Application Note

Document No.: AN1126

Instructions for Use of G32R501 IDE and Tool Chain

Version: V1.1

1. Introduction

In modern embedded system development, choosing appropriate development tools and environments is crucial for successfully achieving the project goals. Regardless of development of new projects or porting of existing projects, developers need to flexibly respond to different development environments and chip architectures. This document aims to provide a detailed guide for developers to help them smoothly configure and use these two popular development environments when developing with the G32R501 MCU.

To make full use of the information in this guide, users shall be familiar with the characteristics of G32R501 series MCU. Users can refer to the following relevant technical documents:

- G32R501 Series User Manual, Datasheet, Programming Manual, and Migration Manual
- Related documents for G32R501 series core, including ARMv8.1-M Architecture Reference Manual, etc.

This document provides a detailed introduction to the steps of configuring projects in MDK-ARM and IAR EW for Arm, covering the entire process from project creation, compilation linking to debugging. By comparing the characteristics and settings of different development environments, readers can better understand how to efficiently develop embedded projects in different development environments.

This document will guide readers to master the skills of developing G32R501 in MDK-ARM and IAR EW for Arm environments, to lay a solid foundation for the successful implementation of the projects.

1.1. Full Name and Abbreviation Description of Terms

The keywords and descriptions used in this document are as shown in Table 1.

Table 1 Abbreviation Description

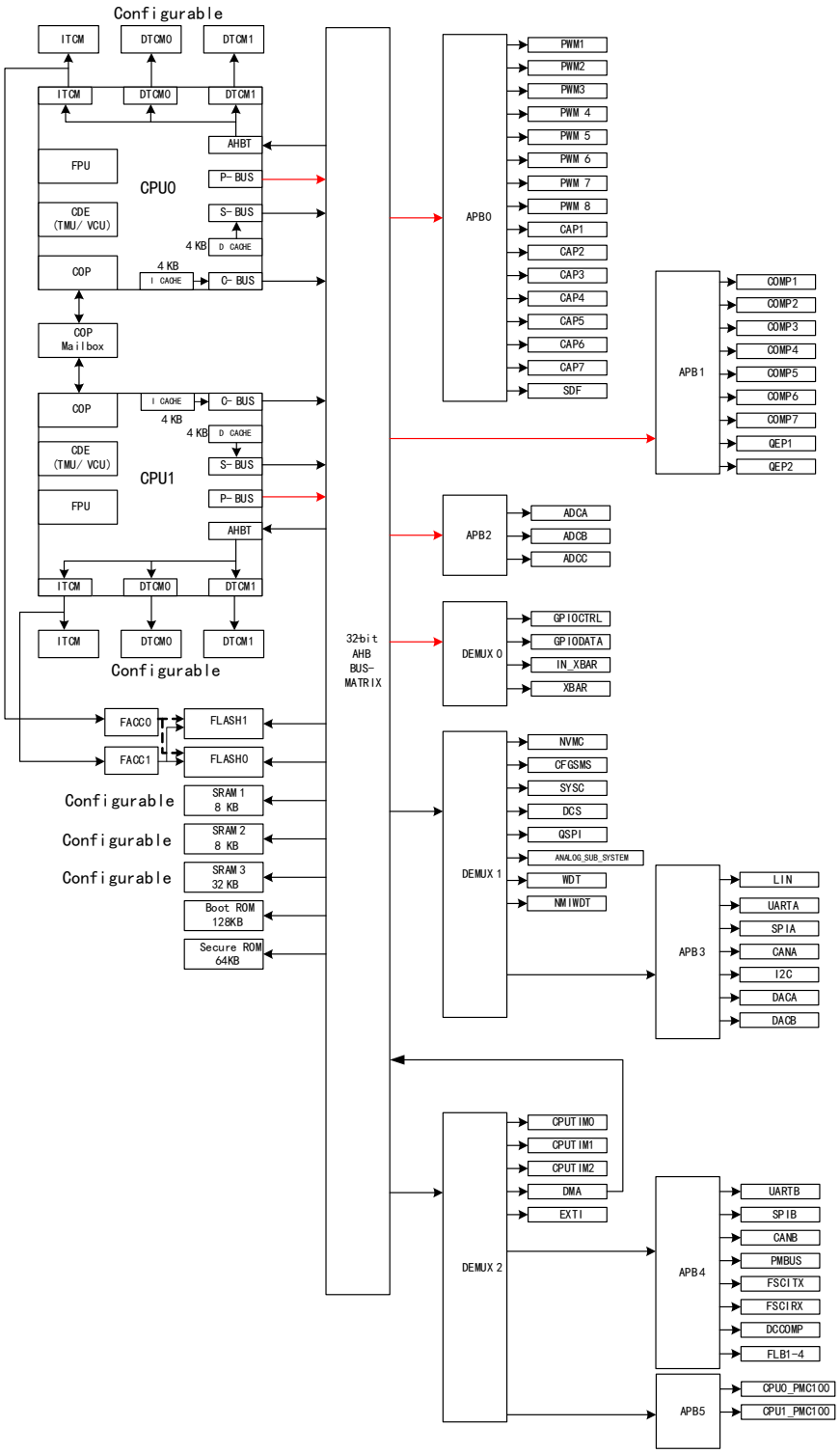
Full name in English	English abbreviation
Configurable Static Memory Subsystem	CFGSMS
Software	SW
Hardware	HW
Interface	IF
AHB slave interface	ahbs_if

1.2. R5xx SoC Introduction

R5xx SoC is a system on a chip (SoC) based on Cortex-M52 dual cores. This SoC adopts the widely used AMBA 2.0 bus to integrate peripheral IP components: the components that require broad bandwidth are connected to the AMBA AHB bus, while the low-speed components are connected to the AMBA APB bus. DMA cannot access the registers related to FLB.

The architecture of R5xx SoC is as shown in Figure 1.

Figure 1 R5xx SoC Architecture



Note:

1) The red lines in the figure represent bus bridge.
2) The size of ITCM/DTCM/SRAM is configured using CFGSMS.

Contents

2. MDK-ARM Development Tool Chain

During the migration from Txx320F28004x series MCU to G32R501 series MCU, the porting of development tools is an important link. Code Composer Studio (CCStudio) integrated development environment (IDE) and MDK-ARM are both commonly used tools in embedded development.

This chapter will introduce how to migrate the existing CCStudio projects to the MDK-ARM environment, and discuss in detail the compatibility of C language and assembly code, and the configuration of linker script files.

2.1. Simulator Support

Support of simulators by G32R501:

- Geehy-Link (WinUSB), DAP Link (the firmware version is CMSIS-DAP V2 and above)
- J-Link V12 (J-Link V7.94g and above)
- Ulink Pro

2.2. IDE Version

Please make sure to use MDK-ARM V5.40 or higher-version IDE.

Note: The known problem with MDK 5.40/5.41 is that the function jump (press F12, and it will jump to the function definition) and other functions during code editing cannot be used properly.

2.3. Project Operation

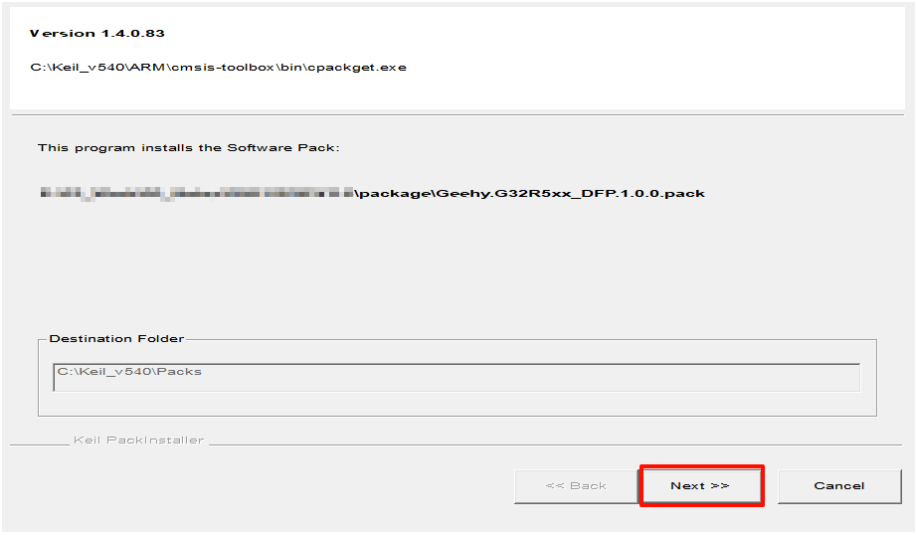
Note: The following operations are all performed on MDK-ARM v5.40.

2.3.1. Install Pack support

You can choose any of the following methods for installation:

1. Direct installation
   - Look for the file “Geehy.G32R5xx_DFP.x.x.x.pack” under the package directory of SDK.
   - Double-click the file and install in the pop-up installation interface (as shown in Figure 2) according to the instructions.

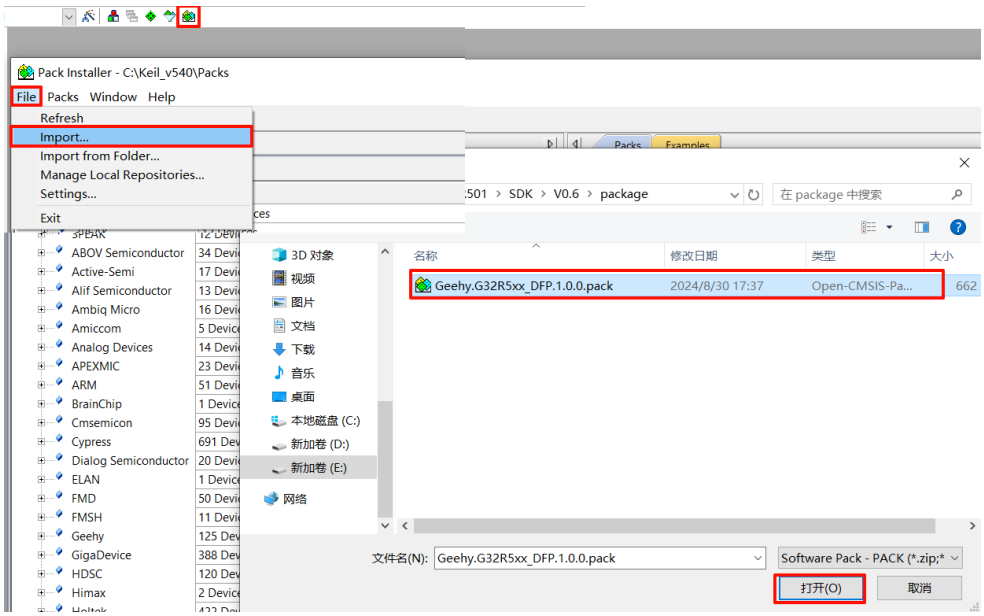
Figure 2 Click Install in Geehy.G32R5xx_DFP.x.x.x.pack



2. Use Pack Installer (as shown in Figure 3)

- Open MDK-ARM v5.40.
- In the menu bar, click “Project”->“Manage”->“Pack Installer”.
- This will open a new window, displaying the available support packages.
- In the new window, select "File"->"Import...".
- In the file browser, try to find the file “Geehy.G32R5xx_DFP.x.x.x.pack” under the package directory of SDK, and then select.
- Click "Open" to install.

Figure 3 Import Pack Installer to Geehy.G32R5xx_DFP.x.x.x.pack



2.3.2. Open the example project

1. Start MDK-ARM v5.40.
2. Click "Project"->"Open Project" in the menu bar.
3. Browse the path of the SDK project file you provide, select the corresponding .uvprojx file, and then click "Open".

Or after installing MDK-ARM v5.40, directly click the project file ".uvprojx file" to open it.

Note: Please complete the above steps after the Pack support installation is completed; otherwise MDK-ARM will indicate that the chip cannot be found.

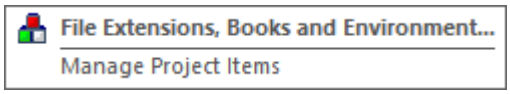
2.3.3. Project establishment

1. Start MDK-ARM:
   - Open MDK-ARM v5.40.
2. Create a new project:
   - Click "Project"->"New uVision Project" in the menu bar.
   - Select the path to save the project, enter the project name, and click "Save".
3. Select the MCU:
   - In the pop-up dialog box, select the appropriate G32R501 series MCU model, and click "OK".

2.3.4. File import

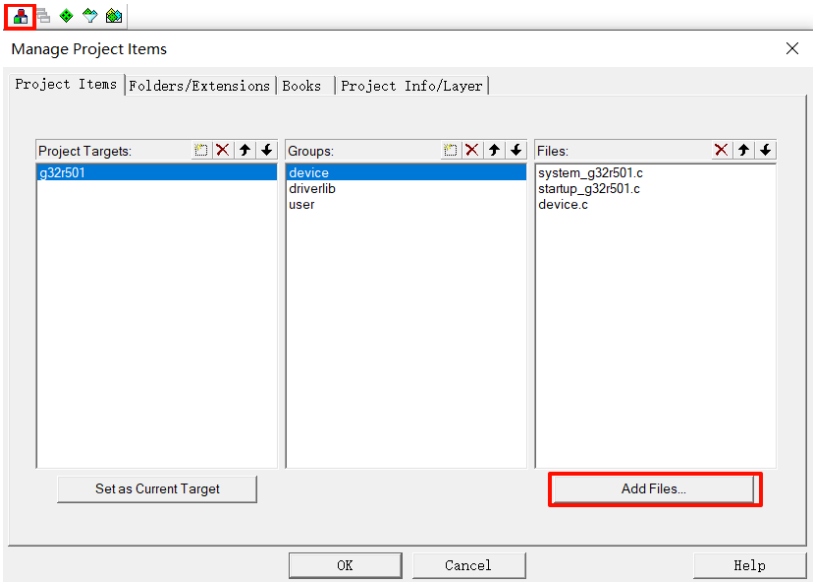
1. Open the project view:
   - Make sure your new project is already open in the Project window.
2. Add an existing file:
   - Click "File Extensions, Books and Environment…"

Figure 4 File Extensions



   - Click "Source Group 1" or the folder where you want to add files, and select "Add Files...".
   - In the pop-up file browser, find and select the source files to be imported (e.g. .c and .h files), and then click "Add".

Figure 5 Add Source Files



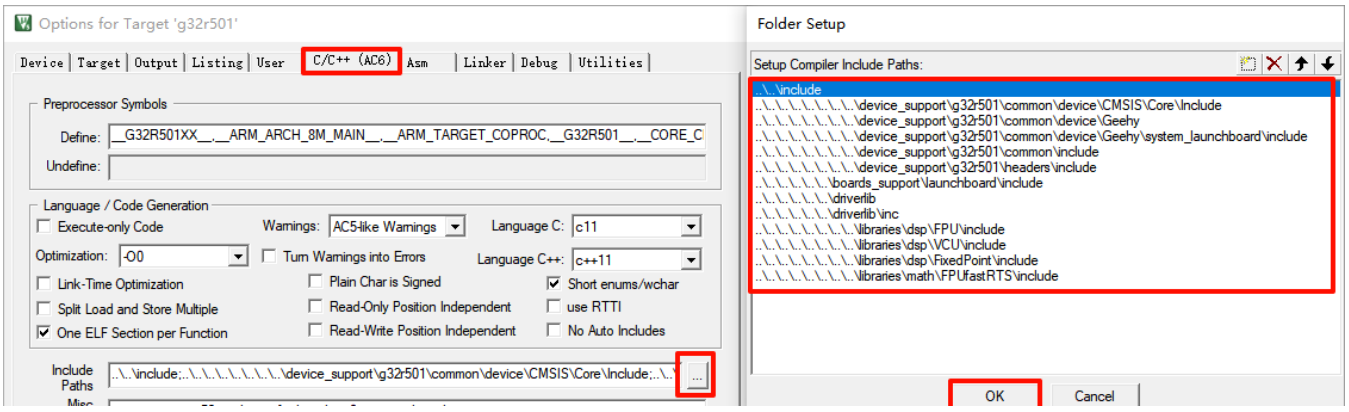
3. New file (if needed):

   - To create a new source file, right-click "Source Group 1" and select "Add New Item".
   - Enter the file name, select the file type (e.g. C file or assembly file), and then click "Add".

4. Configure the header file path (as shown in Figure 6):

   - Select "Project" ->"Options" in the menu, check "Include Paths" under the "C/C++" tab, and ensure that all necessary library file paths have been added.

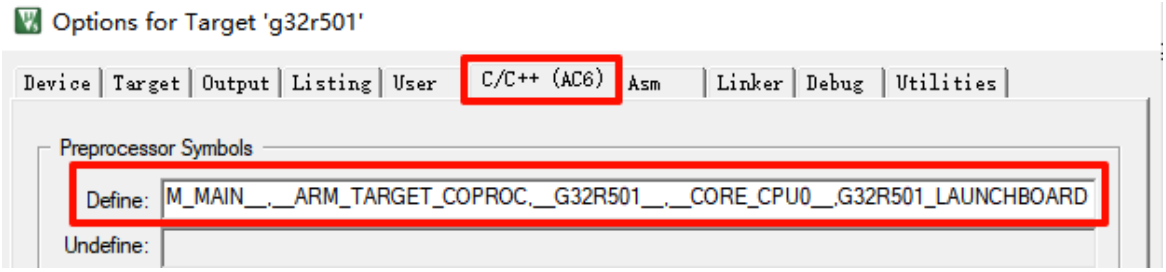
Figure 6 Add the Header File Path



2.3.5. Configure macro definition

In "Project"->"Options", select the "C/C++" tab, add the required macro definition, and configure the corresponding compilation control.

Figure 7 Configure Macro Definition



2.3.6. Compilation command control

In MDK-ARM, compilation commands can be controlled through the "Misc Controls" under the "C/C++(AC6)" tab in the "Options for Target" window of the project attributes. Under the "C/C++(AC6)" tab, users can set the preprocessor directives, compiler flags, and so on, refer to the compiler command support in Table 3; for more related information, please consult the help documentation in MDK.

