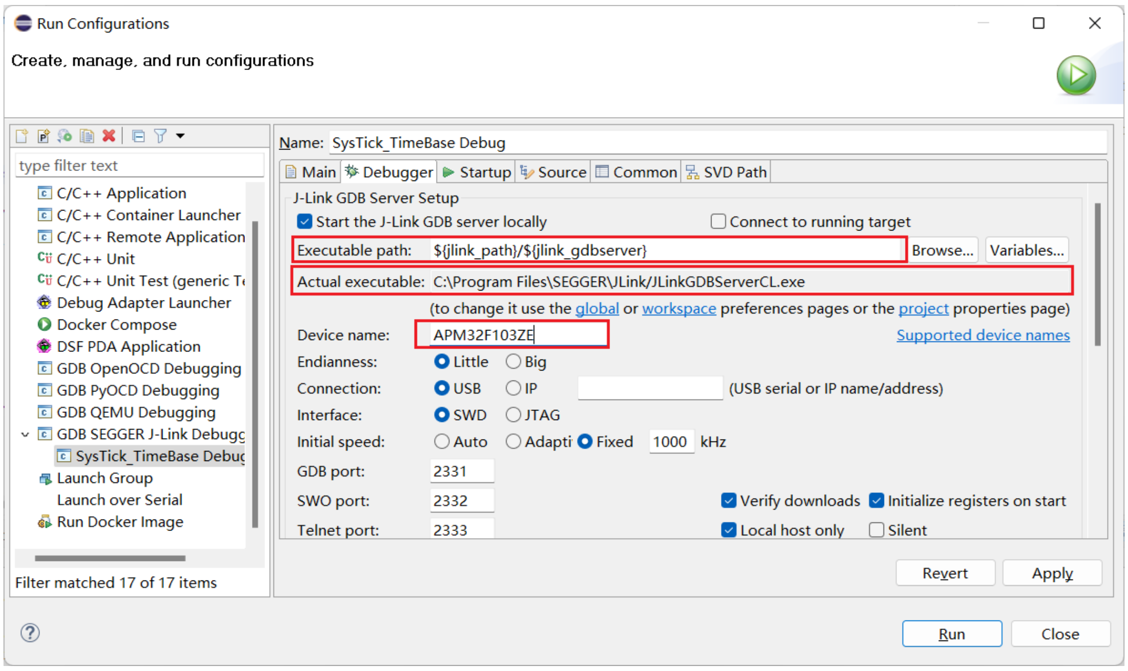
3.5.3 Configure Debugger tab

In Debugger tab, the corresponding chip model needs to be filled out in the Device name, and it is APM32F103ZE in this example.

When building an Eclipse environment, if J-Link path has been correctly configured, it will be automatically identified here. If the J-Link path configuration fails, the Browse of the Executable path can be clicked to select the path of J-Link GDBServerCL.

Note: The configured J-Link driver must support the chip model filled in.

Figure 23 J-Link Debugger Tab


Run Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Container Launcher
- C/C++ Remote Application
- C/C++ Unit
- C/C++ Unit Test (generic T
- Debug Adapter Launcher
- Docker Compose
- DSF PDA Application
- GDB OpenOCD Debugging
- GDB PyOCD Debugging
- GDB QEMU Debugging
- GDB SEGGER J-Link Debugg
  - SysTick_TimeBase Debuc
- Launch Group
- Launch over Serial
- Run Docker Image

Filter matched 17 of 17 items

Name: SysTick_TimeBase Debug

Main | Debugger | Startup | Source | Common | SVD Path

J-Link GDB Server Setup

[✓] Start the J-Link GDB server locally [ ] Connect to running target

Executable path: ${jlink_path}/${jlink_gdbserver} Browse... Variables...

Actual executable: C:\Program Files\SEGGER\JLink/JLinkGDBServerCL.exe

(to change it use the global or workspace preferences pages or the project properties page)

Device name: APM32F103ZE Supported device names

Endianness: [✓ Little] Big

Connection: [✓ USB] IP (USB serial or IP name/address)

Interface: [✓ SWD] JTAG

Initial speed: Auto Adapti [✓ Fixed] 1000 kHz

GDB port: 2331

SWO port: 2332 [✓] Verify downloads [✓] Initialize registers on start

Telnet port: 2333 [✓] Local host only [ ] Silent

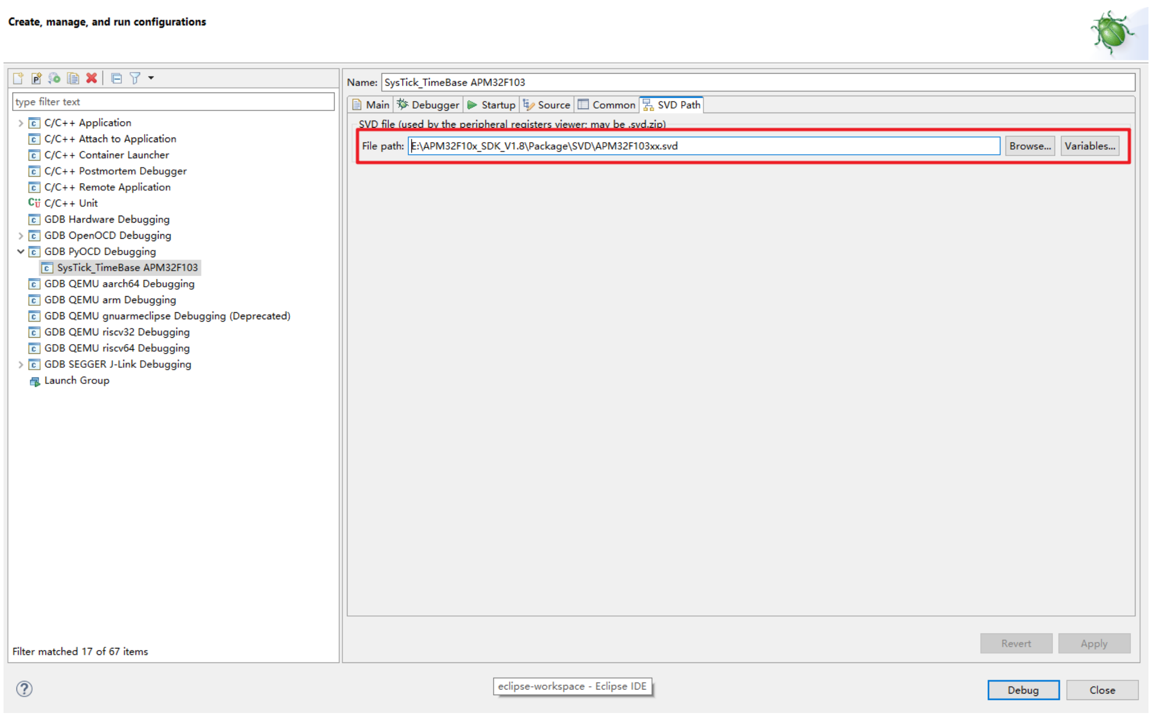
Revert Apply

Run Close

3.5.4 Configure SVD Path tab

In the SVD Path tab, select the SVD file of the target chip. In this example, the SVD file is APM32F103xx.svd, and the default path is: APM32F103_SDK\Packages\SVD

Figure 24 J-Link SVD Path Tab


Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Container Launcher
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- C/C++ Unit
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB PyOCD Debugging
  - SysTick_TimeBase APM32F103
- GDB QEMU aarch64 Debugging
- GDB QEMU arm Debugging
- GDB QEMU gnuarmeclipse Debugging (Deprecated)
- GDB QEMU riscv32 Debugging
- GDB QEMU riscv64 Debugging
- GDB SEGGER J-Link Debugging
- Launch Group

Filter matched 17 of 67 items

Name: SysTick_TimeBase APM32F103

Main | Debugger | Startup | Source | Common | SVD Path

SVD file (used by the peripheral registers viewer; may be .svd.zip)

File path: E:\APM32F10x_SDK_V1.8\Package\SVD\APM32F103xx.svd Browse... Variables...

Revert Apply

eclipse-workspace - Eclipse IDE

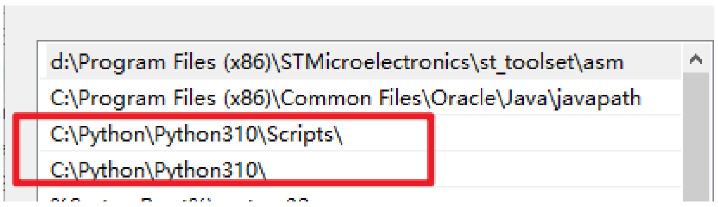
Debug Close

3.6 Download and debug the project with Geehy-Link

3.6.1 Configure Pyocd environment

Install python. The official website download link is Welcome to Python.org. Set the system environment variables after downloading.

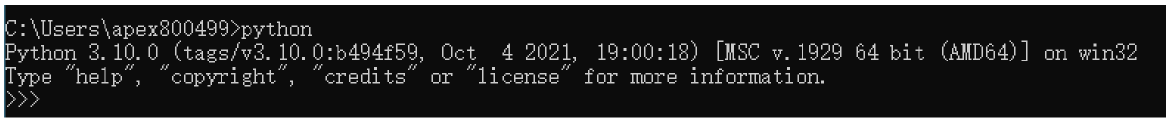
Figure 25 Configure System Environment Variables



```
d:\Program Files (x86)\STMicroelectronics\st_toolset\asm
C:\Program Files (x86)\Common Files\Oracle\Java\javapath
C:\Python\Python310\Scripts\
C:\Python\Python310\
```

Open Windows cmd and enter python. If the following information appears, it indicates that python has been successfully installed.