Table 2 Compiler Command

Characteristic/Option	Scalar FP half-precision	Scalar FP single-precision	Scalar FP double-precision	MVE integer	MVE FP half-precision	Custom Datapath Extension (CDE) cp0
cortex-m52	Included	Included	Not included	Not included	Included	Not included
cortex-m52+nomve	Included	Not included	Included	Not included	Included	Not included
cortex-m52+nomve.fp+nofp.dp	Included	Not included	Included	Not included	Included	Not included
cortex-m52+nomve+nofp.dp	Not included	Not included	Included	Not included	Included	Not included
cortex-m52+nomve.fp+nofp	Included	Not included	Not included	Not included	Included	Not included
cortex-m52+nopacbti	Included	Included	Not included	Not included	Not included	Not included
cortex-m52+nomve+nopacbti	Included	Not included	Not included	Not included	Not included	Not included
cortex-m52+ nomve.fp+nofp.dp+nopacbti	Included	Not included	Not included	Not included	Not included	Not included
cortex-m52+ nomve+nofp.dp+nopacbti	Not included	Not included	Not included	Not included	Not included	Not included
cortex-m52+nomve.fp+nofp+nopacbti	Not included	Not included	Included	Not included	Not included	Not included

Characteristic/Option	Scalar FP half-precision	Scalar FP single-precision	Scalar FP double-precision	MVE integer	MVE FP half-precision	Custom Datapath Extension (CDE) cp0
cortex-m52+cdecp0	Not included	Not included	Not included	Not included	Not included	Included

Figure 8 Help Documentation in MDK

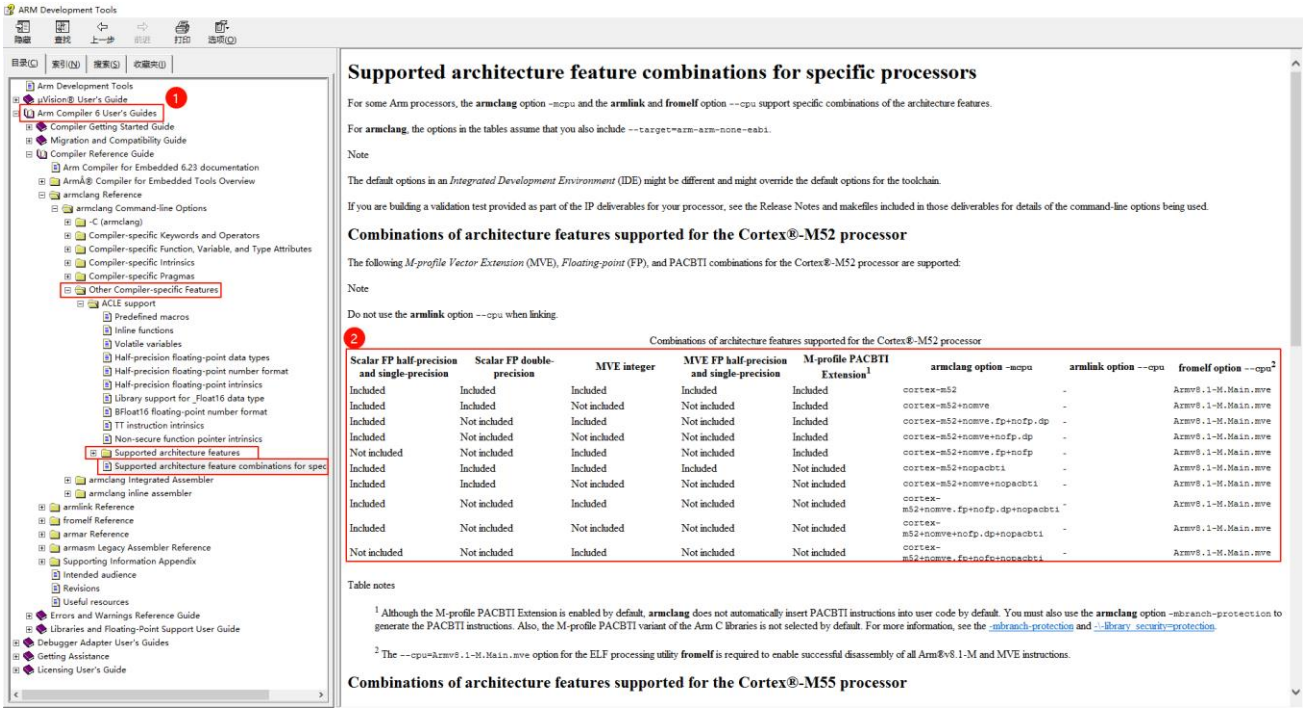
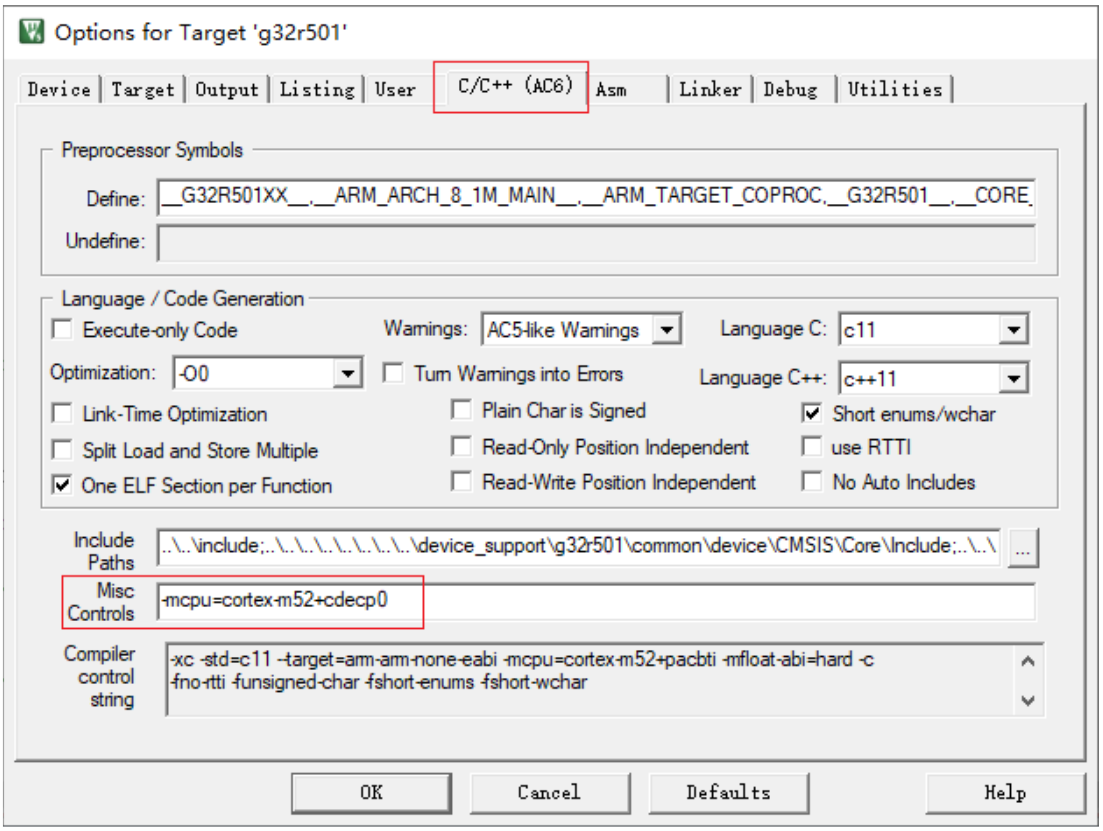


Figure 9 Compilation Command Control

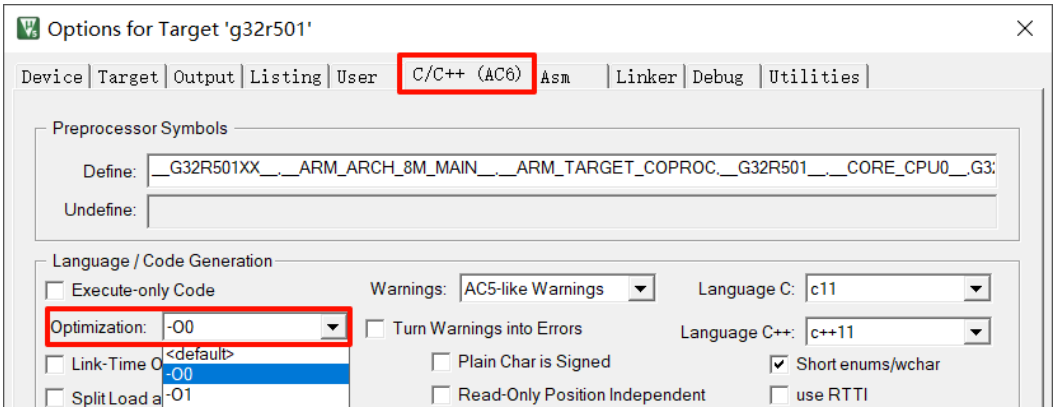


2.3.7. Compilation optimization level settings

MDK-ARM provides multiple optimization level settings, which can be adjusted at the global, single-file, and single-function levels:

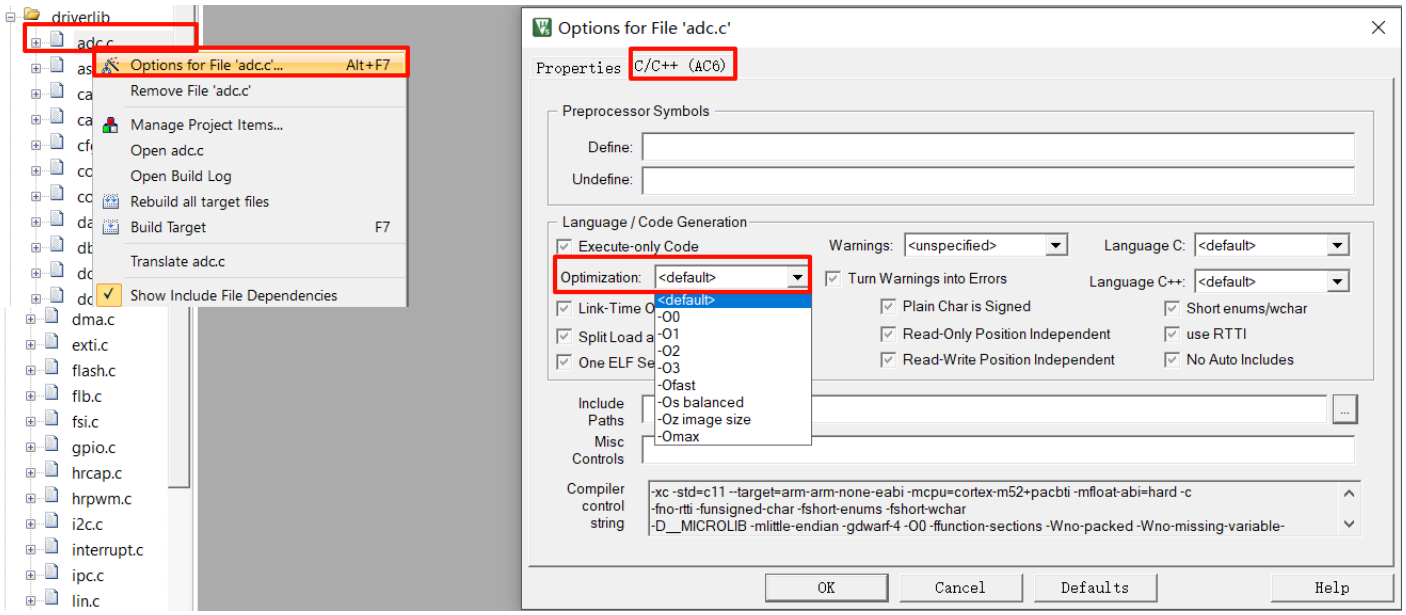
1. Global optimization level settings: In the "Options for Target" window, select the "C/C++" tab, and then select an appropriate optimization level from the "Optimization" drop-down menu.

Figure 10 Global Optimization Level Settings



2. Single-file optimization level settings: Right-click a specific source file, select "Options for File...", and then set the optimization level in the "C/C++" tab.

Figure 11 Single-file Optimization Level Settings



3. Single-function optimization level settings: Specific compiler instructions can be used for optimization setting before the function, e.g. using __attribute__((optnone)) to declare no optimization or use of __attribute__((optimize("O2"))) for optimization.

2.3.8. Program compilation

1. In the menu bar, click "Project" ->"Build Target" (or directly click the "Build" button on the toolbar, or press the "F7" button).

Figure 12 Compiler Program

2. After the compilation process is completed, check for any errors or warning messages in the output window. If there are errors, make corresponding modifications according to the prompt

2.3.9. Program download

1. Select the simulator:

- Click "Project" ->"Options" in the menu bar.
- In the pop-up dialog box, select the "Debug" tab.

- In the "Use" drop-down menu, select the appropriate debugging simulator (e.g. Geehy-Link, J-Link, etc.).

2. Configure the download script:

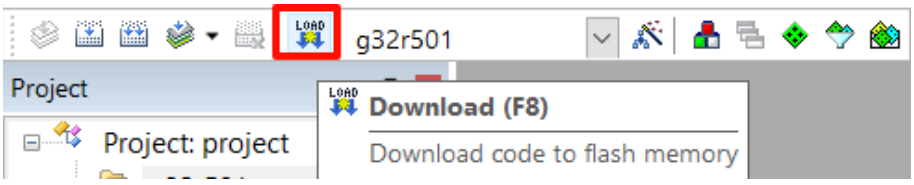
- Open the project options: Select "Options for Target".
- Select the downloading tool: In the "Utilities" tab, select the appropriate downloading tool.
- Load the download script: Click the "Settings" button to load the specific download script file.

Figure 13 Select Download Script

3. Download program:

- On the toolbar, click the "Download" button (small arrow icon) or select "Flash" ->"Download" from the menu (or press "F8" directly).

Figure 14 Download Program



- Ensure that the target device is connected, wait for the download to be completed, and view the output window to confirm the download status.

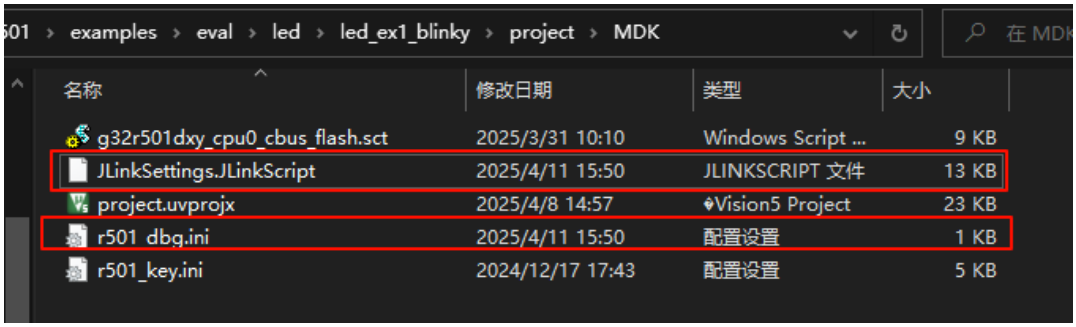
2.3.10. Program simulation

2.3.10.1. J-Link emulation

- Open the project options: Select "Options for Target".
- Select the debugger: In the "Debug" tab, choose the J-Link debugger.

- Add the r501_dbg.ini and JLinkSettings.JLinkScript files in the target project folder.
  - The JLinkSettings.JLinkScript file is a C-like scripting language used to customize the operation of the J-Link debugger. The JLinkScript file includes basic syntax, custom operations, API functions, and global constants (variables) from DLLs, and its syntax is similar to that of the C language.
  - The r501_dbg.ini and JLinkSettings.JLinkScript files are located in SDK/device_support/g32r501/common/Jlink/.

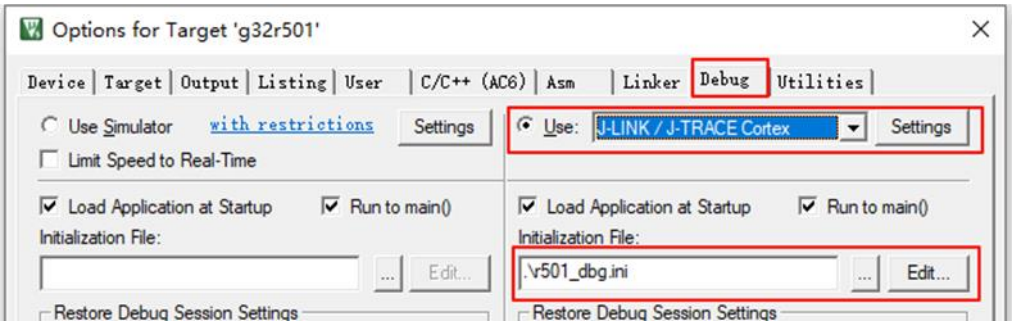
Figure 15 Add J-Link document



Note: The r501_dbg.ini file is different from the r501_dbg.ini file used with GEEHY LINK emulation; please make sure to distinguish between them when in use.

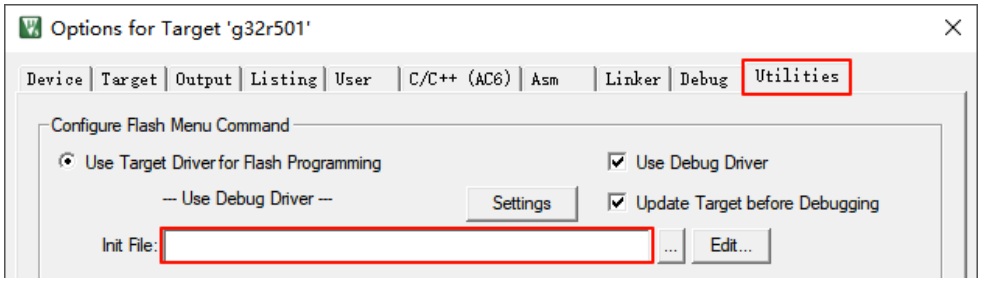
4. Load the emulation script: In the "Settings" menu, click the "Load" button to load the specific emulation script file.