Figure 26 python Is Installed Successfully



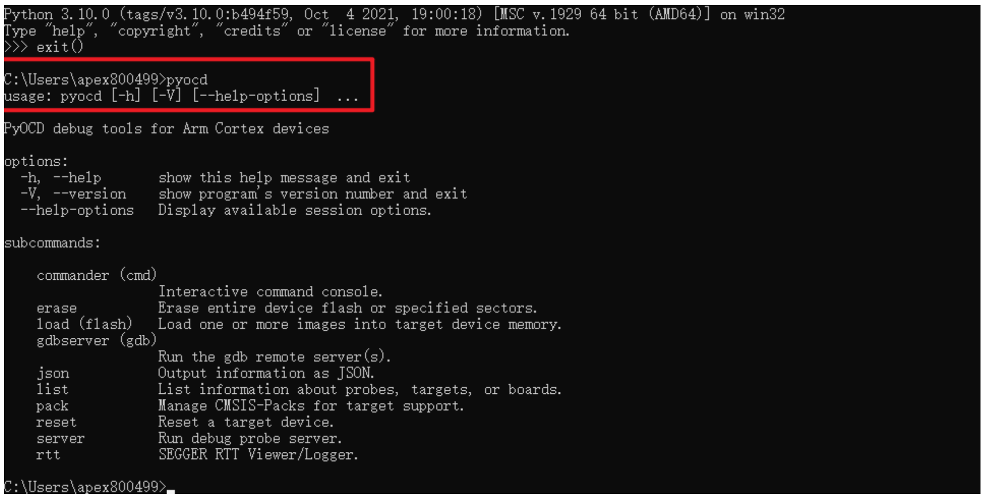
```
C:\Users\apex800499>python
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python is installed successfully. Install pyocd and usb driver library.

1. Exit python and enter exit() in cmd
2. Install pyocd and enter pipinstall - U pyocd in cmd
3. Install libsub library and enter pip install - U libusb in cmd

Enter pyocd in Cmd. If the following information appears, it indicates successful installation.

Figure 27 pyocd Is Installed Successfully

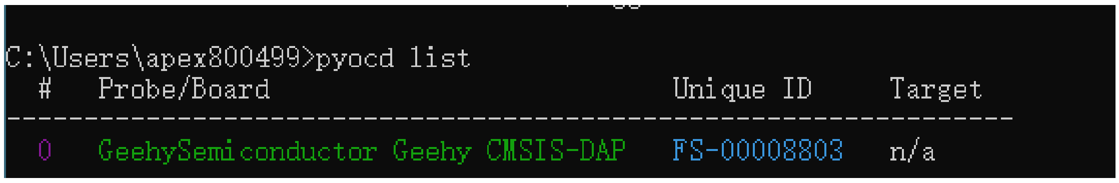


```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\apex800499>pyocd
usage: pyocd [-h] [-V] [--help-options]  ...

PyOCD debug tools for Arm Cortex devices

options:
  -h, --help        show this help message and exit
  -V, --version     show program's version number and exit
  --help-options    Display available session options.

subcommands:

    commander (cmd)
                    Interactive command console.
    erase           Erase entire device flash or specified sectors.
    load (flash)    Load one or more images into target device memory.
    gdbserver (gdb)
                    Run the gdb remote server(s).
    json            Output information as JSON.
    list            List information about probes, targets, or boards.
    pack            Manage CMSIS-Packs for target support.
    reset           Reset a target device.
    server          Run debug probe server.
    rtt             SEGGER RTT Viewer/Logger.

C:\Users\apex800499>
```

3.6.2 Install pack

Connect to Geehy-link emulator, and enter pyocd list in cmd. If the following content appears, it indicates successful connection.

Figure 28 Connect Geehy-link Emulator

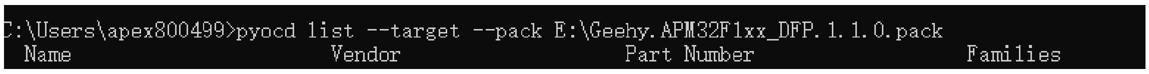


```
C:\Users\apex800499>pyocd list
  #   Probe/Board                          Unique ID     Target
------------------------------------------------------------------
  0   GeehySemiconductor Geehy CMSIS-DAP   FS-00008803   n/a
```

Download the pack for the corresponding apm32f1 series from the official website of keil (https://www.keil.arm.com/devices) or the official website of Geehy (https://geehy.com/MCU).

Enter pyocd list --target —pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack in Cmd, and view all chip models supported by apm32f1 series pack.

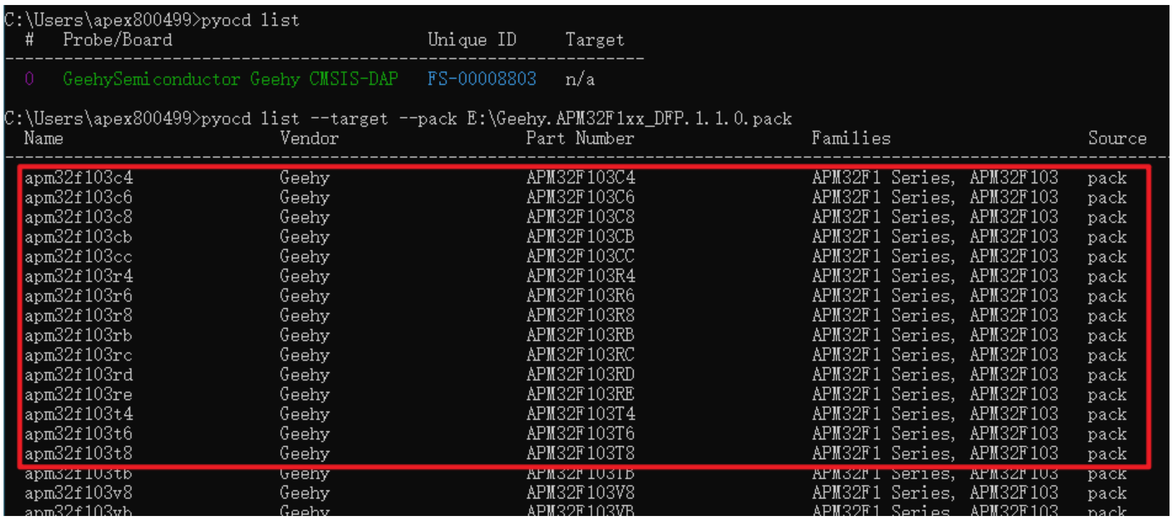
Figure 29 View apm32f1 Series Pack



```
C:\Users\apex800499>pyocd list --target --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack
  Name                    Vendor                 Part Number                Families
```

When the command is successfully executed, the following information will appear, indicating that the information is successfully obtained.

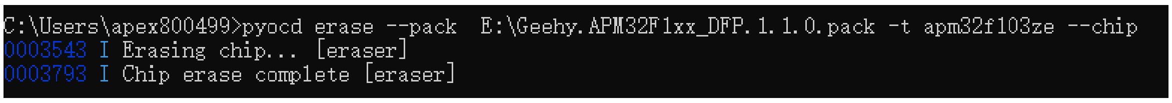
Figure 30 apm32f103 Chip Model



```
C:\Users\apex800499>pyocd list
  #   Probe/Board                         Unique ID     Target
-------------------------------------------------------------------
  0   GeehySemiconductor Geehy CMSIS-DAP   FS-00008803   n/a

C:\Users\apex800499>pyocd list --target --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack
  Name                Vendor              Part Number            Families                     Source
---------------------------------------------------------------------------------------------------------
  apm32f103c4         Geehy               APM32F103C4            APM32F1 Series, APM32F103    pack
  apm32f103c6         Geehy               APM32F103C6            APM32F1 Series, APM32F103    pack
  apm32f103c8         Geehy               APM32F103C8            APM32F1 Series, APM32F103    pack
  apm32f103cb         Geehy               APM32F103CB            APM32F1 Series, APM32F103    pack
  apm32f103cc         Geehy               APM32F103CC            APM32F1 Series, APM32F103    pack
  apm32f103r4         Geehy               APM32F103R4            APM32F1 Series, APM32F103    pack
  apm32f103r6         Geehy               APM32F103R6            APM32F1 Series, APM32F103    pack
  apm32f103r8         Geehy               APM32F103R8            APM32F1 Series, APM32F103    pack
  apm32f103rb         Geehy               APM32F103RB            APM32F1 Series, APM32F103    pack
  apm32f103rc         Geehy               APM32F103RC            APM32F1 Series, APM32F103    pack
  apm32f103rd         Geehy               APM32F103RD            APM32F1 Series, APM32F103    pack
  apm32f103re         Geehy               APM32F103RE            APM32F1 Series, APM32F103    pack
  apm32f103t4         Geehy               APM32F103T4            APM32F1 Series, APM32F103    pack
  apm32f103t6         Geehy               APM32F103T6            APM32F1 Series, APM32F103    pack
  apm32f103t8         Geehy               APM32F103T8            APM32F1 Series, APM32F103    pack
  apm32f103tb         Geehy               APM32F103TB            APM32F1 Series, APM32F103    pack
  apm32f103v8         Geehy               APM32F103V8            APM32F1 Series, APM32F103    pack
  apm32f103vb         Geehy               APM32F103VB            APM32F1 Series, APM32F103    pack
```

After the emulator is connected to the development board, enter pyocd erase --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack –t apm32f103ze --chip in Cmd to erase the chip.

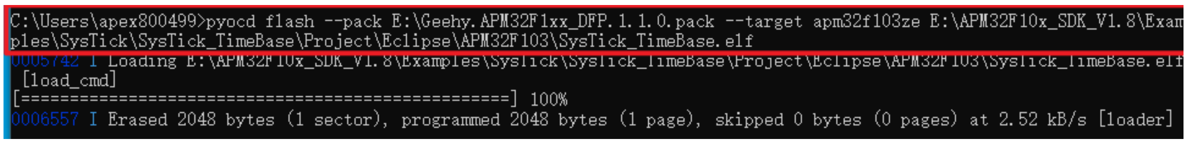
Figure 31 apm32f103ze Chip Erase



```
C:\Users\apex800499>pyocd erase --pack  E:\Geehy.APM32F1xx_DFP.1.1.0.pack -t apm32f103ze --chip
0003543 I Erasing chip... [eraser]
0003793 I Chip erase complete [eraser]
```

After the chip is successfully erased, enter pyocd flash --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack --target apm32f103ze + corresponding hex (or elf, bin) file in cmd to conduct code burning for the chip.

Figure 32 apm32f103ze Chip Burning