图 16 J-Link emulator and emulation script.



5. Remove the download script: When using the J-Link emulator, no additional download scripts are required. Please delete the setting for the "Init File" in the Utilities download script.

Figure 17 Remove the download script



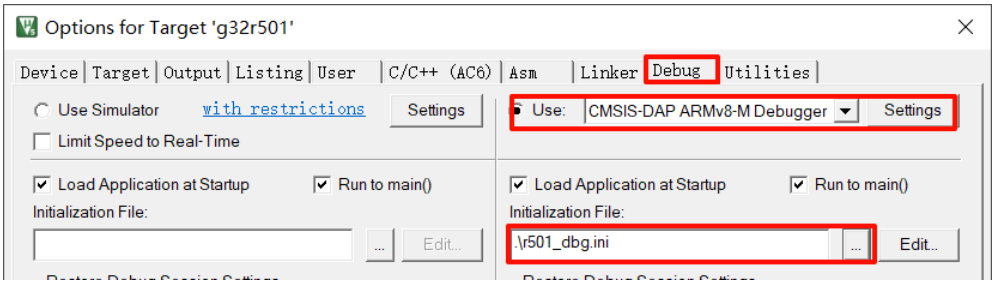
6. Startup debugging:

   - After the download is complete, click the "Debug" button to start debugging the program.
   - In debug mode, it can set breakpoints, view variables, perform step-by-step execution, and other operations.

2.3.10.2. Geehy Link emulation

1. Open the project options: Select "Options for Target".
2. Select the debugger: In the "Debug" tab, choose the appropriate debugger.
3. Load the simulation script: In the "Settings" menu, click the "Load" button to load a specific simulation script file.

Figure 18 Select the Simulator and Simulation Script



4. Start debugging:

- After downloading, click the "Debug" button to start the debugging the program.
- In debugging mode, such operations as setting breakpoints, viewing variables, and single-step execution can be performed.

2.4. C Language Compatibility

When migrating the tool chains, the compatibility of C language code needs special attention. Different compilers may have differences in file format support, built-in functions, memory

operation, and data type. To ensure that the code can be compiled and run correctly in a new compilation environment, necessary adjustments and optimization shall be made on the existing code.

2.4.1. File format support

Support .c/.h files.

2.4.2. Use of sizeof

Different compilers and architectures may have differences in the size of data type, so special attention shall be paid to the results of the sizeof operator on different platforms. The following is a size comparison of common data types on G32R501 and Txx320F28004x:

Table 3 Size Comparison of Different Data Types on G32R501 and Txx320F28004x

Data type	G32R501	Txx320F28004x
char	1	1
short	2	1
int	4	1
long	4	2
long long	8	4
float	4	2
double	8	4
long double	8	4
void*	4	2
int8_t	1	1
uint8_t	1	1
int16_t	2	1
uint16_t	2	1
int32_t	4	2
uint32_t	4	2
int64_t	8	4
uint64_t	8	4

2.5. Assembly Compatibility

There may be significant differences in instruction set and assembly syntax among different target platforms, so the existing assembly code needs to be rewritten and adapted to ensure correct operation on the new platforms

2.5.1. File format support

AC6 is based on LLVM and Clang technology, mainly using GNU-style assembly syntax. Assembly file formats supported by AC6:

- .s file: GNU-style assembly file format.

Meanwhile, AC6 supports the use of inline assembly in C functions.

2.5.2. Assembly code format requirements

1. Single assembly file: Here is a simple assembly code example, which defines an assembly function add, used to add two integers. The content of the file “add.s” is as follows:

```
.syntax unified
.global add
.type add, %function


add:
    @ Function entry
    @ Parameters: r0 and r1
    @ Return value: r0
    adds r0, r0, r1     @ Add r0 and r1, store result in r0
    bx lr                 @ Return to calling function

    .end
```

2. Use inline assembly in C function: The following is an example of using inline assembly in C function, and an inline assembly function add_inline is defined to add two integers:

```
// Inline assembly function
static inline int add_inline(int a, int b) {
    int result;
    __asm volatile (
        "adds %0, %1, %2\n"
        : "=r" (result)          // Output operand
        : "r" (a), "r" (b)       // Input operands
        : "cc"                   // Clobbered registers
    );
    return result;

}
```

2.6. Linker Script Files

The linker script files are used to define the memory layout and section allocation of the program. In the migration process, it is necessary to use linker script files in the corresponding format according to different target platforms and development environments. The G32R501 uses linker script files in the ".sct" format in the MDK development environment, which comply with Arm Company's specifications.

Differences in linker script files

- File format:
  - The G32R501 uses linker script files in the ".sct" format, which comply with Arm Company's specifications.
  - Txx320F28004x uses the linker script files in ".CMD" format, which comply with the company's specifications.
- Memory layout and section allocation:
  - The ".sct" file of G32R501 arranges memory allocation by defining the load region and execution region.
  - The ".CMD" file of Txx320F28004x arranges memory allocation by defining the Memory section (MEMORY) and section allocation (SECTIONS).

2.7. RAM Operation

The Txx320F28004x compiler supports the Progma syntax, which tells the compiler that if a specific function, target file, or the attributes of a section of code are modified, such as CODE_SECTION, a Section is assigned to a certain function, and its usage is as follows:

```
#pragma CODE_SECTION(funcA, "codeA")
```

G32R501 can implement the same function using the following statement:

```
__attribute__((section("xxx")))
```

Where, xxx represents the SECTION name which a certain function or global variable is assigned. During use, users can refer to the following format:

```
__attribute__((section("itcm.ramfunc"))) void SysCtl_delay(uint32_t count){}
```

You only need to specify the attributes when defining functions and variables.

Note: When using "__attribute__((section("itcm.ramfunc")))", a declaration of the "*itcm.ramfunc*" field in the .sct (chained footstep file) is required: *.ANY (itcm.ramfunc)*

3. IAR EW for Arm Development Tool Chain

3.1. Simulator Support

Please refer to Chapter 2.1.

3.2. IDE Version

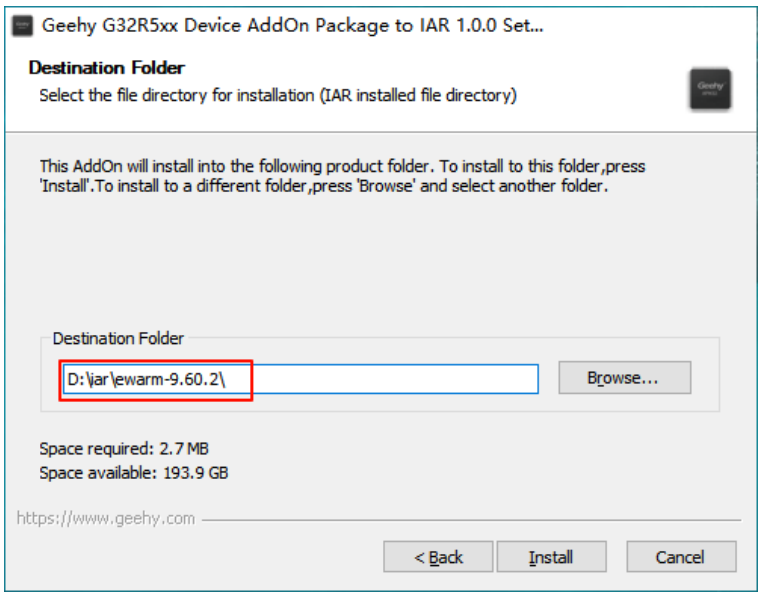
Please make sure to use IAR EW for Arm 9.60.2 or higher-version IDE.

3.3. Install the Chip Support

Before officially using IAR EW for Arm to develop G32R5 series MCU, please install the chip support package first. The path for chip support is: ..\utilities\G32R5xx_AddOn\G32R5xx_AddOn_vx.x.x.exe

1. Open G32R5xx_AddOn_vx.x.x.exe with the administrator rights and go to the interface for selecting the path to install the chip support. This path is the installation path for IAR EW for Arm, e.g. the example: D:\iar\ewarm-9.60.2\.

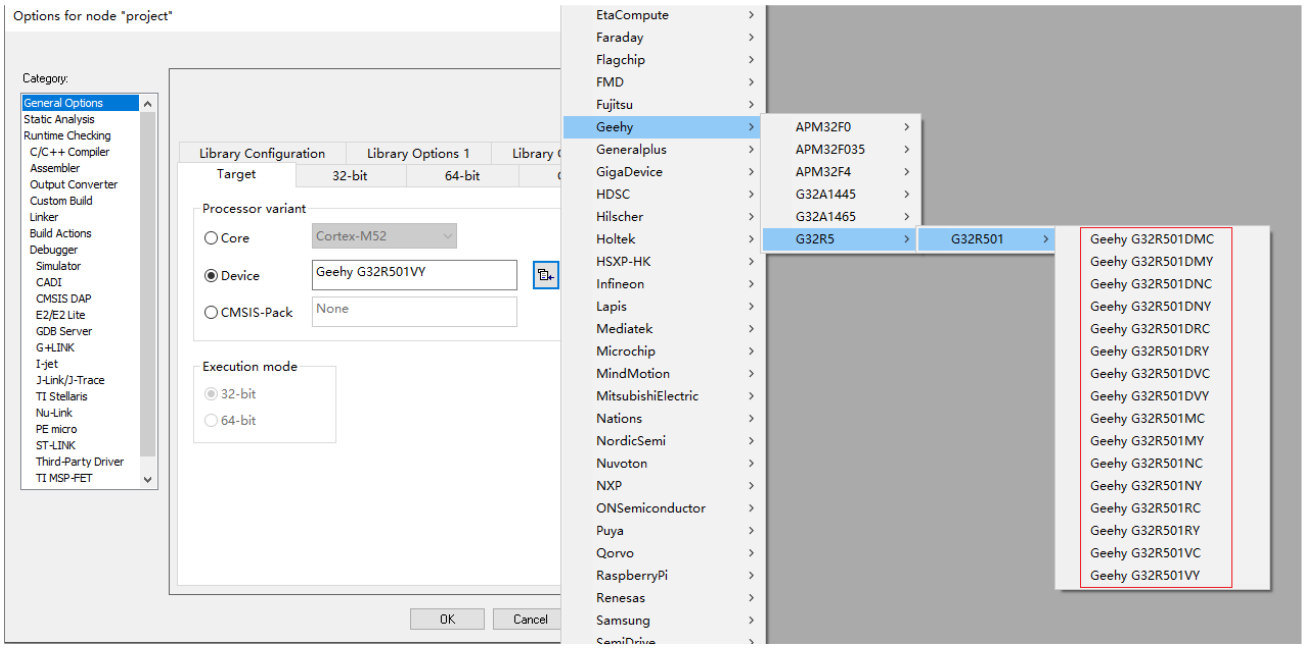
Figure 19 Installation of Chip Support on G32R5xx_AddOn_vx.x.x.exe



If the software cannot obtain the IAR EW for Arm installation path on the computer, please manually select it.

2. After selecting the correct path and adding the chip support, open IAR EW for Arm, select "New project", and in the chip selection tab, the G32R5 series MCU list can be seen.

Figure 20 G32R5 Series MCU List



3.4. Project Operation

3.4.1. Open the example project

1. Start IAR EW for Arm 9.60.2.
2. Click "File" ->"Open Workspace..." in the menu bar.
3. Browse the path of the SDK project file you provide, select the corresponding .eww file, and then click "Open".

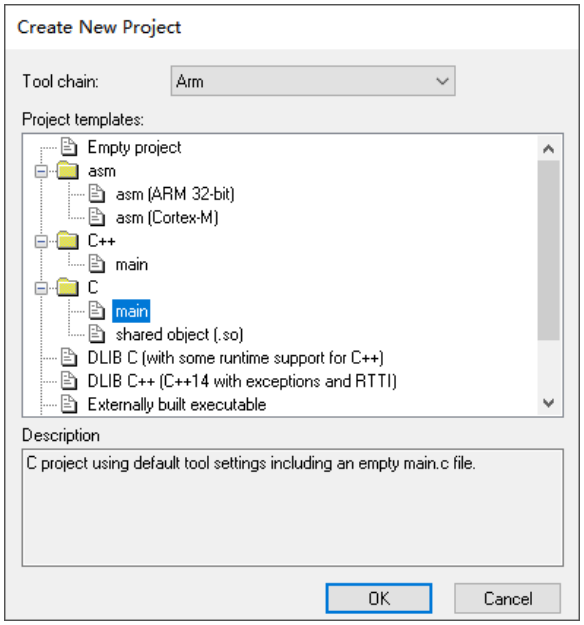
Or, after installing IAR EW for Arm 9.60.2, directly click the project file ".eww file" to open it.

Note: Please complete the above steps in Section 3.3 Install Chip Support; otherwise, the IAR EW for Arm will prompt that the chip cannot be found.

3.4.2. Project establishment

1. Start IAR EW for Arm:
   - Open IAR EW for Arm 9.60.2.
2. Create a new project:
   - Click "Project"->"Create New Project" in the menu bar.
   - Select "Tool chain" as "Arm".
   - Select the "main" under "C" and then click "OK".
   - Select the path to save the project, enter the project name, and click "Save".

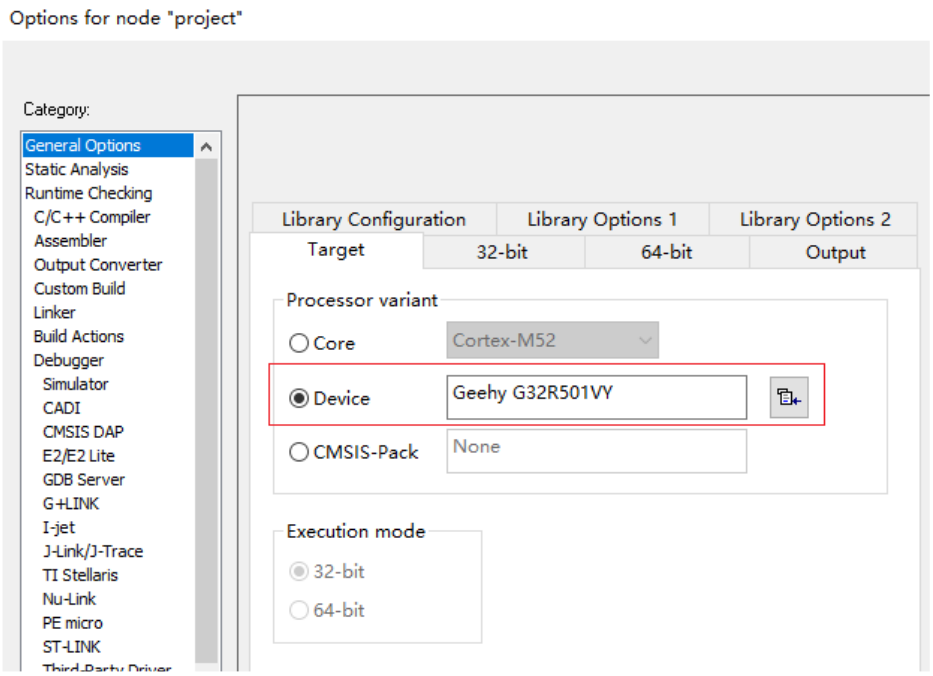
Figure 21 Create New Project



3. Select the MCU:

- Right-click the project name and select "Options...".
- Select "Device" under "Target" in "General Options".
- Select the appropriate G32R501 series MCU model, and click "OK".

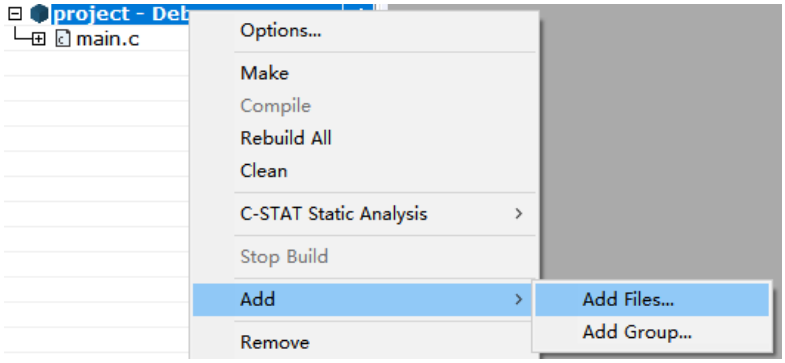
Figure 22 Select MCU



3.4.3. File import

1. Open the project view:
   - Make sure your new project is already open in the Project window.
2. Add an existing file:
   - Right-click the project name, select "Add", and then select "Add Files".

Figure 23 File Extensions

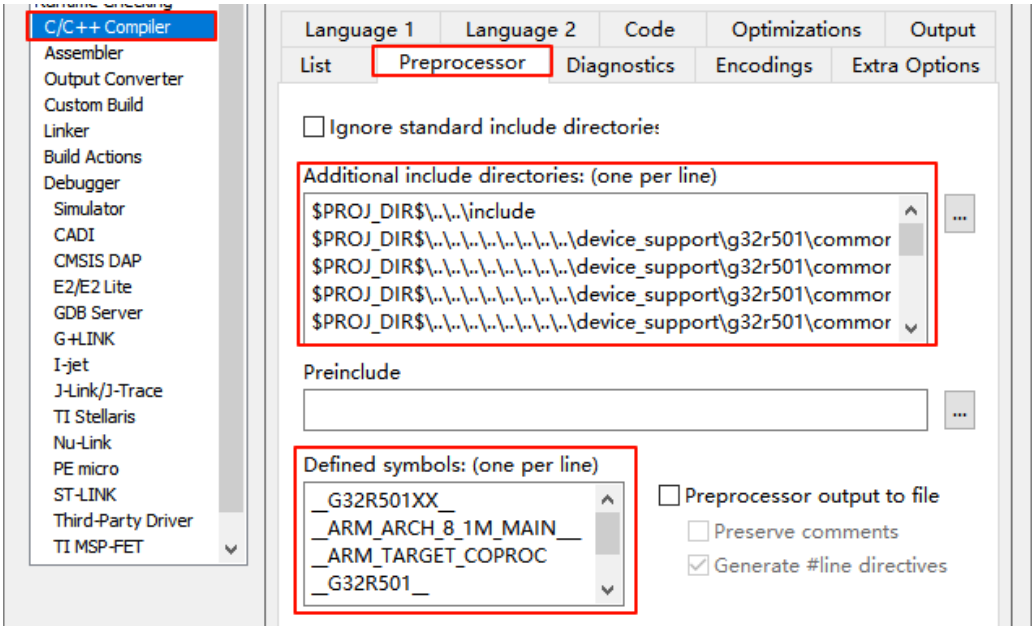


   - In the pop-up file browser, find and select the source files to be imported (e.g. .c and .h files), and then click "Add".
3. New file (if needed):
   - If a new source file needs to be created, click the "File" on the toolbar and select "New File".
   - A file will appear in the IDE; press "CTRL+S" to save the file.
   - Enter the file name, select the file type (e.g. C file or assembly file), and then refer to the steps for “Adding files”.

3.4.4. Configure header file path and macro definition

1. Right-click the project name and select "Options...".
2. Select “Preprocessor” under the “C/C++Compiler” tab.

Figure 24 Configure Header File Path and Macro Definition



3. Add all necessary library file paths in “Additional include directories: (one per line)”.

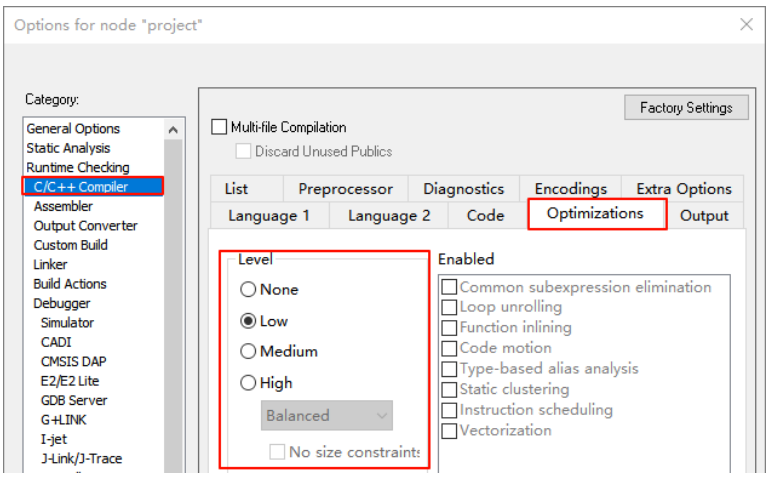
4. In Defined Symbols: The macro definition required for (one per line).

3.4.5. Compilation optimization level settings

IAR EW for Arm provides multiple optimization level settings, which can be adjusted at the global, single-file, and single-function levels:

1. Global optimization level settings: Open the "C/C++ Compiler" tab and then select the appropriate optimization level in "Optimizations".

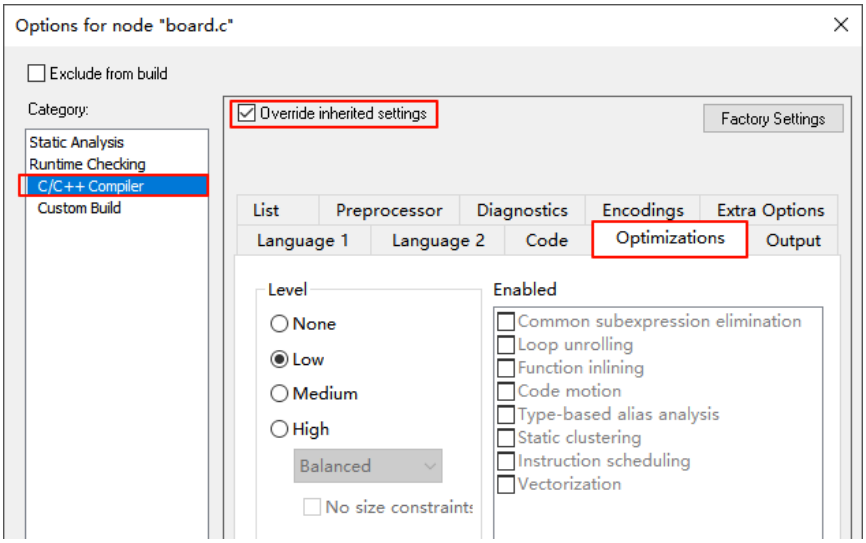
Figure 25 Global Optimization Level Settings



2. Single-file optimization level settings: Right-click a specific source file, select "Options...", check "Override inherited settings", and then select the appropriate optimization level in

"Optimizations".

Figure 26 Single-file Optimization Level Settings

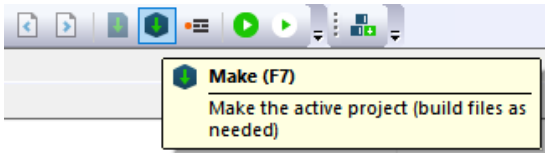


3. Single-function optimization level settings: Specific compiler instructions can be used before the function to set the optimization level, e.g. declaring no optimization using Pragma("optimize=none").

3.4.6. Program compilation

1. In the menu bar, click "Project" ->"Make" (or directly click the "Make" button on the toolbar, or press the "F7" button).

Figure 27 Compiler Program



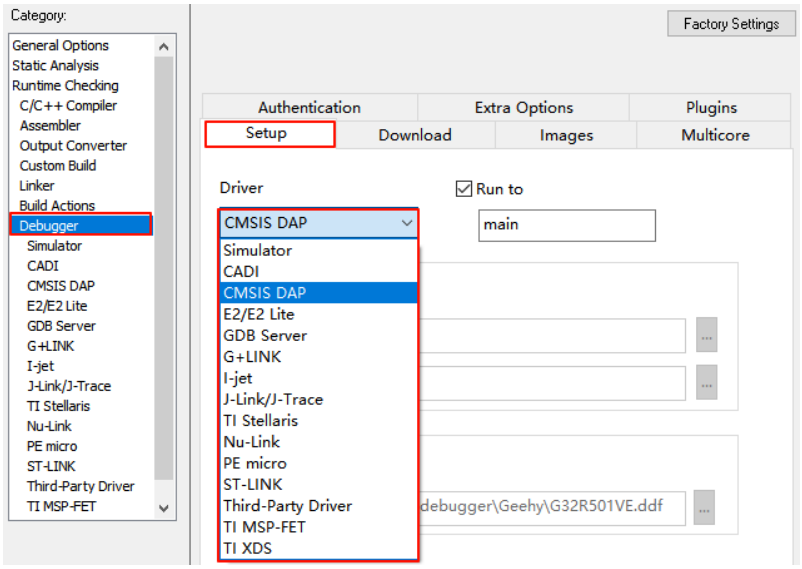
2. After the compilation process is completed, check for any errors or warning messages in the output window. If there are errors, make corresponding modifications according to the prompt

3.4.7. Program simulation and download

1. Select the simulator:
   - Right-click the project name and select "Options...".
   - In the pop-up dialog box, select the "Debugger" tab.
   - In the "Driver" drop-down menu under the "Setup" tab, select the appropriate

debugging simulator ((e.g. Geehy-Link (CMSIS DAP), J-Link, etc.).

Figure 28 Select the Simulator

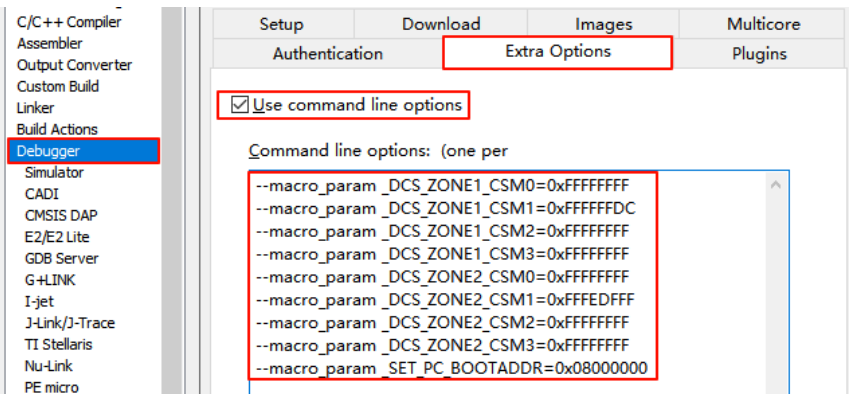


2. Configure simulation instructions:

As the G32R5 series MCU supports DCS encryption, corresponding instructions need to be configured for normal simulation.

- Right-click the project name and select "Options...".
- In the pop-up dialog box, select the "Debugger" tab.
- Check "Use command line options" under the "Extra Options" tab, and add instructions to "Command line options"
  - “--macro_param _DCS_ZONE1_CSM0=0xFFFFFFFF”, setting the DCS key.
  - “--macro_param _SET_PC_BOOTADDR=0x08000000”, setting the boot address

Figure 29 Configure Simulation Instructions

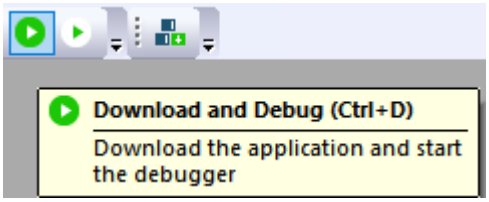


3. Erase and download programs:

- On the toolbar, click "Project" and select "Download" from the menu
  - “Erase memory": Erases the chip Flash.
  - “Download active application”: download the program of this project to the chip.
  - “Download file...": Download other programs to the chip

4. Program simulation:
   - Click the "Download and Debug" button on the toolbar or press "Ctrl+D" to start the simulation.

Figure 30 IAR Download and Debug



   - In debugging mode, such operations as setting breakpoints, viewing variables, and single-step execution can be performed.

5. Problems in Debugging
   - During debugging, the Flash content in the Memory window does not update.

     Reason: The AccType of the Flash corresponding Memory region in the ddf (device description file) is set to "R", which means the debugger has read-only access to the Flash and cannot modify its content. Therefore, the debugger will not update the values of the corresponding Memory region during the debugging process.

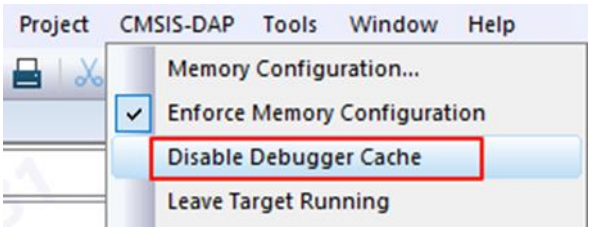
Figure 31 ddf file (Excerpt)

```
[Memory]
;;          Name                 AdrSpace    StartAdr     EndAdr       AccType   Width
Memory =   Boot_ROM             Memory      0x10000000   0x1001FFFF   R
Memory =   Secure_ROM           Memory      0x10020000   0x1002FFFF   R
Memory =   ITCM_Flash           Memory      0x00100000   0x0019FFFF   R
Memory =   BusMatrix_Flash      Memory      0x08000000   0x0809FFFF   R
Memory =   CPU0_ITCM            Memory      0x00000000   0x0000BFFF   RW
Memory =   CPU1_ITCM            Memory      0x00000000   0x00001FFF   RW
Memory =   CPU0_DTCM            Memory      0x20000000   0x20003FFF   RW
```

Solution:

1) Enable “Disable Debugger Cache”

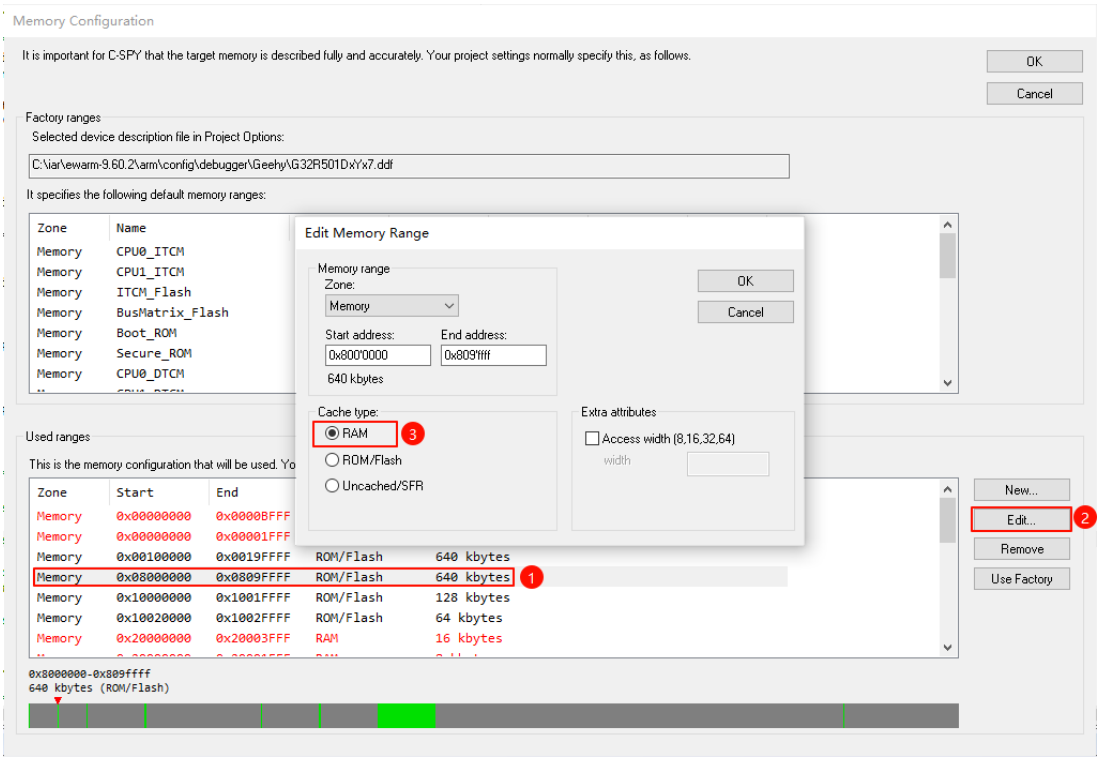
Figure 32 Disable Debugger Cache



2) Change the Cache Type of the Flash corresponding Memory region to RAM

Click on Memory Configuration to enter the Memory Configuration window and modify the Cache Type.

Figure 33 Cache type



3) Modify the AccType of the Flash corresponding Memory region in the ddf file to RW.

Figure 34 ddf file AccType

```
[Memory]
;;          Name                 AdrSpace    StartAdr     EndAdr       AccType   Width
Memory =   Boot_ROM             Memory      0x10000000   0x1001FFFF   R
Memory =   Secure_ROM           Memory      0x10020000   0x1002FFFF   R
Memory =   ITCM_Flash           Memory      0x00100000   0x0019FFFF   R
Memory =   BusMatrix_Flash      Memory      0x08000000   0x0809FFFF   RW
Memory =   CPU0_ITCM            Memory      0x00000000   0x0000BFFF   RW
Memory =   CPU1_ITCM            Memory      0x00000000   0x00001FFF   RW
Memory =   CPU0_DTCM            Memory      0x20000000   0x20003FFF   RW
Memory =   CPU1_DTCM            Memory      0x20000000   0x20001FFF   RW
Memory =   SRAM1                Memory      0x20100000   0x2011FFFF   RW
Memory =   SRAM2                Memory      0x20200000   0x2021FFFF   RW
Memory =   SRAM3                Memory      0x20300000   0x2031FFFF   RW
Memory =   APB0_Peripherals     Memory      0x40000000   0x4000FFFF   W
Memory =   APB1_Peripherals     Memory      0x40010000   0x4001FFFF   W
Memory =   APB2_Peripherals     Memory      0x40020000   0x4002FFFF   W
Memory =   DEMUX0_AHB           Memory      0x40030000   0x4003FFFF   W
Memory =   APB3_Peripherals     Memory      0x50000000   0x5000FFFF   W
Memory =   DEMUX1_AHB0          Memory      0x50010000   0x50027FFF   W
Memory =   DEMUX1_AHB1          Memory      0x60000000   0x6FFFFFFF   W
Memory =   APB4_Peripherals     Memory      0x50100000   0x50103FFF   W
Memory =   APB5_Peripherals     Memory      0x50104000   0x5010FFFF   W
Memory =   DEMUX2_AHB           Memory      0x50110000   0x5011FFFF   W
Memory =   PPB                  Memory      0xE0000000   0xE00FFFFF   W
```

More detail: 调试时 Memory 窗口中 Flash 内容不更新

3.5. C Language Compatibility

Please refer to Chapter 2 "C Language Compatibility".

3.6. Assembly Compatibility

There may be significant differences in instruction set and assembly syntax among different target platforms, so the existing assembly code needs to be rewritten and adapted to ensure correct operation on the new platforms

3.6.1. File format support

IAR Embedded Workbench uses the specific IAR assembly syntax. Assembly file formats supported:

- .s file: IAR-style assembly file format.

Meanwhile, it supports the use of inline assembly in C functions.

3.6.2. Assembly code format requirements

- Single assembly file: Here is a simple assembly code example, which defines an assembly function add, used to add two integers. The content of the file “add.s” is as follows:

```
SECTION .text:CODE      ; Define code section
PUBLIC add              ; Declare global symbol
```

```
add:
    ; Function entry
    ; Parameters: r0 and r1
    ; Return value: r0

    ADD r0, r0, r1          ; Add r0 and r1, store result in r0
    BX lr                    ; Return to calling function

    END
```

- Use inline assembly in C function: The following is an example of using inline assembly in C function, and an inline assembly function add_inline is defined to add two integers:

```
// Inline assembly function
static inline int add_inline(int a, int b) {
    int result;
    __asm volatile (
        "adds %0, %1, %2\n"
        : "=r" (result)            // Output operand
        : "r" (a), "r" (b)        // Input operands
        : "cc"                          // Clobbered registers
    );
    return result;

}
```

3.7. Linker Script Files

The linker script files are used to define the memory layout and section allocation of the program. In the migration process, it is necessary to use linker script files in the corresponding format according to different target platforms and development environments. G32R501 uses the linker script files in the ".icf" format in the IAR EW for Arm development environment, which comply with IAR Company's specifications.