```
C:\Users\apex800499>pyocd flash --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack --target apm32f103ze E:\APM32F10x_SDK_V1.8\Exam
ples\SysTick\SysTick_TimeBase\Project\Eclipse\APM32F103\SysTick_TimeBase.elf
0005742 I Loading E:\APM32F10x_SDK_V1.8\Examples\SysTick\SysTick_TimeBase\Project\Eclipse\APM32F103\SysTick_TimeBase.elf
 [load_cmd]
[==================================================] 100%
0006557 I Erased 2048 bytes (1 sector), programmed 2048 bytes (1 page), skipped 0 bytes (0 pages) at 2.52 kB/s [loader]
```

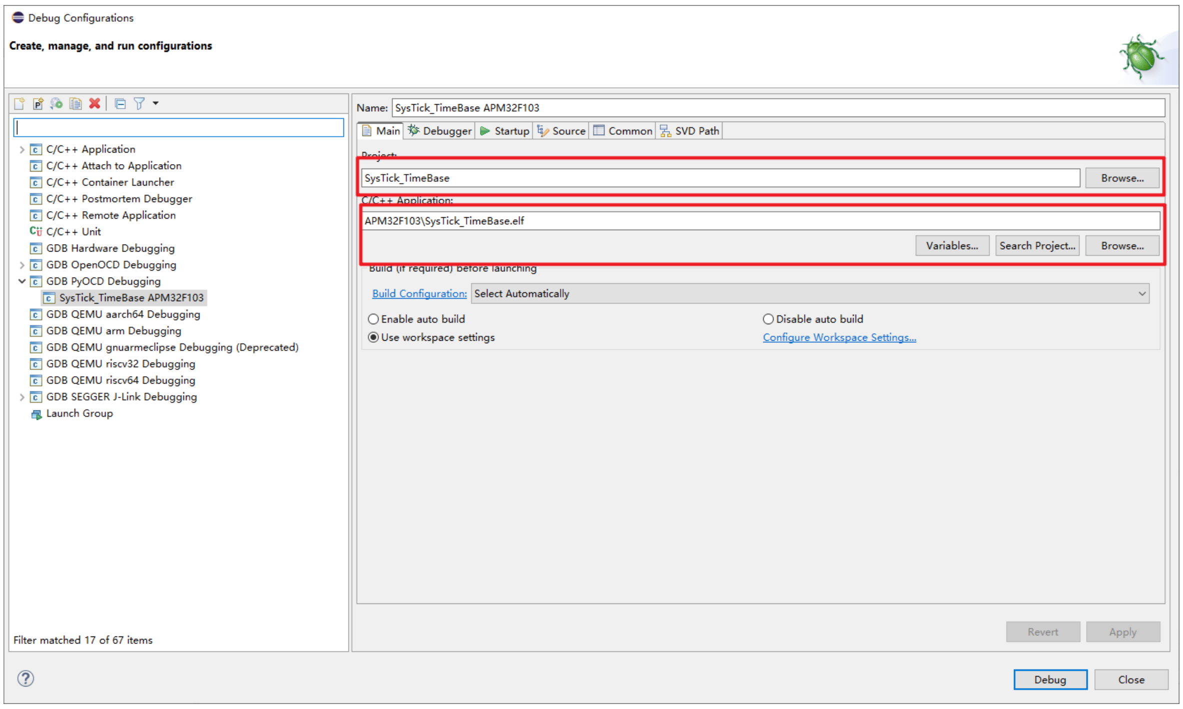
3.6.3 Open Debug configuration interface

In the menu bar, click Run and select Debug Configurations in the pop-up selection box to enter the Debug configuration interface.

Use PyOCD and GDB Client as Geehy-Link configuration option tools. In this example, GDB Server is PyOCD, and GDB Client is GDB tool in GCC tool chain. To create a new set of Geehy-Link configuration options, double-click GDB PyOCD Debugging.

3.6.4 Configure Main tab

Figure 33 Geehy-Link Main Tab



Debug Configurations

Create, manage, and run configurations

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Container Launcher
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- C/C++ Unit
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB PyOCD Debugging
  - SysTick_TimeBase APM32F103
- GDB QEMU aarch64 Debugging
- GDB QEMU arm Debugging
- GDB QEMU gnuarmeclipse Debugging (Deprecated)
- GDB QEMU riscv32 Debugging
- GDB QEMU riscv64 Debugging
- GDB SEGGER J-Link Debugging
- Launch Group

Filter matched 17 of 67 items

Name: SysTick_TimeBase APM32F103

Main | Debugger | Startup | Source | Common | SVD Path

Project: SysTick_TimeBase

C/C++ Application: APM32F103\SysTick_TimeBase.elf

Build (if required) before launching

Build Configuration: Select Automatically

Enable auto build
[✓ Use workspace settings]
Disable auto build
Configure Workspace Settings...

After a new set of Geehy-Link configuration options is created, the main tab generally will automatically add the elf file of the current project by default. If not, click Browse on the right, find the elf file generated by the project compilation and manually add it.

Note: After different chip models are compiled, an elf file will be generated. If there is the need to compile multiple chip models, a new set of Debug configuration can be created for different models of the chip.

3.6.5 Configure Debugger tab

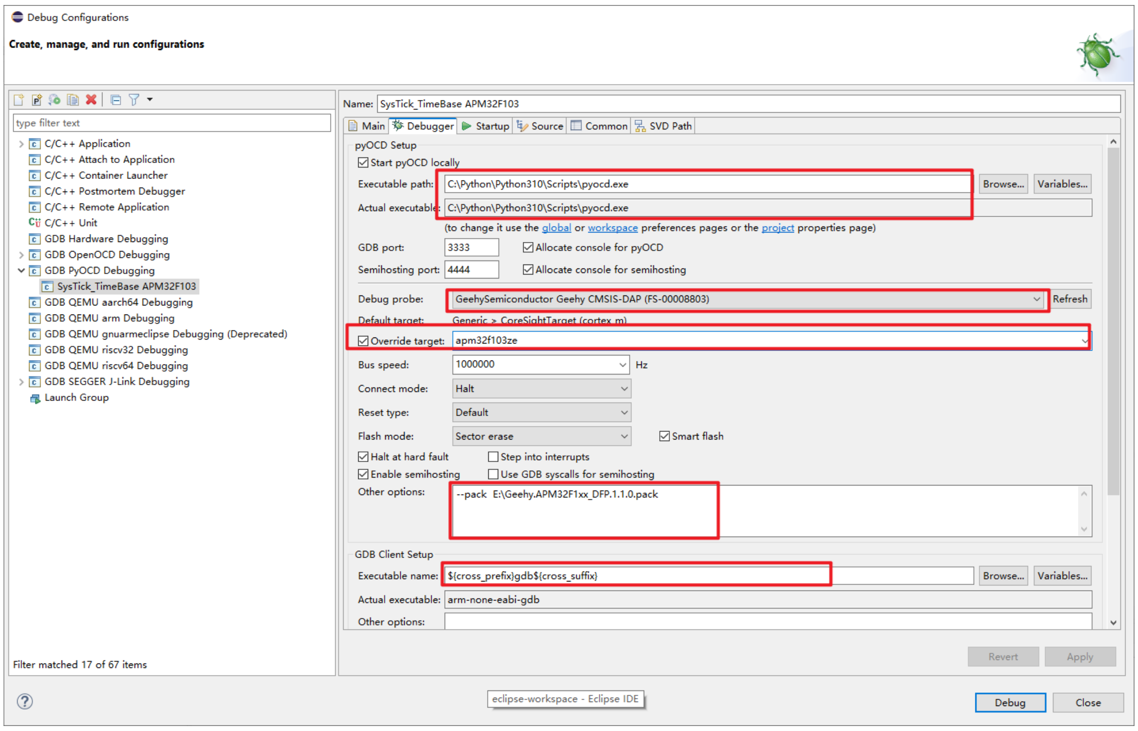
When building an Eclipse environment, if PyOCD path has been correctly configured, it will be automatically identified here. If the PyOCD path configuration fails, the Browse of the Executable path can be clicked to select the path of PyOCD. In the Debug probe bar, select the used CMSIS-DAP emulator.

Check override target, select the corresponding chip model, and enter the following command in Cmd to view the supported models. Here, apm32f103ze is used.

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

Add the pack path in other options, -- pack E:\ Geehy.APM32F1xx_DFP.1.1.0.pack.

Figure 34 Geehy-Link Debugger Tab


Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Container Launcher
C/C++ Postmortem Debugger
C/C++ Remote Application
C/C++ Unit
GDB Hardware Debugging
GDB OpenOCD Debugging
GDB PyOCD Debugging
SysTick_TimeBase APM32F103
GDB QEMU aarch64 Debugging
GDB QEMU arm Debugging
GDB QEMU gnuarmeclipse Debugging (Deprecated)
GDB QEMU riscv32 Debugging
GDB QEMU riscv64 Debugging
GDB SEGGER J-Link Debugging
Launch Group
Filter matched 17 of 67 items
Main
Debugger
Startup
Source
Common
SVD Path
Browse...
Variables...
Refresh
Revert
Apply
eclipse-workspace - Eclipse IDE
Debug
Close

Name: SysTick_TimeBase APM32F103

pyOCD Setup

☑ Start pyOCD locally

Executable path: C:\Python\Python310\Scripts\pyocd.exe

Actual executable: C:\Python\Python310\Scripts\pyocd.exe

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 3333 ☑ Allocate console for pyOCD

Semihosting port: 4444 ☑ Allocate console for semihosting

Debug probe: GeehySemiconductor Geehy CMSIS-DAP (FS-00008803)

Default target: Generic > CoreSightTarget (cortex_m)

☑ Override target: apm32f103ze

Bus speed: 1000000 Hz

Connect mode: Halt

Reset type: Default

Flash mode: Sector erase ☑ Smart flash

☑ Halt at hard fault ☐ Step into interrupts

☑ Enable semihosting ☐ Use GDB syscalls for semihosting

Other options: --pack E:\Geehy.APM32F1xx_DFP.1.1.0.pack

GDB Client Setup

Executable name: ${cross_prefix}gdb${cross_suffix}

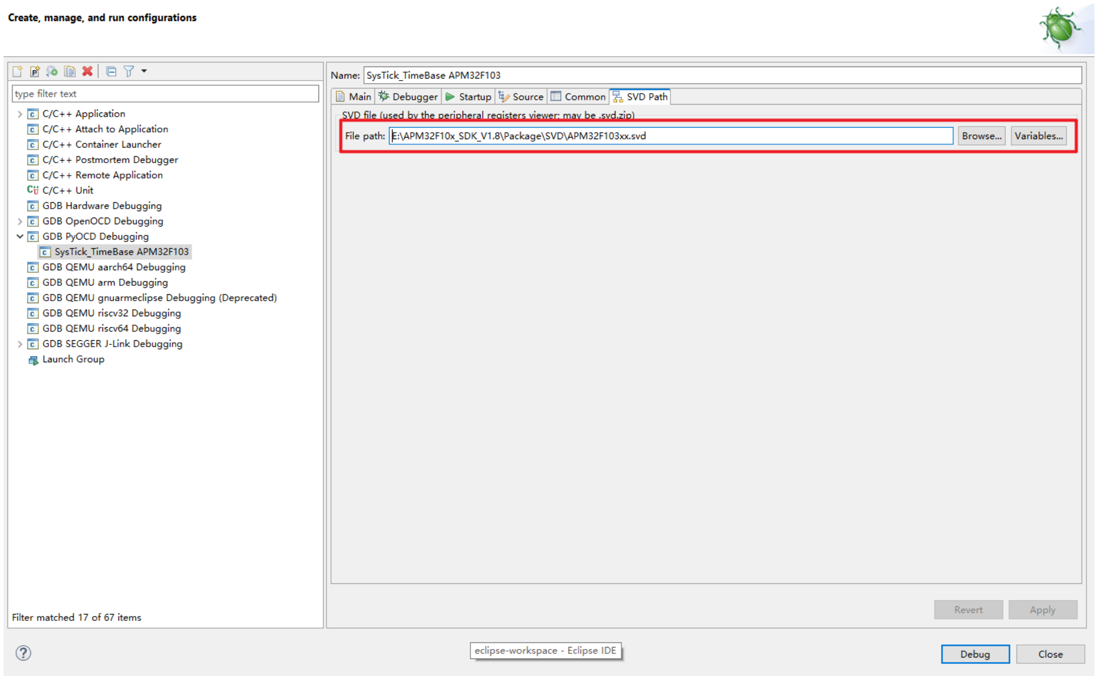
Actual executable: arm-none-eabi-gdb

Other options:

3.6.6 Configure SVD Path tab

In the SVD Path tab, select the SVD file required for the chip. In this example, APM32F103xx.svd is used, and the default file path of SVD is APM32F10x_SDK\Package\SVD.

Figure 35 Geehy-Link SVD Path Tab


Create, manage, and run configurations
type filter text
C/C++ Application
C/C++ Attach to Application
C/C++ Container Launcher
C/C++ Postmortem Debugger
C/C++ Remote Application
C/C++ Unit
GDB Hardware Debugging
GDB OpenOCD Debugging
GDB PyOCD Debugging
SysTick_TimeBase APM32F103
GDB QEMU aarch64 Debugging
GDB QEMU arm Debugging
GDB QEMU gnuarmeclipse Debugging (Deprecated)
GDB QEMU riscv32 Debugging
GDB QEMU riscv64 Debugging
GDB SEGGER J-Link Debugging
Launch Group
Filter matched 17 of 67 items
Main
Debugger
Startup
Source
Common
SVD Path
Browse...
Variables...
Revert
Apply
eclipse-workspace - Eclipse IDE
Debug
Close

Name: SysTick_TimeBase APM32F103

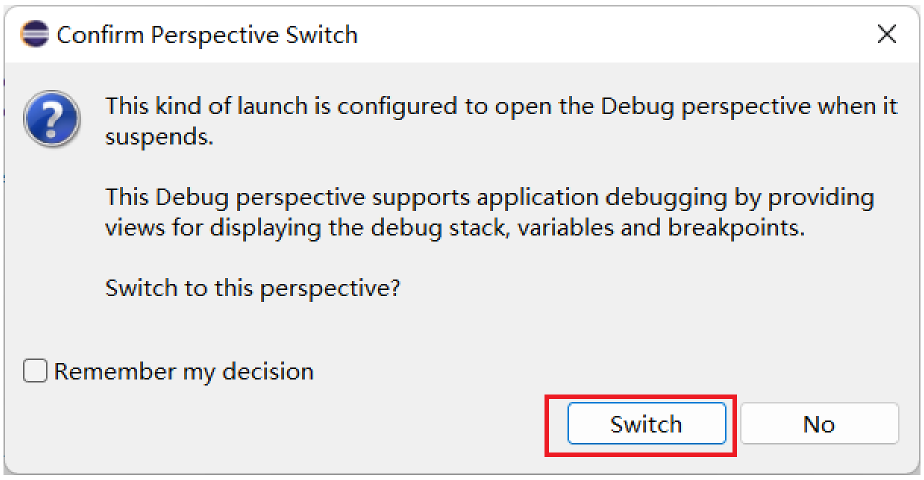
SVD file (used by the peripheral registers viewer; may be .svd.zip)

File path: E:\APM32F10x_SDK_V1.8\Package\SVD\APM32F103xx.svd

3.7 Debug interface

If Debug Configurations are completed, click Debug to debug. After the following figure appears, click Switch to enter the Debug view.

Figure 36 Enter Debug View



Confirm Perspective Switch

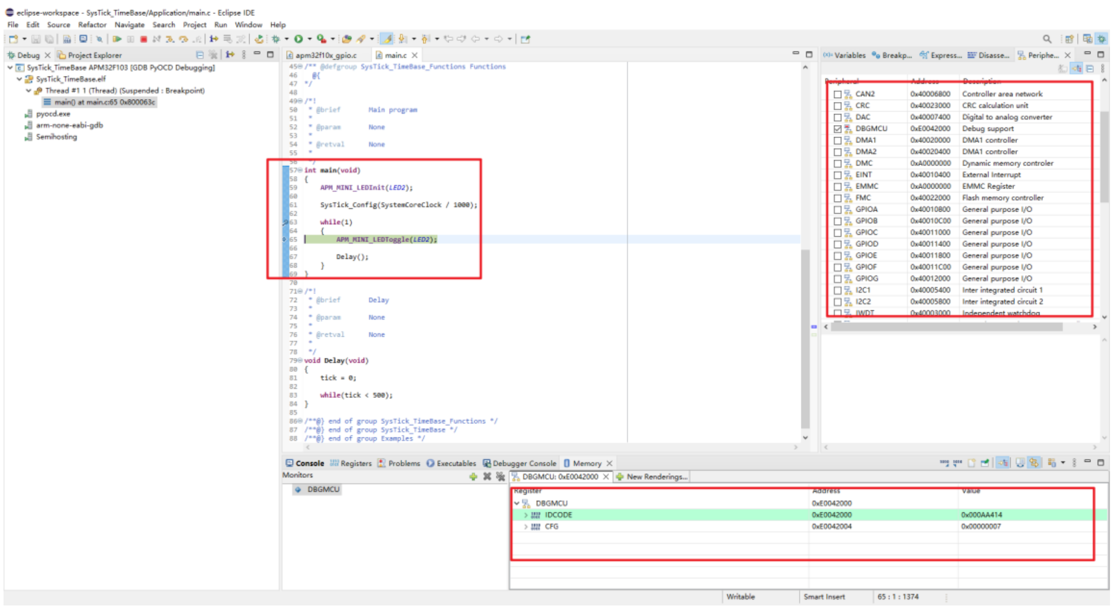
This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective supports application debugging by providing views for displaying the debug stack, variables and breakpoints.

Switch to this perspective?

☐ Remember my decision

Figure 37 Debug View



eclipse-workspace - SysTick_TimeBase/Application/main.c - Eclipse IDE