Differences in linker script files

- File format:
  - The G32R501 uses linker script files in the ".icf" format, which comply with IAR Company's specifications.
  - Txx320F28004x uses the linker script files in ".CMD" format, which comply with the company's specifications.
- Memory layout and section allocation:
  - The ".icf" files of G32R501 allocate memory attributes by defining different "regions"

and their attributes.

- The ".CMD" file of Txx320F28004x arranges memory allocation by defining the Memory section (MEMORY) and section allocation (SECTIONS).

3.8. RAM Operation

Please refer to Chapter 2.7.

4. Eclipse

4.1. Emulator Support

- Geehy-Link (WinUSB), DAP Link (firmware version CMSIS-DAP V2 or above)
- J-Link V12 (J-Link V7.94g or above)

4.2. IDE Version

Ensure the use of Eclipse 4.35 or a newer version of the IDE.

4.3. LLVM_For_ARM_Toolchain

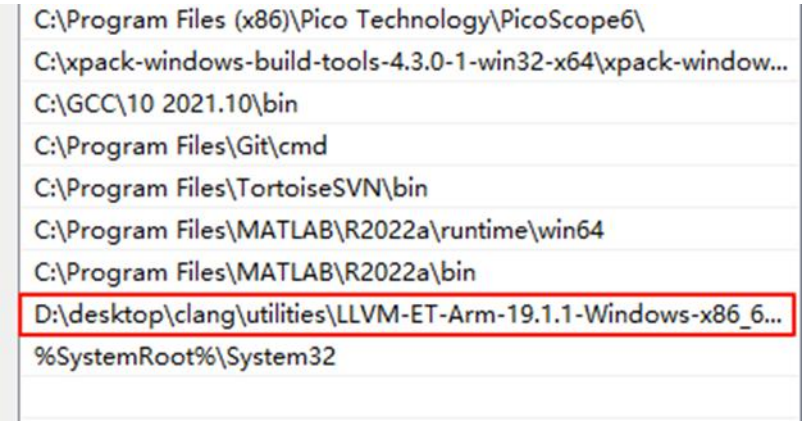
1. Download the LLVM_For_ARM_Toolchain compiler

- From the LLVM_For_ARM_Toolchain repository: https://github.com/ARM-software/LLVM-embedded-toolchain-for-Arm, download the LLVM-ET-Arm-19.1.1-Windows-x86_64 compiler.

2. Add Environment Variables

- Extract the downloaded compressed package (example extraction path: D:\desktop\clang\utilities\LLVM-ET-Arm-19.1.1-Windows-x86_64)
- Add the "bin" folder from the extracted directory to the system environment variable.

Figure 35 Add the system environment variable



4.4. GDB Service

It is recommended to use the arm-none-eabi-gdb.exe provided by Arm. The version of arm-none-eabi-gdb.exe used in the example is 14.2.

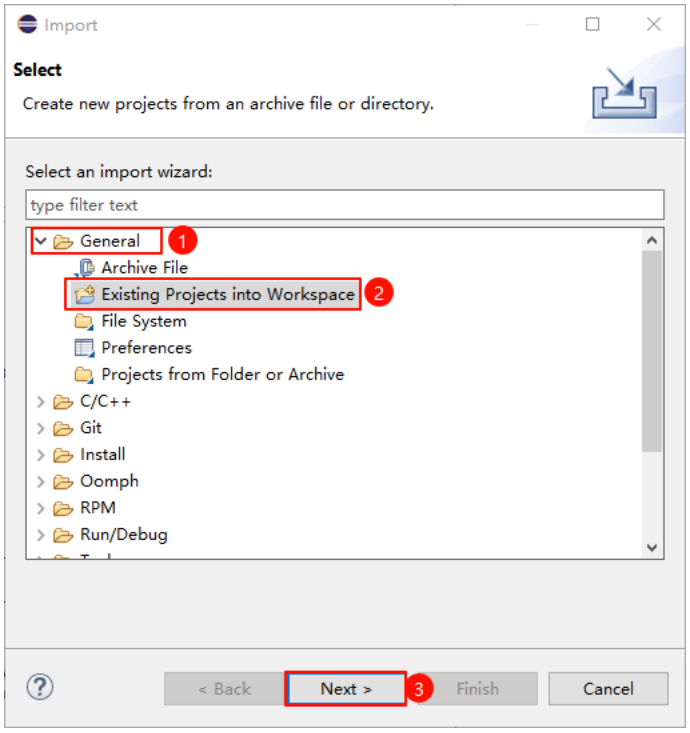
4.5. Project Operations

4.5.1. Opening the Example Project

1. Run Eclipse 4.35.

2. Click "File" -> "Import..." -> "General" -> "Existing Project into Workspace" in the menu bar.

Figure 36 Import project



3. Click "Select root directory", navigate to the path of the SDK project file you provided, select the corresponding project folder, and then click "Finish".

Note: Please complete the above steps after finishing the LLVM compilation configuration in section 4.3.

4.5.2. Project Creation

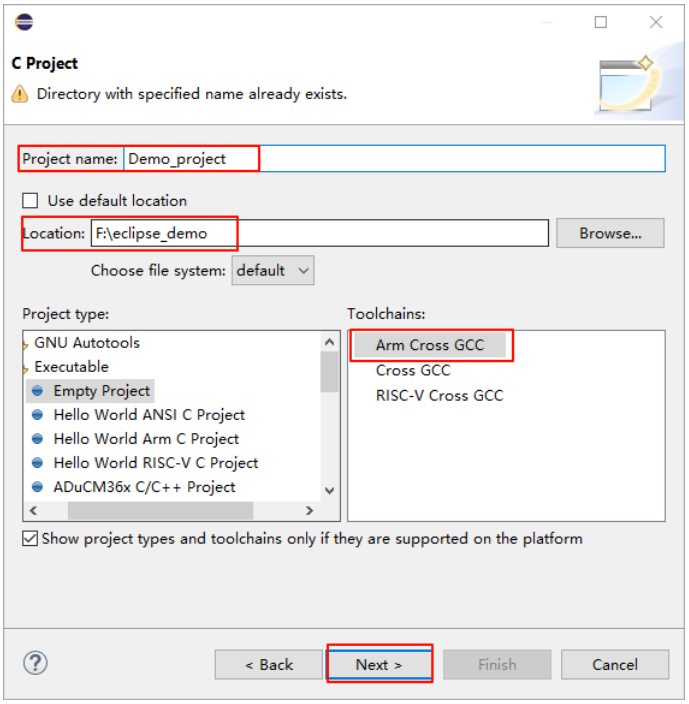
1. Run Eclipse:

- Open Eclipse 4.35.

2. Create a New Project:

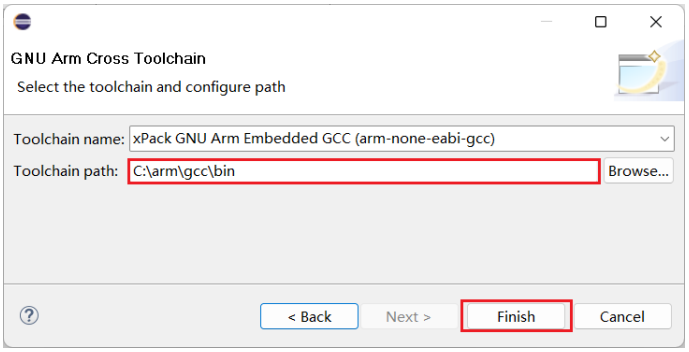
- Under "File->New," select to create a new C/C++ Project, and choose C Managed Build.
- Enter the project name, configure the project type, and it is recommended to place the project under the Project directory. Configure the toolchain, selecting Arm Cross GCC as the toolchain.

Figure 37 Configure project



- If the Arm Toolchains in the Eclipse IDE are correctly configured, the path will be automatically selected here. If not properly configured, you can click Browse to select the corresponding absolute path.

Figure 38 Set Arm Toolchains Path

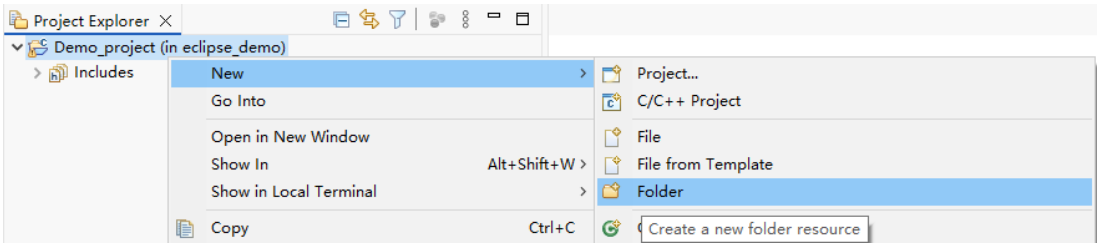


- Click “Finish”, completed setting Project.

4.5.3. File Import

1. Right-click on the project folder to create a virtual folder.

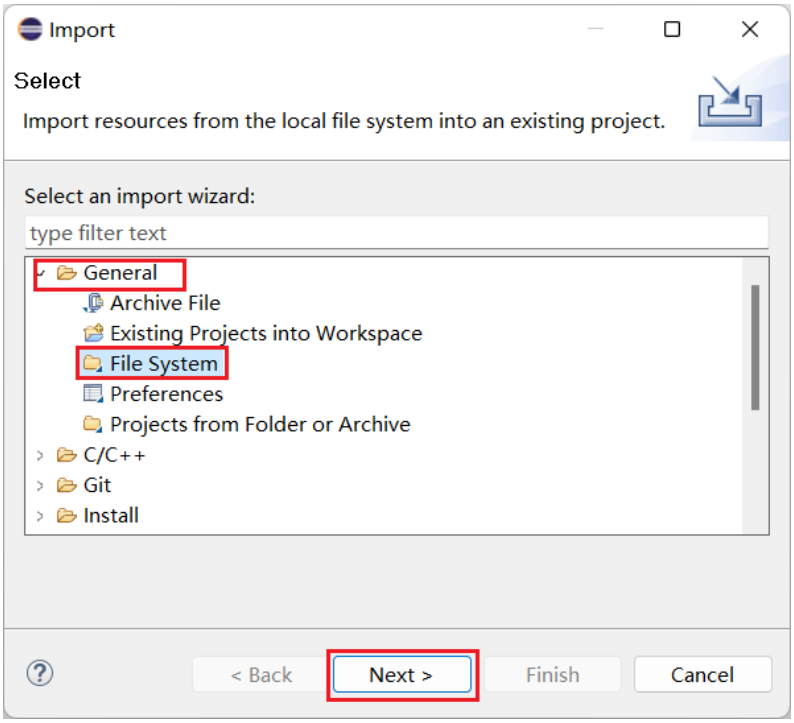
Figure 39 New file



2. Add files:

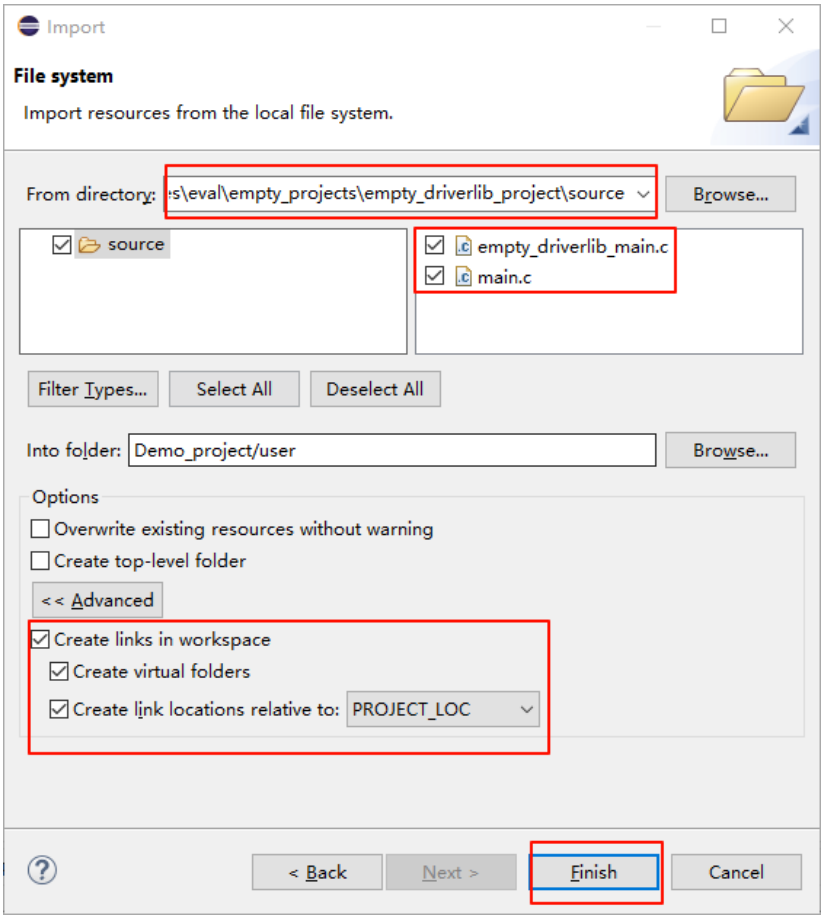
- Select the folder, right-click and choose the Import option to directly import files.

Figure 40 Import file



- When importing files, select "File System" as the import method. In the pop-up file path selection dialog, choose the path of the files to be imported, then check the files you need to import.

Figure 41 Select file



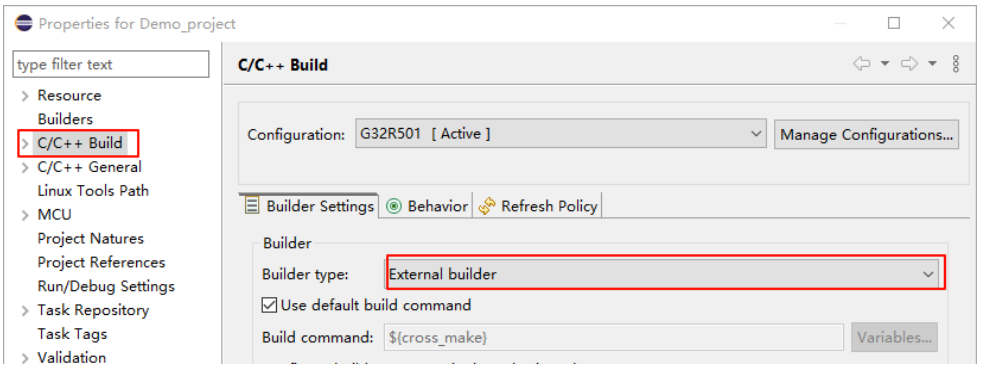
3. New file (If needed):

   - If you need to create a new source file, click "File" -> "New" and select the source file to create.
   - - Choose the folder and filename for the new source file.\

4.5.4. Compilation Configuration

1. Right-click on the project name and select "Properties".
2. Select the "External builder" configuration:

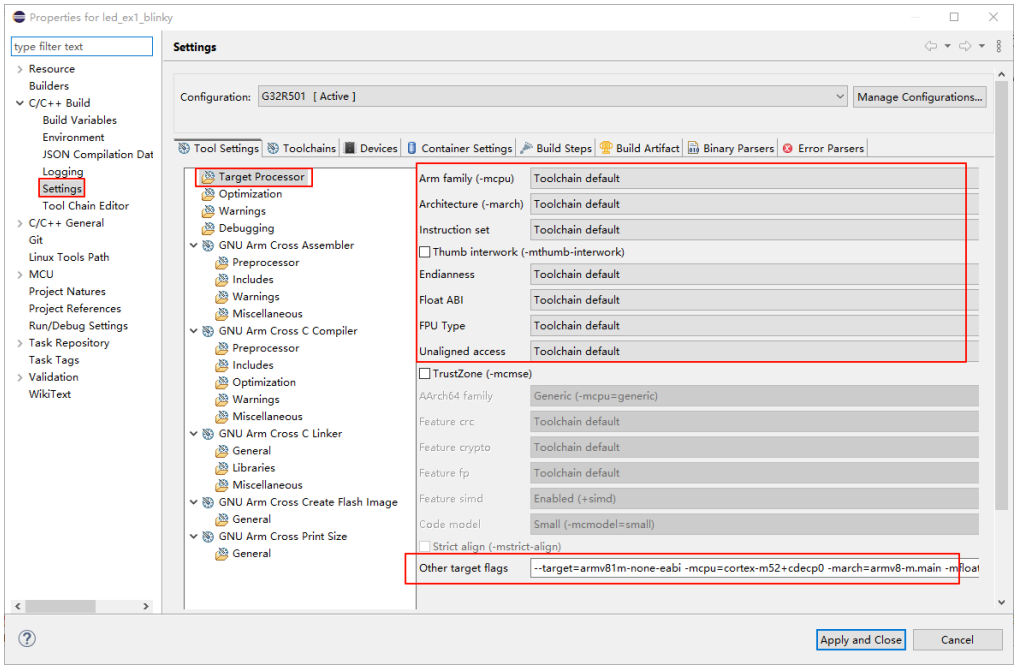
Figure 42 External builder



3. Under the "C/C++ Build" tab, select "Settings" -> "Tool Settings" -> "Target Processor"

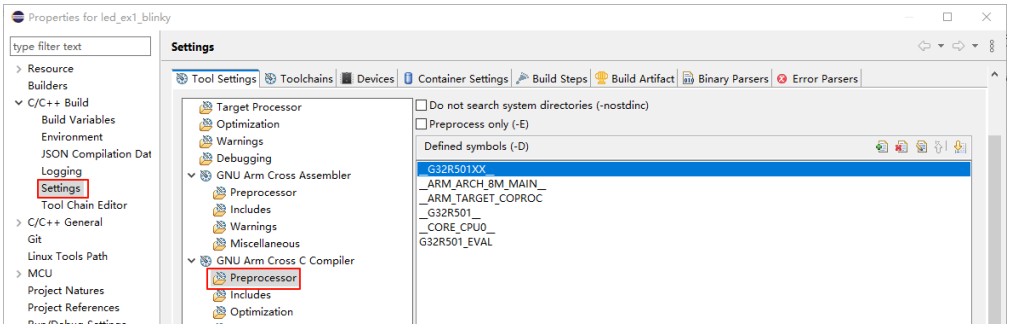
   - Set all dropdown configuration items to default, then manually add the command line: “--target=armv81m-none-eabi -mcpu=cortex-m52 -mfpu=none -fno-exceptions -fno-rtti -lcrt0-semihost –lsemihost” (These parameters can be modified according to the actual chip specifications.)

Figure 43 Target Processor



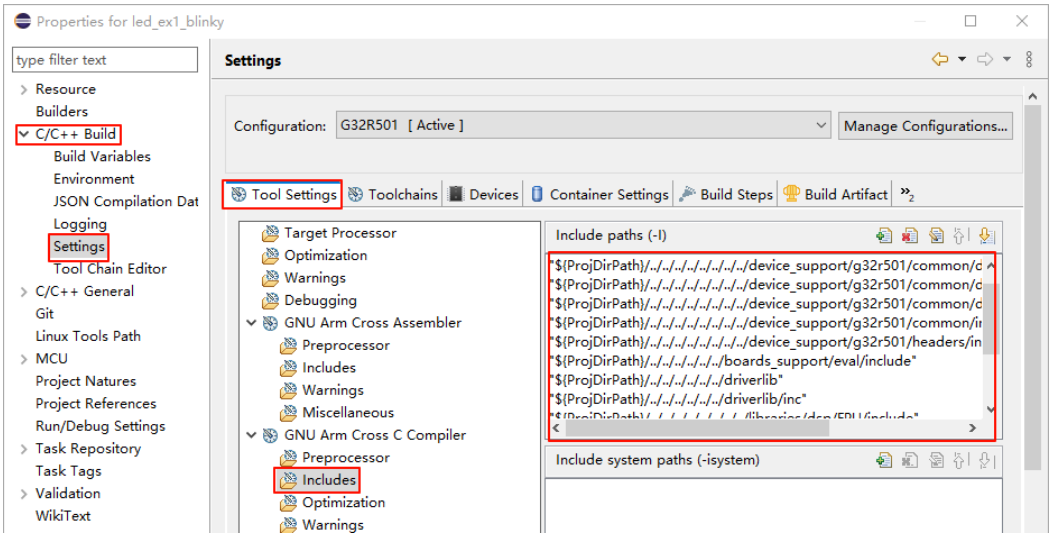
4. Under the "C/C++ Build" tab, select "Settings" -> "Tool Settings" -> "GNU Arm Cross C Compiler" -> "Preprocessor"

Figure 44 Add Macro Definitions



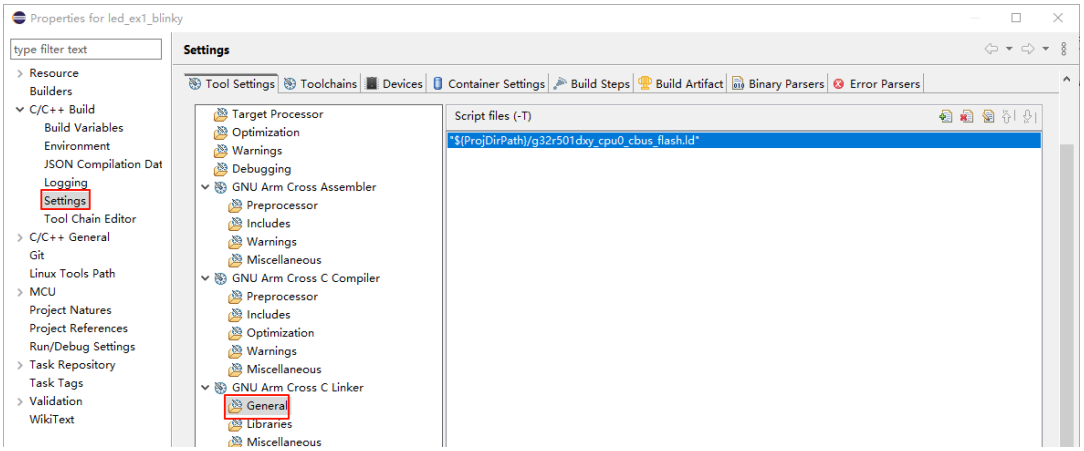
5. Under the "C/C++ Build" tab, select "Settings" -> "Tool Settings" -> "GNU Arm Cross C Compiler" -> "Includes".

Figure 45 Configure head file path



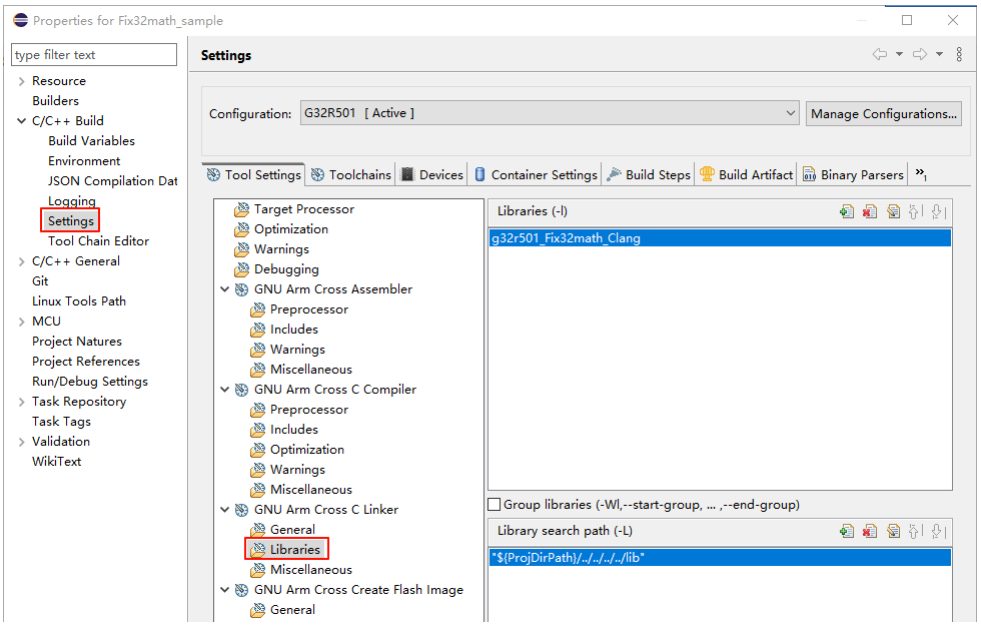
6. Under the "C/C++ Build" tab, select "Settings" -> "Tool Settings" -> "GNU Arm Cross C Linker" -> "General".

Figure 46 Add link file



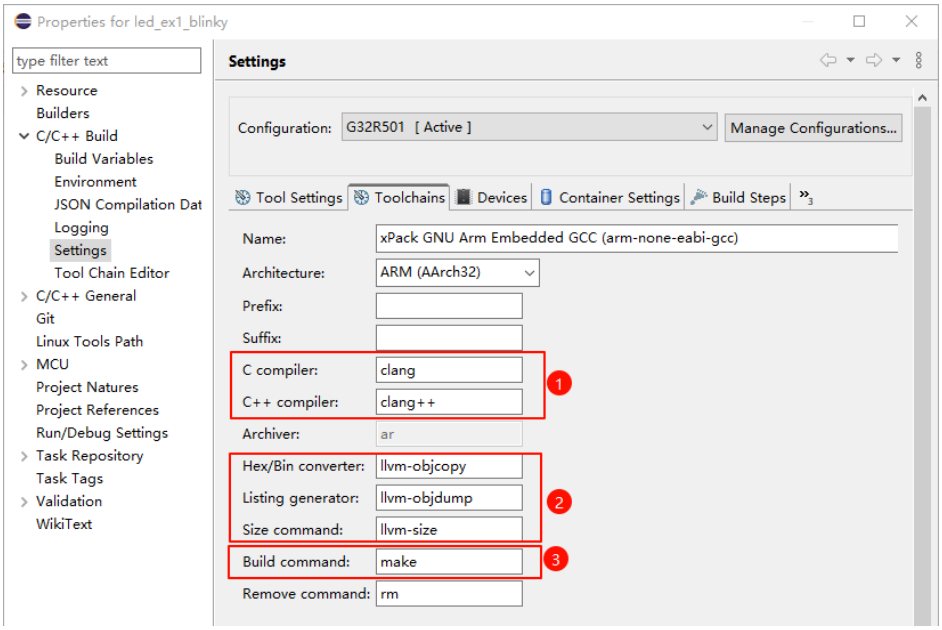
7. Under the "C/C++ Build" tab, select "Settings" -> "Tool Settings" -> "GNU Arm Cross C Linker" -> "Libraries".

Figure 47 Add libraries



8. Configure “ToolChain”.

Figure 48 Configure ToolChain



1) Use the built-in clang compiler from llvm.

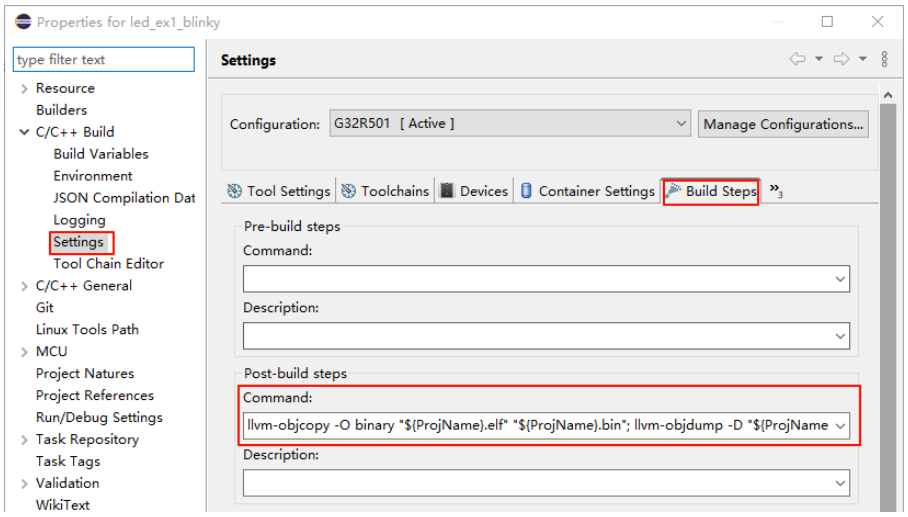
2) Built-in tools from the llvm toolchain.

3) Use the built-in make compiler from xpack.

9. Configure "Build Step":

   - Under the "C/C++ Build" tab, select "Settings" -> "Build Step" -> "Post-build steps" -> "Command".
   - Add the command: `llvm-objcopy -O binary "${ProjName}.elf" "${ProjName}.bin"; llvm-objdump -D "${ProjName}.elf" > "${ProjName}.dump"`.

Figure 49 Build Step

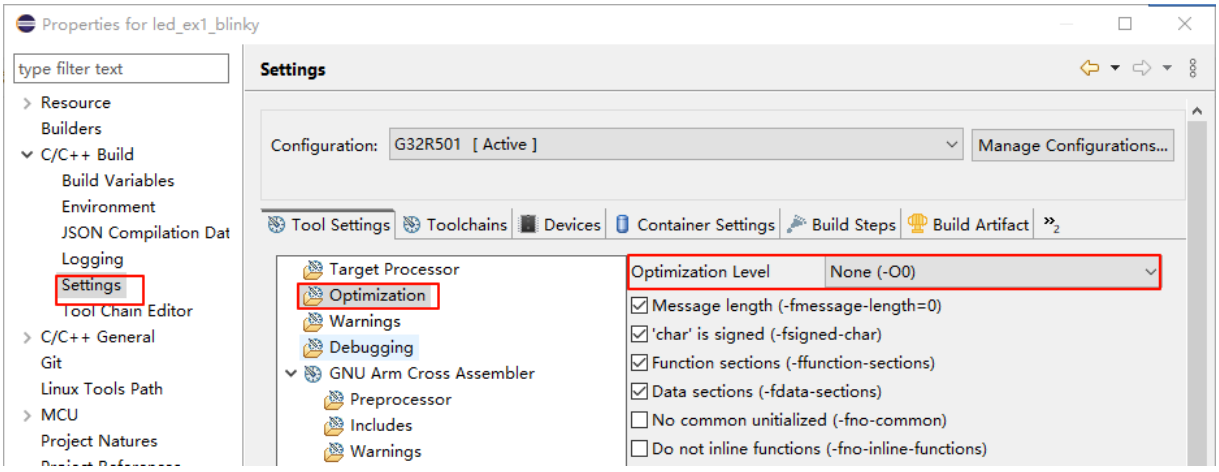


4.5.5. Compilation Optimization Level Settings

Eclipse provides multiple optimization level settings that can be adjusted at the global, single-file, and single-function levels:

1. Global Optimization Level Setting: Under the "C/C++ Build" tab, select "Settings" -> "Tool Settings" -> "Optimization" -> "Optimization Level".

Figure 50



2. Single-Function Optimization Level Setting: Specific compiler directives can be used before

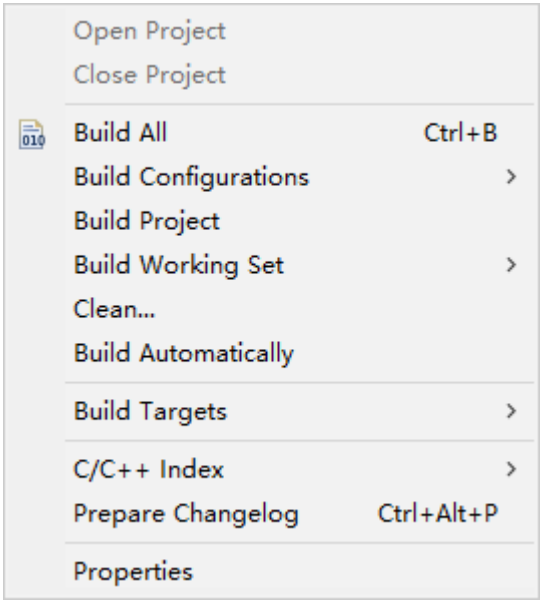
functions to set optimization levels, such as using “_Pragma("optimize=none")” to declare no optimization.

4.5.6. Program Compilation

1. Click on **Project**, then select **Build Project** to compile the current project.

Note: Using **Build Project** only compiles the current project, while **Build All** compiles all projects in the current workspace.

Figure 51 Program Compilation



Note: Before compiling, be sure to save the current project. Otherwise, the compilation may be based on the last saved version rather than the latest modifications. To ensure compilation accuracy, perform a **Clean** operation on the project after making changes, then proceed with compilation. After compilation is complete, corresponding.elf, .hex, and .bin files will be generated.

2. Wait for the compilation process to complete and check the output window for any error or warning messages. If errors are present, make the necessary corrections based on the prompts.

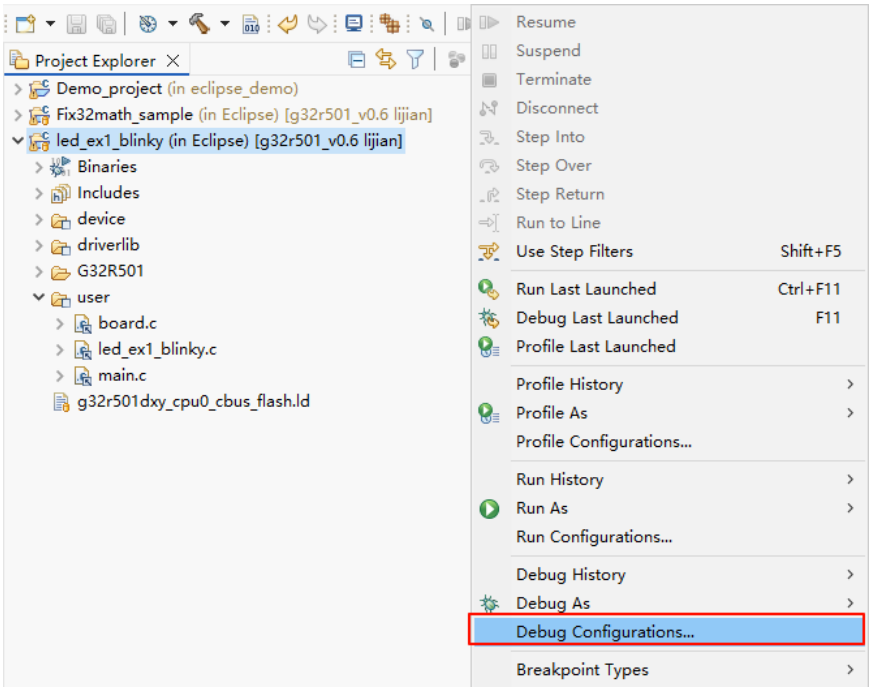
4.5.7. Program Simulation and Download

4.5.7.1. J-Link Simulation

1. Open the Debug Configuration Interface

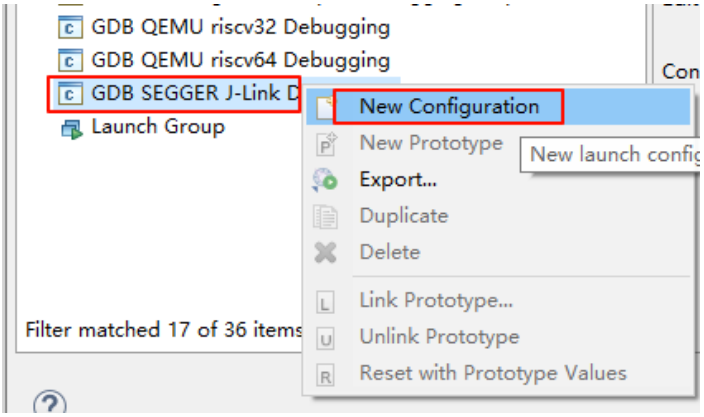
   - In the project, click **Run** in the menu bar, then select **Debug Configurations** from the pop-up options to enter the Debug configuration interface.