```
SysTick_TimeBase APM32F103 [GDB PyOCD Debugging]
  SysTick_TimeBase.elf
    Thread #1 1 (Thread) (Suspended : Breakpoint)
      main() at main.c:65 0x800063c
  pyocd.exe
  arm-none-eabi-gdb
  Semihosting
```

```
/** @defgroup SysTick_TimeBase_Functions Functions
  @{
*/

/*!
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
 *
 */
int main(void)
{
    APM_MINI_LEDInit(LED2);

    SysTick_Config(SystemCoreClock / 1000);

    while(1)
    {
        APM_MINI_LEDToggle(LED2);

        Delay();
    }
}

/*!
 * @brief       Delay
 *
 * @param       None
 *
 * @retval      None
 *
 */
void Delay(void)
{
    tick = 0;

    while(tick < 500);
}

/**@} end of group SysTick_TimeBase_Functions */
/**@} end of group SysTick_TimeBase */
/**@} end of group Examples */
```

Peripheral	Address	Description
☐ CAN2	0x40006800	Controller area network
☐ CRC	0x40023000	CRC calculation unit
☐ DAC	0x40007400	Digital to analog converter
☑ DBGMCU	0xE0042000	Debug support
☐ DMA1	0x40020000	DMA1 controller
☐ DMA2	0x40020400	DMA1 controller
☐ DMC	0xA0000000	Dynamic memory controler
☐ EINT	0x40010400	External Interrupt
☐ EMMC	0xA0000000	EMMC Register
☐ FMC	0x40022000	Flash memory controller
☐ GPIOA	0x40010800	General purpose I/O
☐ GPIOB	0x40010C00	General purpose I/O
☐ GPIOC	0x40011000	General purpose I/O
☐ GPIOD	0x40011400	General purpose I/O
☐ GPIOE	0x40011800	General purpose I/O
☐ GPIOF	0x40011C00	General purpose I/O
☐ GPIOG	0x40012000	General purpose I/O
☐ I2C1	0x40005400	Inter integrated circuit 1
☐ I2C2	0x40005800	Inter integrated circuit 2
☐ IWDT	0x40003000	Independent watchdog

Monitors: DBGMCU — DBGMCU: 0xE0042000

Register	Address	Value
DBGMCU	0xE0042000	
IDCODE	0xE0042000	0x000AA414
CFG	0xE0042004	0x00000007

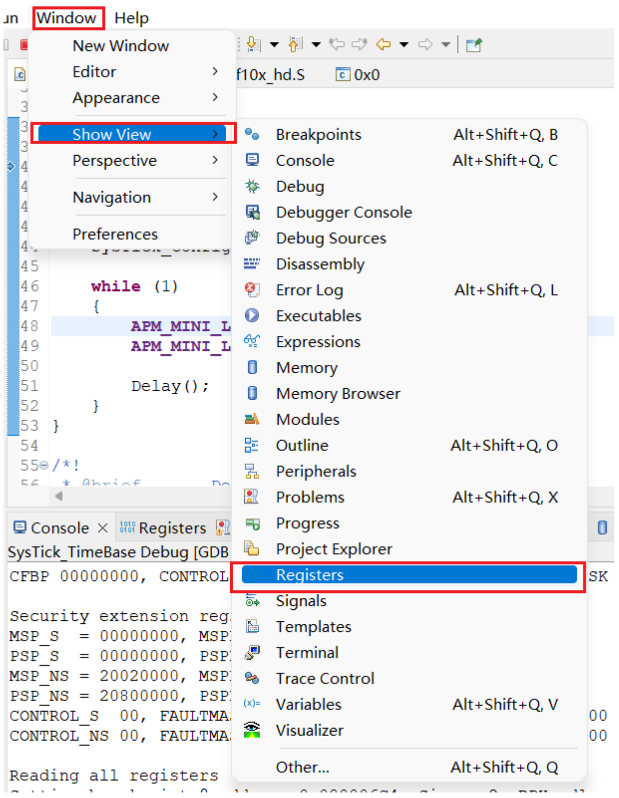
3.7.1 Introduction to toolbars

: resume to full-speed operation; : suspend; : terminate; : step into; : step over; : step out; : reset.

3.7.2 Registers window

Select the Window>Show view>Registers option in the menu bar to view the values of general registers.

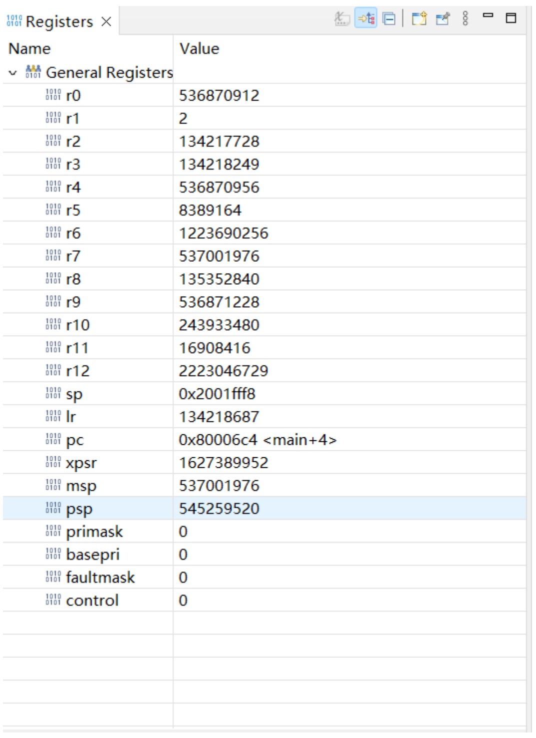
Figure 38 Registers Option



```
    while (1)
    {
        APM_MINI_L
        APM_MINI_L

        Delay();
    }
}

/*!
```

```
SysTick_TimeBase Debug [GDB
CFBP 00000000, CONTROL

Security extension reg
MSP_S  = 00000000, MSP
PSP_S  = 00000000, PSP
MSP_NS = 20020000, MSP
PSP_NS = 20800000, PSP
CONTROL_S  00, FAULTMA
CONTROL_NS 00, FAULTMA

Reading all registers
```

Figure 39 Registers Window



Registers

Name	Value
General Registers	
r0	536870912
r1	2
r2	134217728
r3	134218249
r4	536870956
r5	8389164
r6	1223690256
r7	537001976
r8	135352840
r9	536871228
r10	243933480
r11	16908416
r12	2223046729
sp	0x2001fff8
lr	134218687
pc	0x80006c4 <main+4>
xpsr	1627389952
msp	537001976
psp	545259520
primask	0
basepri	0
faultmask	0
control	0

3.7.3 Peripherals window

In the menu bar, to view the values of peripheral registers, click the Window>Show view>Peripherals option.

Figure 40 Open Peripherals window

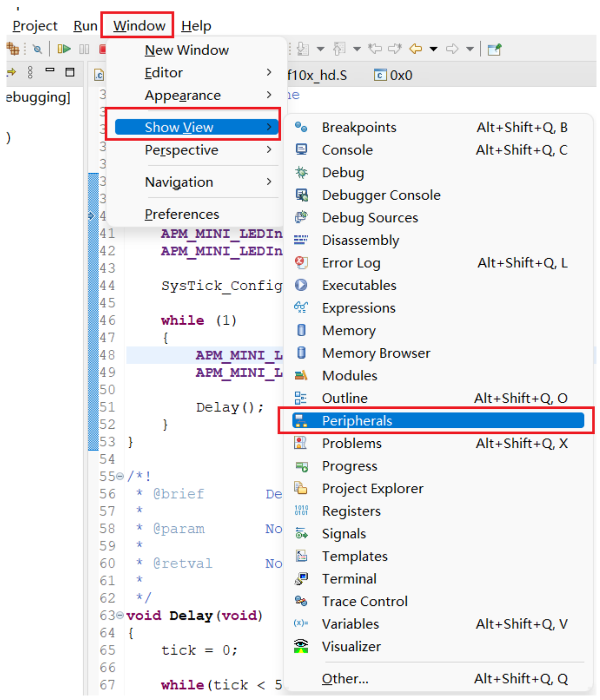
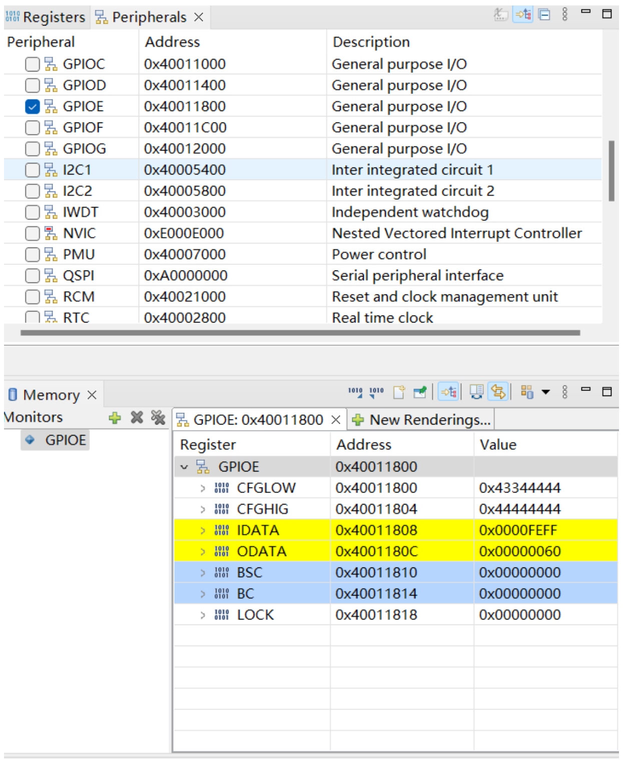


Figure 41 View Peripheral Registers



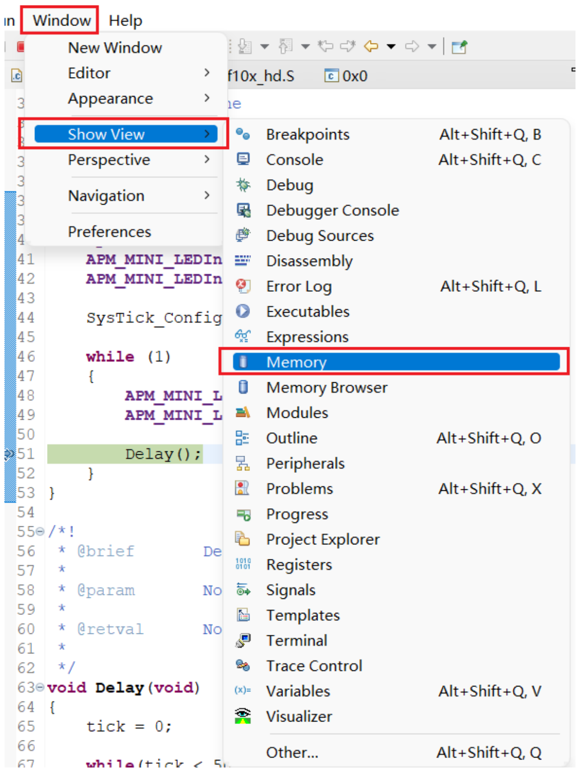
Peripheral	Address	Description
☐ GPIOC	0x40011000	General purpose I/O
☐ GPIOD	0x40011400	General purpose I/O
☑ GPIOE	0x40011800	General purpose I/O
☐ GPIOF	0x40011C00	General purpose I/O
☐ GPIOG	0x40012000	General purpose I/O
☐ I2C1	0x40005400	Inter integrated circuit 1
☐ I2C2	0x40005800	Inter integrated circuit 2
☐ IWDT	0x40003000	Independent watchdog
☐ NVIC	0xE000E000	Nested Vectored Interrupt Controller
☐ PMU	0x40007000	Power control
☐ QSPI	0xA0000000	Serial peripheral interface
☐ RCM	0x40021000	Reset and clock management unit
☐ RTC	0x40002800	Real time clock

Register	Address	Value
GPIOE	0x40011800	
CFGLOW	0x40011800	0x43344444
CFGHIG	0x40011804	0x44444444
IDATA	0x40011808	0x0000FEFF
ODATA	0x4001180C	0x00000060
BSC	0x40011810	0x00000000
BC	0x40011814	0x00000000
LOCK	0x40011818	0x00000000

3.7.4 Memory window

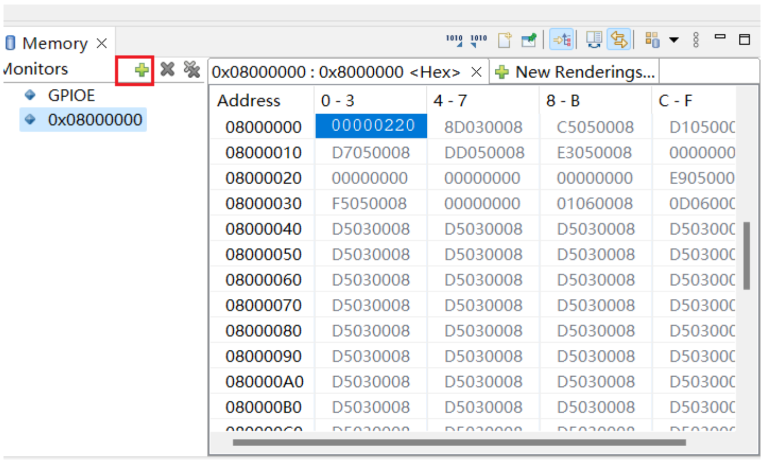
To view the corresponding memory address of the register or variable, select "Window>Show View>Memory" in the menu bar, click the "+" sign above the Memory window to view the corresponding Memory address of the current register or variable.

Figure 42 Open Memory Window