Figure 52 Debug configuration interface



- In the new window, select “GDB SEGGER J-Link Debugging”, right-click, and choose “New Configuration” to create a new simulation configuration.

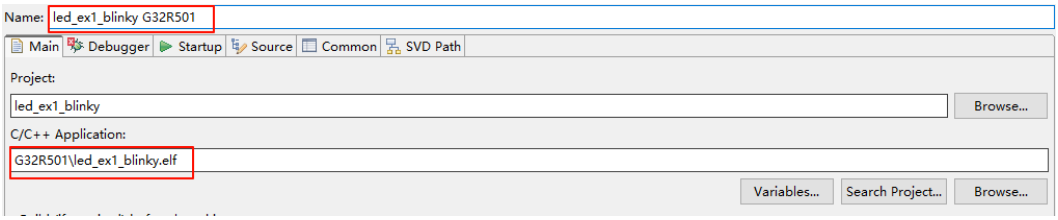
Figure 53 New simulation J-Link configuration



2. Configure the Main Tab

- Name the current simulation configuration at the top.
- Click **Browse...** to select the project corresponding to the current simulation configuration.
- Choose the corresponding simulation ELF file (e.g., `G32R501\led_ex1_blinky.elf`). Relative paths (relative to the project file) are used here, but absolute paths are also supported.

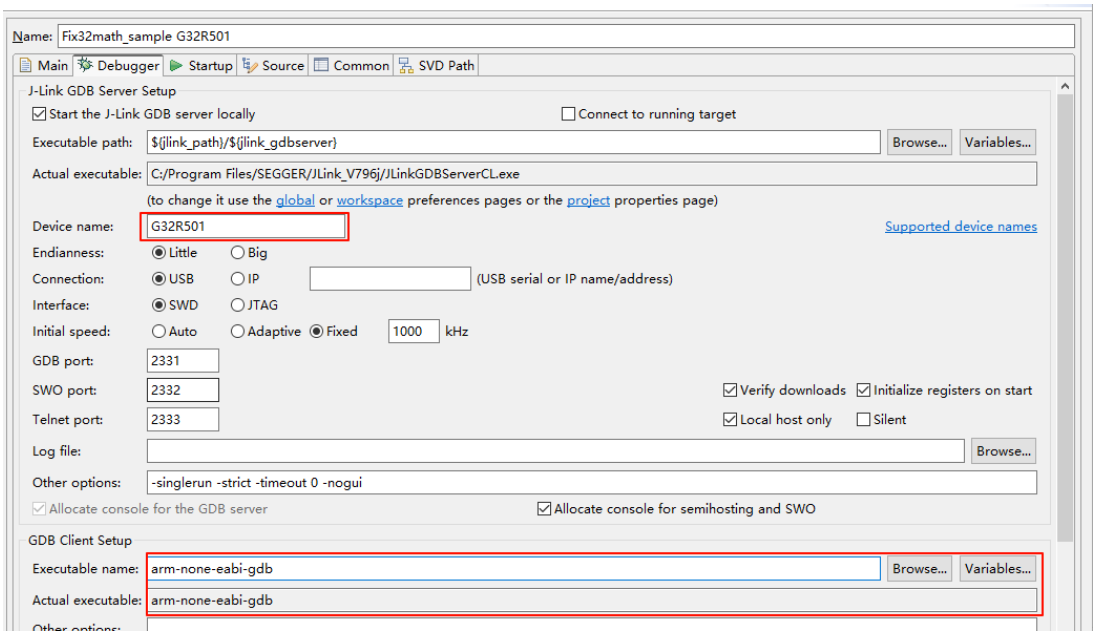
Figure 54 Configure Main



3. Configure the Debugger Tab

- Select the corresponding simulation chip (e.g., G32R501).
- Choose the GDB service—it is recommended to use the Arm-provided arm-none-eabi-gdb.exe.

Figure 55 Debugger Tab



4. Configure the Startup Tab

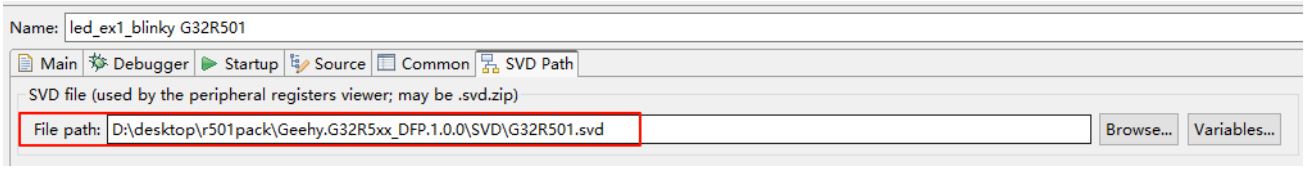
- Add **DCS KEY** (must match the chip-side configuration) in the **Initialization Commands** and **Run/Restart Commands** fields.
- The content added in both fields should be identical.

```
set {unsigned int}0x50024020 = 0xFFFFFFFF

set {unsigned int}0x50024024 = 0xFFFFFFDC

set {unsigned int}0x50024028 = 0xFFFFFFFF

set {unsigned int}0x5002402C = 0xFFFFFFFF
```

```
set {unsigned int}0x500240A0 = 0xFFFFFFFF
set {unsigned int}0x500240A4 = 0xFFFEDFFF
set {unsigned int}0x500240A8 = 0xFFFFFFFF
set {unsigned int}0x500240AC = 0xFFFFFFFF
set $t0 = *(unsigned int *)0x08000000
set $sp=$t0
set $t1 = *(unsigned int *)0x08000004
set $pc=$t1
set $xpsr=$xpsr|(1<<24)
```

5. Configure the SVD Tab

Figure 56 Configure SVD path



6. Finally, click the **Apply** button at the bottom right of the tab to save all configurations.

4.5.7.2. GEEHY LINK Simulation

Use Geehy-Link (WinUSB) to download and debug the project, and use pyocd to download and simulate the project. For details, refer to pyocd 适配 G32R501.

4.6. C Language Compatibility

Refer to Section 2 "C Language Compatibility".

4.7. Assembly Compatibility

The instruction sets and assembly syntax of different target platforms may vary significantly. Existing assembly code may need to be rewritten and adapted to ensure correct operation on new platforms.

4.7.1. File Format Support

Eclipse uses a specific GCC assembly syntax. The assembly syntax of GCC (GNU Assembler, abbreviated as GAS) is consistent with MDK (Keil's ARM assembler, i.e., ARMASM) in terms of core instruction sets, but there are significant differences in pseudo-instructions, syntax formats,

and symbol definitions. Below is a detailed comparison.

- .S files: GCC-style assembly file format.

Additionally, it supports inline assembly within C functions.

4.7.2. Assembly Coding Format Requirements

- Single assembly file: Here is a simple example of assembly code that defines an assembly function `add` to add two integers. The content of the file "add.s" is as follows:

```
.section .text    //Define code section
.global add     // Declare global symbol


add:
    //Function entry
    //Parameters: r0 and r1
    //Return value: r0
    add r0, r0, r1     // Add r0 and r1, store result in r0
    bx  lr          // Return to calling function
```

- Using inline assembly in C functions: The following is an example of using inline assembly in a C function, defining an inline assembly function add_inline to add two integers:

```
// Inline assembly function
static inline int add_inline(int a, int b) {
    int result;
    asm volatile (
        "ADD %0, %1, %2"
        : "=r" (result)
        : "r" (a), "r" (b)
        :
    );
    return result;

}
```

4.8. Linker Script Files

Linker script files are used to define the memory layout and section allocation of a program. During the migration process, it is necessary to use linker script files in the appropriate format according to the target platform and development environment. The G32R501 uses ".ld" format linker script files under the Eclipse development environment, adhering to Eclipse's

specifications.

Differences in Linker Script Files

- File Format:
  - The G32R501 uses ".ld" format linker script files.
  - The Txx320F28004x uses ".CMD" format linker script files, following its company's specifications.
- Memory Layout and Section Allocation:
  - The ".ld" file of the G32R501 allocates memory attributes by defining different "MEMORY" and its "SECTIONS."
  - The ".CMD" file of the Txx320F28004x arranges memory allocation by defining memory segments (MEMORY) and section allocations (SECTIONS).

4.9. RAM Operating

Refer to Chapter 2.7.

5. pyocd Adaptation for G32R501

5.1. Background

To enable customers to perform operations such as program downloading and debugging for the G32R501 in an open-source environment, the G32R501 series MCUs need to support PyOCD.

5.2. PyOCD Adaptation Modifications

The current PyOCD (https://github.com/pyocd/pyOCD/releases/tag/v0.36.0) release version is 0.36, which does not support the M52 core chips or the G32R501 chip. Source code modifications are required to enable support.

5.2.1. Adding M52 Core Support

1. Add M52 core support in PyOCD. The main files to be modified are:

   - pyocd\coresight\component_ids.py

   Under class CmpInfo(NamedTuple):, add:

```
# Designer          |Component Class |Part    |Type |Archid                  |Name            |Product
|Factory

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x31, 0x0a31) : CmpInfo('MTB',          'Star-
MC2', None                    ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x43, 0x1a01) : CmpInfo('ITM',          'Star-
MC2', ITM.factory             ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x1a02) : CmpInfo('DWT',          'Star-
MC2', DWTv2.factory           ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x1a03) : CmpInfo('BPU',          'Star-
MC2', FPB.factory             ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x14, 0x1a14) : CmpInfo('CTI',          'Star-
MC2', None                    ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x2a04) : CmpInfo('SCS',          'Star-
MC2', CortexM_v8M.factory ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x13, 0x4a13) : CmpInfo('ETM',          'Star-
MC2', None                    ),

(ARM_CHINA_ID, CORESIGHT_CLASS, 0x132, 0x11, 0)      : CmpInfo('TPIU',         'Star-
MC2', TPIU.factory            ),
```

Figure 57 Modified component_ids.py

```
## Map from (designer, class, part, devtype, archid) to component name, product name, and factory.
COMPONENT_MAP: Dict[Tuple[int, int, Optional[int], Optional[int], int], CmpInfo] = {
  # Archid-only entries
     # Designer      |Component Class |Part  |Type |Archid          |Name         |Product    |Factory
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x31, 0x0a31) : CmpInfo('MTB',       'Star-MC2', None                ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x43, 0x1a01) : CmpInfo('ITM',       'Star-MC2', ITM.factory         ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x1a02) : CmpInfo('DWT',       'Star-MC2', DWTv2.factory       ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x1a03) : CmpInfo('BPU',       'Star-MC2', FPB.factory         ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x14, 0x1a14) : CmpInfo('CTI',       'Star-MC2', None                ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x00, 0x2a04) : CmpInfo('SCS',       'Star-MC2', CortexM_v8M.factory ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0xD24, 0x13, 0x4a13) : CmpInfo('ETM',       'Star-MC2', None                ),
    (ARM_CHINA_ID, CORESIGHT_CLASS, 0x132, 0x11, 0)      : CmpInfo('TPIU',      'Star-MC2', TPIU.factory        ),
  # Designer|Component Class |Part  |Type |Archid          |Name          |Product    |Factory
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x0a00) : CmpInfo('RASv1',          None,       None                ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x1a01) : CmpInfo('ITMv2',          None,       ITM.factory         ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x1a02) : CmpInfo('DWTv2',          None,       DWTv2.factory       ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x1a03) : CmpInfo('FPBv2',          None,       FPB.factory         ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x2a04) : CmpInfo('v8-M Debug',     None,       CortexM_v8M.factory ),
    (ARM_ID, CORESIGHT_CLASS, None,  None, 0x6a05) : CmpInfo('v8-R Debug',     None,       None                ),
```

- pyocd\coresight\core_ids.py

  1) Add the core ID after # CPUID PARTNO values

ARM_China_StarMC2 = 0xD24

  2) Add the core name after CORE_TYPE_NAME: Dict[Tuple[int, int], str]

(CPUID_ARM_CHINA, ARM_China_StarMC2): "Star-MC2",

Figure 58 Modiffied core_ids.py

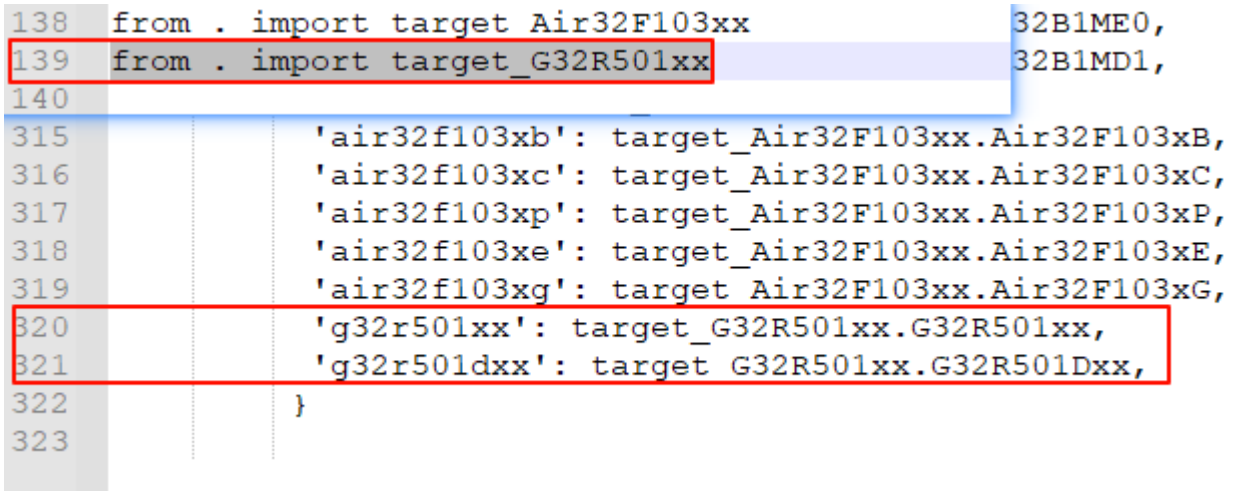
```
ARM_CortexM55 = 0xD22
ARM_CortexM85 = 0xD23
ARM_China_StarMC1 = 0x132
ARM_China_StarMC2 = 0xD24

# pylint: enable=invalid_name

## @brief User-friendly names for core types.
CORE_TYPE_NAME: Dict[Tuple[int, int], str] = {
        (CPUID_ARM,         ARM_SC000):         "SecurCore SC000",
        (CPUID_ARM,         ARM_SC300):         "SecurCore SC300",
        (CPUID_ARM,         ARM_CortexM0):      "Cortex-M0",
        (CPUID_ARM,         ARM_CortexM1):      "Cortex-M1",
        (CPUID_ARM,         ARM_CortexM3):      "Cortex-M3",
        (CPUID_ARM,         ARM_CortexM4):      "Cortex-M4",
        (CPUID_ARM,         ARM_CortexM7):      "Cortex-M7",
        (CPUID_ARM,         ARM_CortexM0p):     "Cortex-M0+",
        (CPUID_ARM,         ARM_CortexM23):     "Cortex-M23",
        (CPUID_ARM,         ARM_CortexM33):     "Cortex-M33",
        (CPUID_ARM,         ARM_CortexM35P):    "Cortex-M35P",
        (CPUID_ARM,         ARM_CortexM55):     "Cortex-M55",
        (CPUID_ARM,         ARM_CortexM85):     "Cortex-M85",
        (CPUID_ARM_CHINA,   ARM_China_StarMC1): "Star-MC1",
        (CPUID_ARM_CHINA,   ARM_China_StarMC2): "Star-MC2",
    }
```

5.2.2. Add G32R501 Chip Support

1. Add the G32R501 download algorithm support file in pyocd\target\builtin: target_G32R501xx.py. This file has been added in SDK/device_support/g32r501/common/pyOCD/target_G32R501xx.py.
2. To add G32R501 support, include the following in pyocd\target\builtin__init__.py:

```
from . import target_G32R501xx

'g32r501dxx': target_G32R501xx.G32R501Dxx,
'g32r501xx': target_G32R501xx.G32R501xx,
```

Figure 59 Modified __init__.py



5.2.3. pyocd_user.py

pyocd supports parsing the corresponding script using "--script=<path>" to add custom commands and additional operations before or after connection.

The pyocd_user.py file has been added in SDK/device_support/g32r501/common/pyOCD/target_G32R501xx.py.

5.2.3.1. DCS Key

Due to the features of the G32R501, the chip includes DCS functionality, requiring the transmission of a key to the aforementioned target_G32R501xx.py during the connection process for use during the download operation.

If users need to modify the DCS key, they can directly edit NEW_DECRYPT_KEYS:

```
NEW_DECRYPT_KEYS = [
    (0x50024020, 0xFFFFFFFF),
    (0x50024024, 0xFFFFFFDC),
    (0x50024028, 0xFFFFFFFF),
    (0x5002402C, 0xFFFFFFFF),
    (0x500240A0, 0xFFFFFFFF),
    (0x500240A4, 0xFFFEDFFF),
    (0x500240A8, 0xFFFFFFFF),
    (0x500240AC, 0xFFFFFFFF),
]
```

Note: The front part is the write address, and the back part is the DCS KEY content to be written.

5.2.3.2. Connection Phase

The connection phase requires the following steps:

1. Perform DCS decryption
2. Initialize the CPU

The above operations are implemented in “def did_connect(board) -> None:”.

5.3. pyocd Installation

5.3.1. Windows

5.3.1.1. Install Python

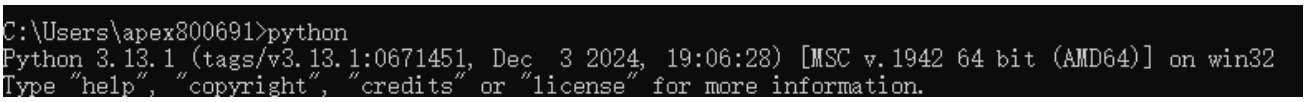
pyocd 支持需要在 python 环境下，请至 python 官网（https://www.python.org）下载最新的

python 安装包进行安装。

注意：安装完成后，请确保将 Python 添加到系统的 PATH 环境变量中，以便在命令行中使用。

验证：使用 Win+R 键，输入 CMD，在命令行窗口上输入：python，然后回车。显示已安装的 Python 版本号等内容，并进入 Python 的交互式命令行（REPL）中。如需退出输入 exit() 或 quit()，然后按下回车键。

Figure 60 Python install check



5.3.1.2. Install pyocd

pyocd is a component package for Python that supports online installation (online installation is recommended due to various dependency packages that can be resolved during the process).