```
    APM_MINI_LEDIn
    APM_MINI_LEDIn

    SysTick_Config

    while  (1)
    {
        APM_MINI_L
        APM_MINI_L

        Delay();
    }
}

/*!
 * @brief        De
 *
 * @param        No
 *
 * @retval       No
 *
 */
void Delay(void)
{
    tick = 0;

    while(tick < 5
```

Figure 43 Memory Window

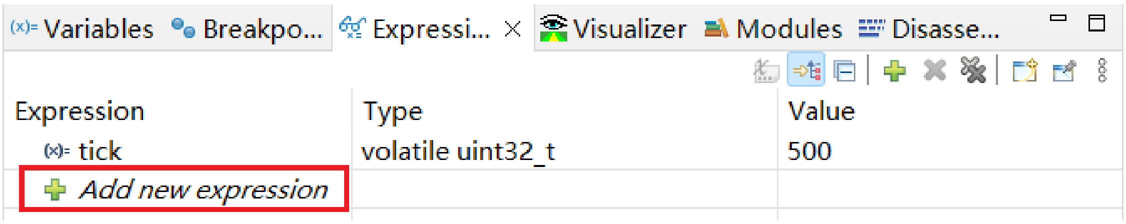


Address	0 - 3	4 - 7	8 - B	C - F
08000000	00000220	8D030008	C5050008	D105000
08000010	D7050008	DD050008	E3050008	0000000
08000020	00000000	00000000	00000000	E905000
08000030	F5050008	00000000	01060008	0D0600C
08000040	D5030008	D5030008	D5030008	D50300C
08000050	D5030008	D5030008	D5030008	D50300C
08000060	D5030008	D5030008	D5030008	D50300C
08000070	D5030008	D5030008	D5030008	D50300C
08000080	D5030008	D5030008	D5030008	D50300C
08000090	D5030008	D5030008	D5030008	D50300C
080000A0	D5030008	D5030008	D5030008	D50300C
080000B0	D5030008	D5030008	D5030008	D50300C

3.7.5 Expressions window

To view the value of the corresponding variable, select the "Window>Show View>Expressions" in the menu bar, click the "+" sign in the Expressions window to add and the variable can be viewed.

Figure 44 Expressions Window



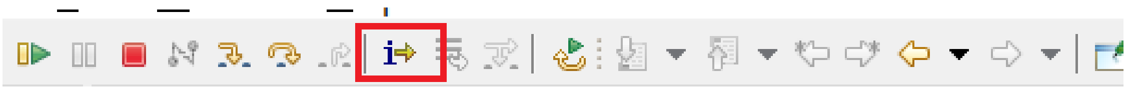
Expression	Type	Value
tick	volatile uint32_t	500
Add new expression		

Note: In Eclipse, if the program runs at full speed, the value of the corresponding variable cannot be updated in real time. The change in the value of the variable can only be viewed when the code stops running.

3.7.6 Disassembly window

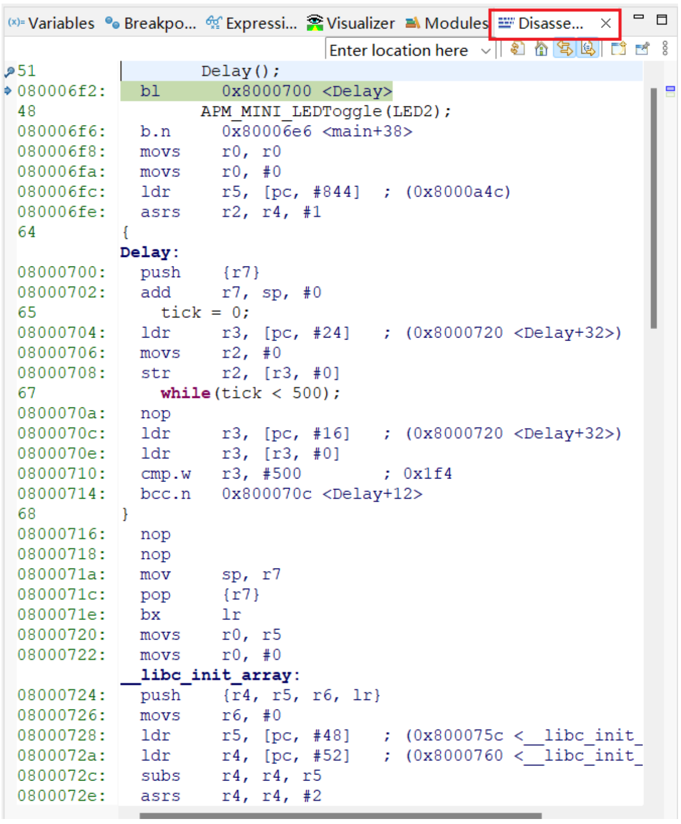
In the Debug mode, click Instruction Stepping Mode to open the disassembly window.

Figure 45 Open Disassembly Window



In the disassembly window, functions such as single-step debugging of assembly instruction can be implemented at the breakpoint.

Figure 46 Disassembly Window

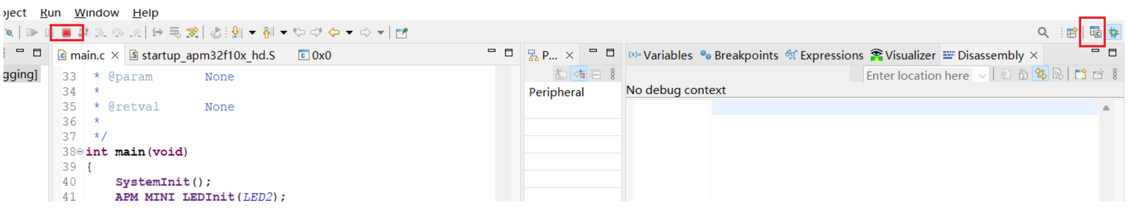


```
51                  Delay();
080006f2:   bl      0x8000700 <Delay>
48                  APM_MINI_LEDToggle(LED2);
080006f6:   b.n     0x80006e6 <main+38>
080006f8:   movs    r0, r0
080006fa:   movs    r0, #0
080006fc:   ldr     r5, [pc, #844]  ; (0x8000a4c)
080006fe:   asrs    r2, r4, #1
64        {
          Delay:
08000700:   push    {r7}
08000702:   add     r7, sp, #0
65            tick = 0;
08000704:   ldr     r3, [pc, #24]   ; (0x8000720 <Delay+32>)
08000706:   movs    r2, #0
08000708:   str     r2, [r3, #0]
67            while(tick < 500);
0800070a:   nop
0800070c:   ldr     r3, [pc, #16]   ; (0x8000720 <Delay+32>)
0800070e:   ldr     r3, [r3, #0]
08000710:   cmp.w   r3, #500        ; 0x1f4
08000714:   bcc.n   0x800070c <Delay+12>
68        }
08000716:   nop
08000718:   nop
0800071a:   mov     sp, r7
0800071c:   pop     {r7}
0800071e:   bx      lr
08000720:   movs    r0, r5
08000722:   movs    r0, #0
          __libc_init_array:
08000724:   push    {r4, r5, r6, lr}
08000726:   movs    r6, #0
08000728:   ldr     r5, [pc, #48]   ; (0x800075c <__libc_init_
0800072a:   ldr     r4, [pc, #52]   ; (0x8000760 <__libc_init_
0800072c:   subs    r4, r4, r5
0800072e:   asrs    r4, r4, #2
```

3.7.7 Exit Debug view

Click the "Stop debugging" button to exit the Debug view. To enter the project view, click the "C/C++" in the upper right corner.

Figure 47 Enter Project View



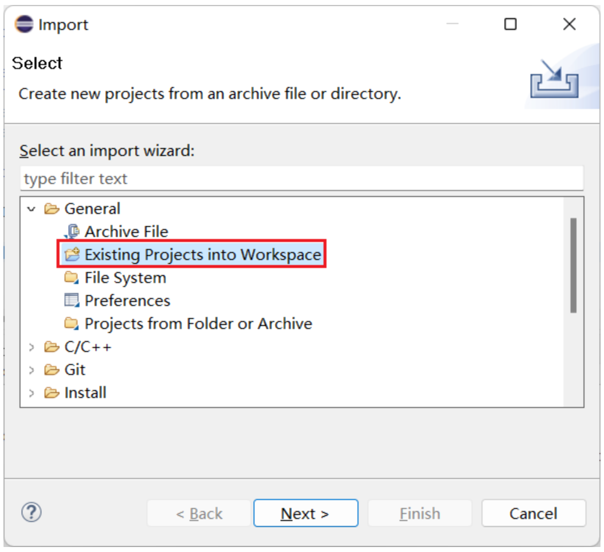
```
 * @param       None
 *
 * @retval      None
 *
 */
int main(void)
{
    SystemInit();
    APM MINI LEDInit(LED2);
```

No debug context

4 How to import existing project

Eclipse can directly import the existing project. In the project view, click File->Import. In the pop-up selection view, select General->Exiting Project into Workspace, click "Next", select the Eclipse project path, and click finish to import the existing project.

Figure 48 Import Existing Project



Import

Select

Create new projects from an archive file or directory.

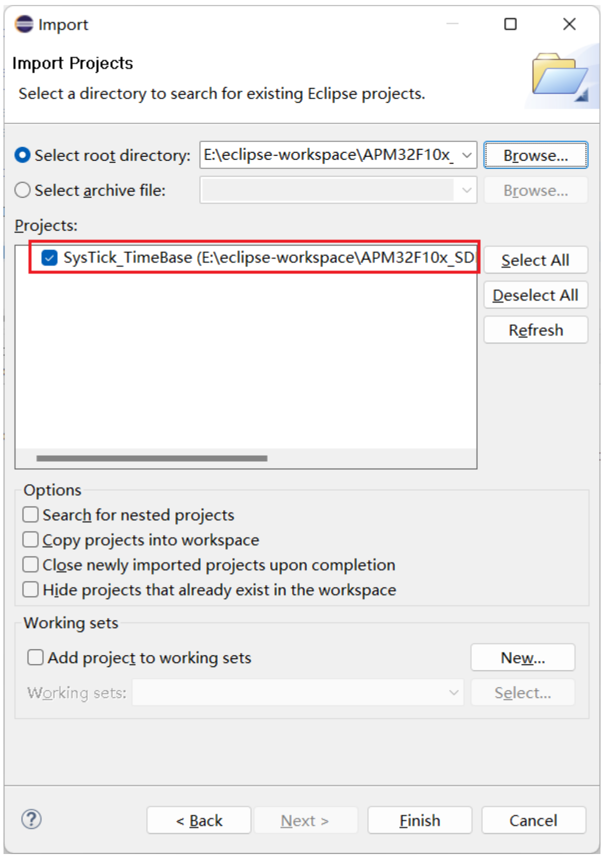
Select an import wizard:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
  - Projects from Folder or Archive
- C/C++
- Git
- Install

< Back | Next > | Finish | Cancel

Figure 49 Select Existing Project



Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory: E:\eclipse-workspace\APM32F10x_ | Browse...

Select archive file: | Browse...

Projects:

☑ SysTick_TimeBase (E:\eclipse-workspace\APM32F10x_SD

Select All | Deselect All | Refresh

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

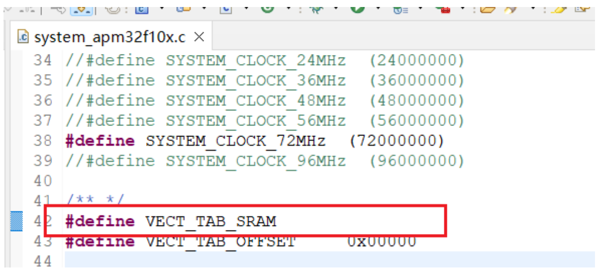
5 How to debug in RAM

Step 1: Modify the project link script. Shown below is an example of modification.

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

Step 2: Modify the interrupt vector table and relocate to SRAM. Recompile the project after finishing steps 1 and 2.

Figure 50 Change Interrupt Vector Address

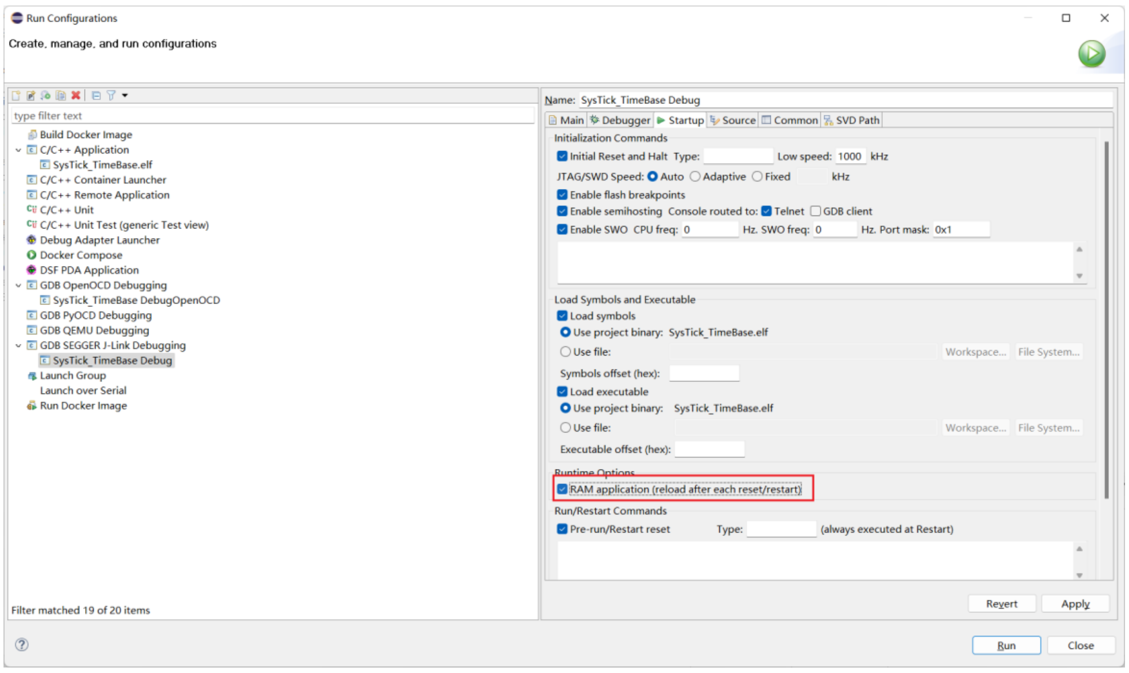


system_apm32f10x.c

```
//#define SYSTEM_CLOCK_24MHz   (24000000)
//#define SYSTEM_CLOCK_36MHz   (36000000)
//#define SYSTEM_CLOCK_48MHz   (48000000)
//#define SYSTEM_CLOCK_56MHz   (56000000)
#define SYSTEM_CLOCK_72MHz   (72000000)
//#define SYSTEM_CLOCK_96MHz   (96000000)

/** */
#define VECT_TAB_SRAM
#define VECT_TAB_OFFSET       0x00000
```

Step 3: Right-click the project, select properties, select Debug Configuration->Startup in the pop-up option box, and check RAM application.

Figure 51 Check RAM application



Run Configurations

Create, manage, and run configurations

type filter text

- Build Docker Image
- C/C++ Application
  - SysTick_TimeBase.elf
- C/C++ Container Launcher
- C/C++ Remote Application
- C/C++ Unit
- C/C++ Unit Test (generic Test view)
- Debug Adapter Launcher
- Docker Compose
- DSF PDA Application
- GDB OpenOCD Debugging
  - SysTick_TimeBase DebugOpenOCD
- GDB PyOCD Debugging
- GDB QEMU Debugging
- GDB SEGGER J-Link Debugging
  - SysTick_TimeBase Debug
- Launch Group
- Launch over Serial
- Run Docker Image

Filter matched 19 of 20 items

Name: SysTick_TimeBase Debug

Main | Debugger | Startup | Source | Common | SVD Path

Initialization Commands

- ✓ Initial Reset and Halt Type: Low speed: 1000 kHz
- JTAG/SWD Speed: [● Auto] ○ Adaptive ○ Fixed kHz
- ✓ Enable flash breakpoints
- ✓ Enable semihosting Console routed to: ✓ Telnet ☐ GDB client
- ✓ Enable SWO CPU freq: 0 Hz. SWO freq: 0 Hz. Port mask: 0x1

Load Symbols and Executable

- ✓ Load symbols
- [● Use project binary: SysTick_TimeBase.elf]
- ○ Use file: Workspace... File System...
- Symbols offset (hex):
- ✓ Load executable
- [● Use project binary: SysTick_TimeBase.elf]
- ○ Use file: Workspace... File System...
- Executable offset (hex):

Runtime Options

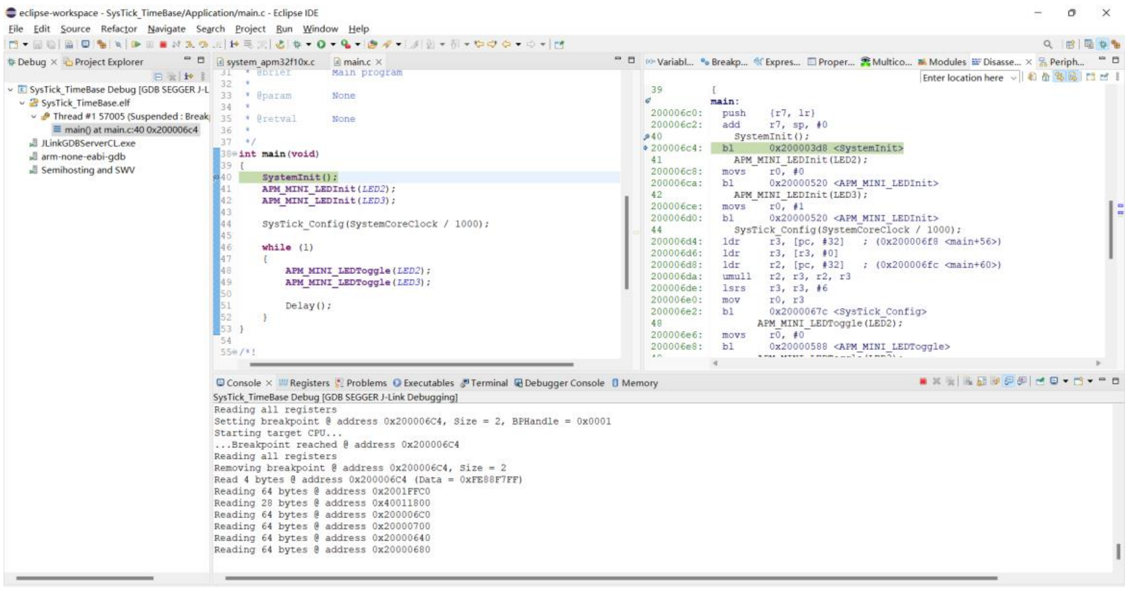
- ✓ RAM application (reload after each reset/restart)

Run/Restart Commands

- ✓ Pre-run/Restart reset Type: (always executed at Restart)

Step 4: Debug. The following figure shows the Debug view during debugging in RAM.

Figure 52 Debug View during debugging in RAM



SysTick_TimeBase Debug [GDB SEGGER J-L
SysTick_TimeBase.elf
Thread #1 57005 (Suspended : Break
main() at main.c:40 0x200006c4
JLinkGDBServerCL.exe
arm-none-eabi-gdb
Semihosting and SWV

```c
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
 *
 */
int main(void)
{
    SystemInit();
    APM_MINI_LEDInit(LED2);
    APM_MINI_LEDInit(LED3);

    SysTick_Config(SystemCoreClock / 1000);

    while (1)
    {
        APM_MINI_LEDToggle(LED2);
        APM_MINI_LEDToggle(LED3);

        Delay();
    }
}

/*!
```

```
39          {
          main:
200006c0:   push    {r7, lr}
200006c2:   add     r7, sp, #0
40            SystemInit();
200006c4:   bl      0x200003d8 <SystemInit>
41            APM_MINI_LEDInit(LED2);
200006c8:   movs    r0, #0
200006ca:   bl      0x20000520 <APM_MINI_LEDInit>
42            APM_MINI_LEDInit(LED3);
200006ce:   movs    r0, #1
200006d0:   bl      0x20000520 <APM_MINI_LEDInit>
44            SysTick_Config(SystemCoreClock / 1000);
200006d4:   ldr     r3, [pc, #32]   ; (0x200006f8 <main+56>)
200006d6:   ldr     r3, [r3, #0]
200006d8:   ldr     r2, [pc, #32]   ; (0x200006fc <main+60>)
200006da:   umull   r2, r3, r2, r3
200006de:   lsrs    r3, r3, #6
200006e0:   mov     r0, r3
200006e2:   bl      0x2000067c <SysTick_Config>
48                APM_MINI_LEDToggle(LED2);
200006e6:   movs    r0, #0
200006e8:   bl      0x20000588 <APM_MINI_LEDToggle>
```

SysTick_TimeBase Debug [GDB SEGGER J-Link Debugging]

```
Reading all registers
Setting breakpoint @ address 0x200006C4, Size = 2, BPHandle = 0x0001
Starting target CPU...
...Breakpoint reached @ address 0x200006C4
Reading all registers
Removing breakpoint @ address 0x200006C4, Size = 2
Read 4 bytes @ address 0x200006C4 (Data = 0xFE88F7FF)
Reading 64 bytes @ address 0x2001FFC0
Reading 28 bytes @ address 0x40011800
Reading 64 bytes @ address 0x200006C0
Reading 64 bytes @ address 0x20000700
Reading 64 bytes @ address 0x20000640
Reading 64 bytes @ address 0x20000680
```

6 How to use printf for printing

6.1 Steps of use

Step 1: Add the syscall.c file and add the following code to the file to define by adding _write function, and redirect usart to __io_putchar function.

```
#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;
}
```

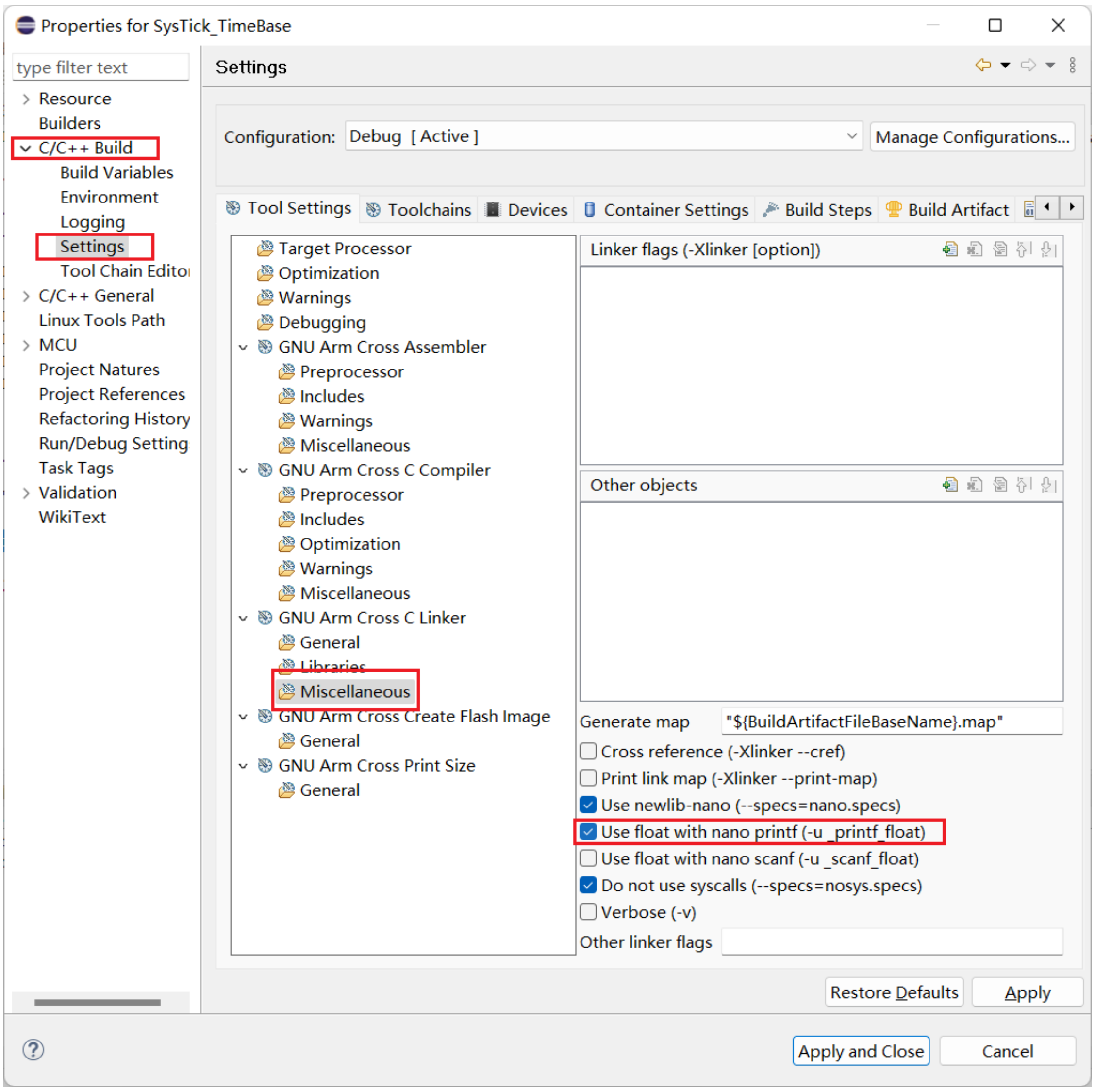
Step 2: Use the printf function to print normally.

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

6.2 Configuration of printing floating point data

In the project view, select File->Properties option, and in C/C++ Build->Settings->Tool Settings->GNU Arm Cross C Linker->Miscellaneous, check -u _prinft_float option.

Figure 53 Check -u _printf_float Option



Configuration: Debug [Active]

Generate map: "${BuildArtifactFileBaseName}.map"

- [ ] Cross reference (-Xlinker --cref)
- [ ] Print link map (-Xlinker --print-map)
- [✓] Use newlib-nano (--specs=nano.specs)
- [✓] Use float with nano printf (-u _printf_float)
- [ ] Use float with nano scanf (-u _scanf_float)
- [✓] Do not use syscalls (--specs=nosys.specs)
- [ ] Verbose (-v)

Other linker flags:

Note:

1. When using printf printing, add \r\n at the end of the printed content, such as printf("Running!\r\n").

2. If there is a high requirement for code size, it is not recommended to use printf. If printf is used in GCC, the code size will be greatly increased.

7 Revision History

Table 1 Document Revision History

Date	Revision	Changes
June 13, 2023	1.0	First release

Statement

used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If loses are caused to users due to the user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Compliance requirements

Users shall abide by all applicable local laws and regulations when using this manual and the matching Geehy products. Users shall understand that the products may be restricted by the export, re-export or other laws of the countries of the product suppliers, Geehy, Geehy distributors and users. Users (on behalf of itself, subsidiaries and affiliated enterprises) shall agree and promise to abide by all applicable laws and regulations on the export and re-export of Geehy products and/or technologies and direct products.

6. Disclaimer

This manual is provided by Geehy "as is". To the extent permitted by applicable laws, Geehy does not provide any form of express or implied warranty, including without limitation the warranty of product merchantability and applicability of specific purposes.

Geehy will bear no responsibility for any disputes arising from the subsequent design and use of Geehy products by users.

7. Limitation of liability

In any case, unless required by applicable laws or agreed in writing, Geehy and/or any third party providing this manual "as is" shall not be liable for damages, including any general damages, special direct, indirect or collateral damages arising from the use or no use of the information in this manual (including without limitation data loss or inaccuracy, or losses suffered by users or third parties).

8. Scope of application

The information in this manual replaces the information provided in all previous versions of the manual.