The installation method is as follows:

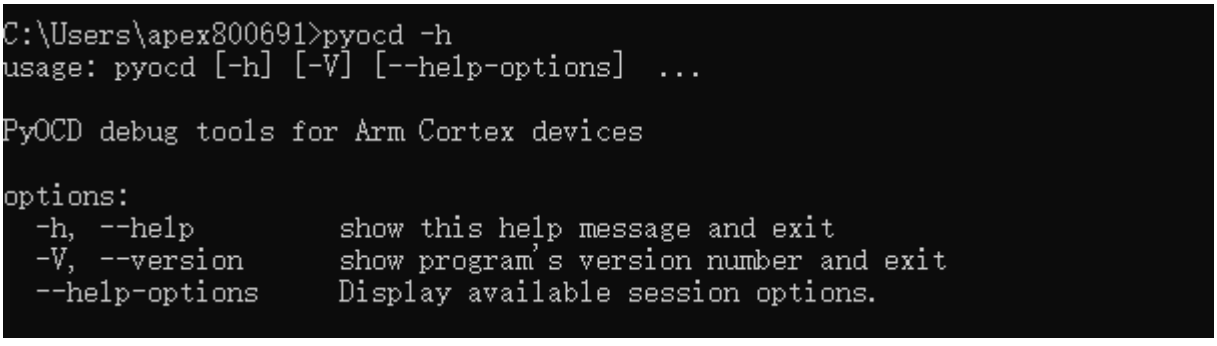
1. Install using the pip command:

Figure 61 Install pyocd

Note: After installation, add PyOCD's installation path to the system's PATH environment variable for command-line usage. For example:C:\Users\Geehy\AppData\Local\Programs\Python\Python313\Scripts

2. Verify the installation by pressing **Win + R**, entering **CMD**, and running the following command in the terminal: pyocd -h, this will display the command help.

Figure 62 pyocd install check



5.3.2. Ubuntu

5.3.2.1. Installing Python

1. Ubuntu usually comes with Python pre-installed. You can check the Python and pip versions by running the following command in the terminal:

```
python --version
```

2. If Python or pip is not installed, use the following command to install them:

```
sudo apt update
sudo apt install python3 python3-pip
```

5.3.2.2. python3-venv

Some Ubuntu systems come with externally managed Python3 environments, which prevent direct installation of packages in the global environment using pip. To avoid this issue, you can use Python's built-in venv module to create a virtual environment.

1. Use the following command to install the python3-venv package:

```
sudo apt install python3-venv
```

2. Use the following command to create a virtual environment (assuming you name it "venv"):

```
python3 -m venv venv
```

3. Use the following command to activate the virtual environment:

   - Activate the virtual environment to install packages within it.

```
source venv/bin/activate
```

   - To deactivate the virtual environment later, use the following command:

```
deactivate
```

5.3.2.3. Install pyocd

1. In the activated virtual environment, install pyOCD using pip:

```
pip install pyocd
```

2. Verify the installation results

```
pyocd --version
```

3. Query the installation location of pyocd (for subsequent modification of pyocd source code)

```
pip show pyocd
```

5.3.2.4. USB Permissions for pyocd

If pyocd cannot access the debugger under a regular user, consider adding appropriate permissions for the current user. Use udev rules to ensure USB device access without sudo. Follow these steps:

1. Create a new rule file by executing the following command in the terminal to create a new udev rule file (e.g., named 99-pyocd.rules):

```
sudo nano /etc/udev/rules.d/99-pyocd.rules
```

2. Add the following content to the file (the Geehy-Link device ID is 314B):

```
SUBSYSTEM=="usb", ATTR{idVendor}=="314b", MODE="0666"
```

In the nano editor, press Ctrl + O to save the file, then press **Enter** to confirm. Next, press Ctrl + X to exit the editor.

3. After saving the file, reload the udev rules.

```
sudo udevadm control --reload-rules

sudo udevadm trigger
```

4. Verify whether pyocd can correctly recognize Geehy-Link.

```
pyocd list
```

Figure 63 Serial number of the emulator connected to Ubuntu

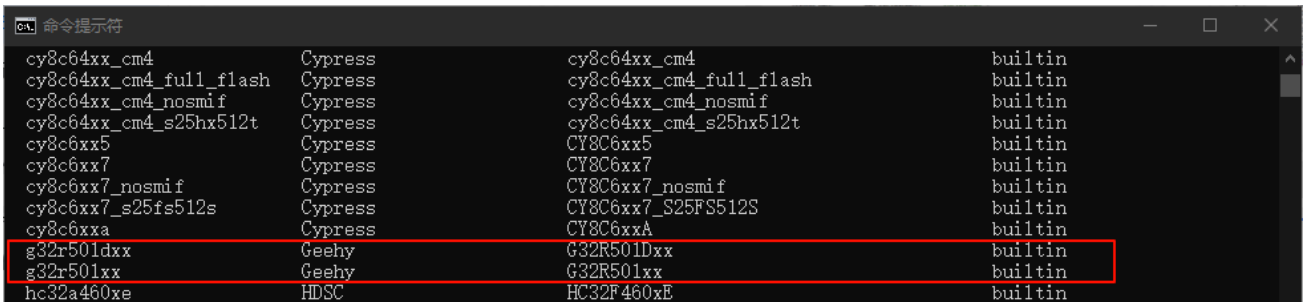
```
(venv) kai@Ubuntu:~$ pyocd list
  #   Probe/Board             Unique ID                          Target
------------------------------------------------------------------------
  0   Geehy CMSIS-DAP WinUSB  00350043500000144e544859000258     n/a
```

5.3.3. Modify Replacement Item Content

To support G32R501, please refer to Chapter 5.2 to modify the downloaded pyocd content.

To verify whether g32r501 support has been successfully added, press Win+R, enter CMD, and in the command prompt window, input: “pyocd list –targets”. This will display all supported chips. Check if "g32r501xx" is listed among them.

Figure 64 Support chips list



5.4. Command Line Usage

pyocd supports operation via the CMD command line. The following steps outline the usage (ensure pyocd has been added to PATH before use):

1. Connect the board or debugger to the chip and then to the PC.
2. In the working directory, add a `pyocd.yaml` configuration file. This file specifies the default target chip, connection method, and other settings. Users can refer to the official documentation: https://pyocd.io/docs/configuration.html.

For the G32R501, the reference `pyocd.yaml` configuration file is located at `SDK/device_support/g32r501/common/pyOCD/target_G32R501xx.py`.

3. In the working directory, add a `pyocd_user.py` file. Refer to Chapter 2.3 for its content.
4. Open CMD in the working directory and enter: `pyocd commander`. You can then input relevant commands in the command window to perform corresponding operations.

Figure 65 pyocd commander

```
 #   Probe/Board             Unique ID                         Target
-------------------------------------------------------------------------
 0   Geehy CMSIS-DAP WinUSB   00330057500000144e544859000258   n/a

 1   Geehy CMSIS-DAP WinUSB   00480051500000054e544859000258   n/a

nter the number of the debug probe or 'q' to quit> 1
010556 W Invalid coresight component, cidr=0x0 [rom_table]
onnected to G32R501Dxx [Halted]: 00480051500000054e544859000258
yocd> erase
yocd> rw 0x08000000 0x10
8000000:  ffffffff ffffffff ffffffff ffffffff    |................|
yocd>
```

5.5. Integration with Eclipse

Currently, Eclipse+pyocd has version requirements for Eclipse. It is recommended to use Eclipse 202503.

Additionally, it is advised to configure the pyocd path globally in Eclipse (as some computers may encounter issues with Eclipse retrieving the PATH). The steps are as follows:

1. Right-click on the "Windows" menu bar to display all configurations.
2. Select "Preference" from the displayed configurations.
3. In the new window, navigate to the sub-options under the "MCU" option.
4. Select the sub-option "Global pyOCD Path."
5. On the right side, choose the directory where the corresponding pyocd.exe is located.
6. Finally, click "Apply and Close."

Figure 66 Global pyOCD Path Step 1-2

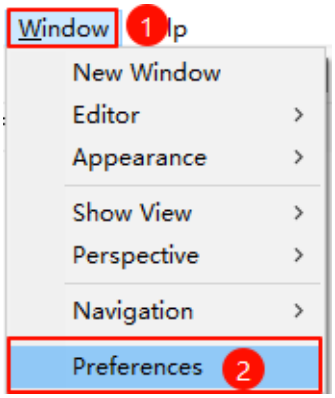
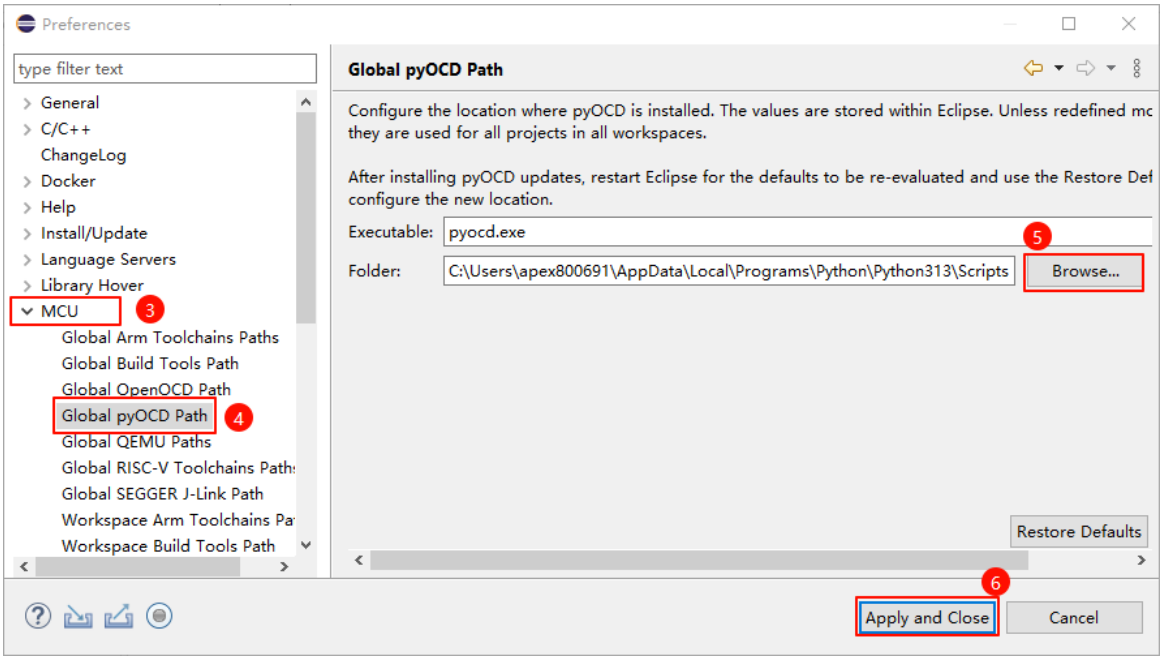


Figure 67 Global pyOCD Path Step 3-6



5.5.1. Single-core Simulation Configuration

After importing the project into Eclipse and ensuring it compiles without errors, follow the steps below to configure the simulation tab.

1. Create a New Simulation Configuration

   1) Left-click the Debug icon to display the Debug configurations.

   2) Select the displayed "Debug Configurations..." option.

   3) In the new window, choose "GDB PyOCD Debugging" and right-click.

   4) Select "New Configuration" to proceed with the simulation configuration.

Figure 68 New Configuration Step1-2

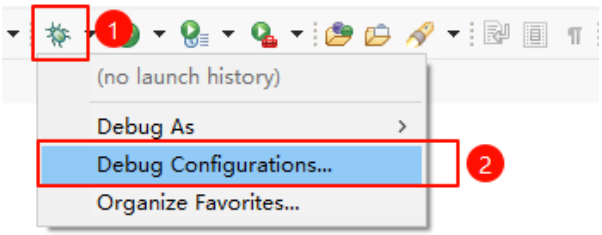
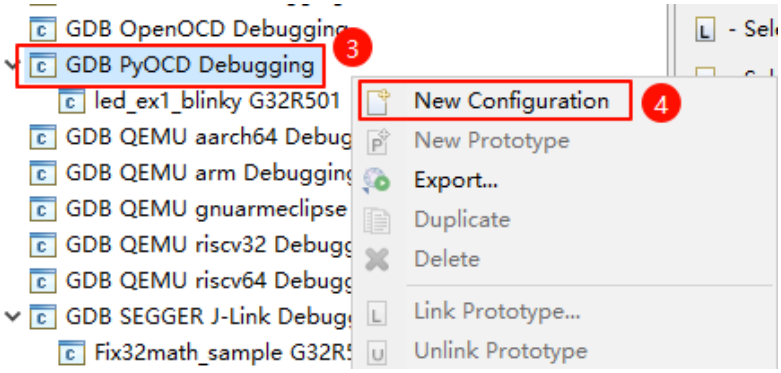


Figure 69 New Configuration Step 3-4



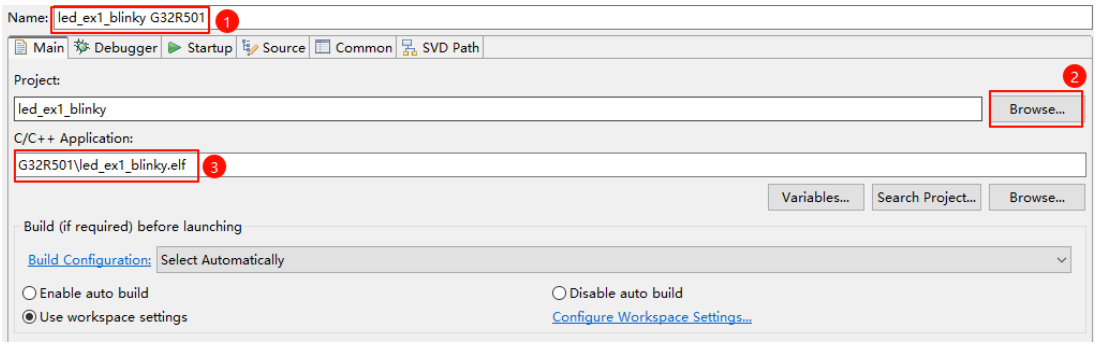
2. Configure the Main Tab

   1) Name the current simulation configuration at the top.

   2) Select "Browse..." to choose the project corresponding to the current simulation configuration.

   3) Select the corresponding simulation ELF file, e.g., `G32R501\led_ex1_blinky.elf`. The example uses a relative path to the project file, but absolute paths are also supported.

Figure 70 Configure Main



3. Configure the Debugger Tab

   1) Select the Geehy-Link emulator to be used. The characters in parentheses represent the emulator's serial number.

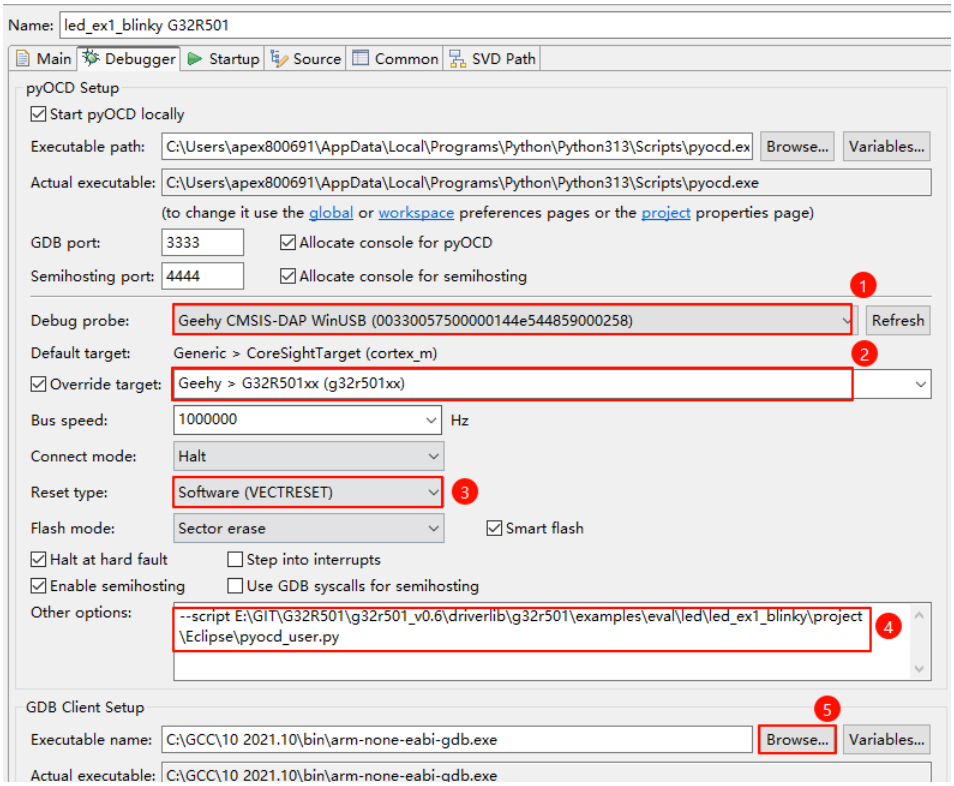
   2) Choose the corresponding simulation chip, e.g., `Geehy > G32R501xx (g32r501xx)`.

   3) For the reset mode, select "Software (SYSRESETREQ)."

   4) For the configuration file, select the previously prepared `pyocd_user.py`. It is recommended to use an absolute path without Chinese characters or spaces.

   5) For the GDB service, it is recommended to use the Arm-provided `arm-gnu-toolchain-14.2.rel1\bin\arm-none-eabi-gdb.exe`.

Figure 71



4. Configure the Startup Tab

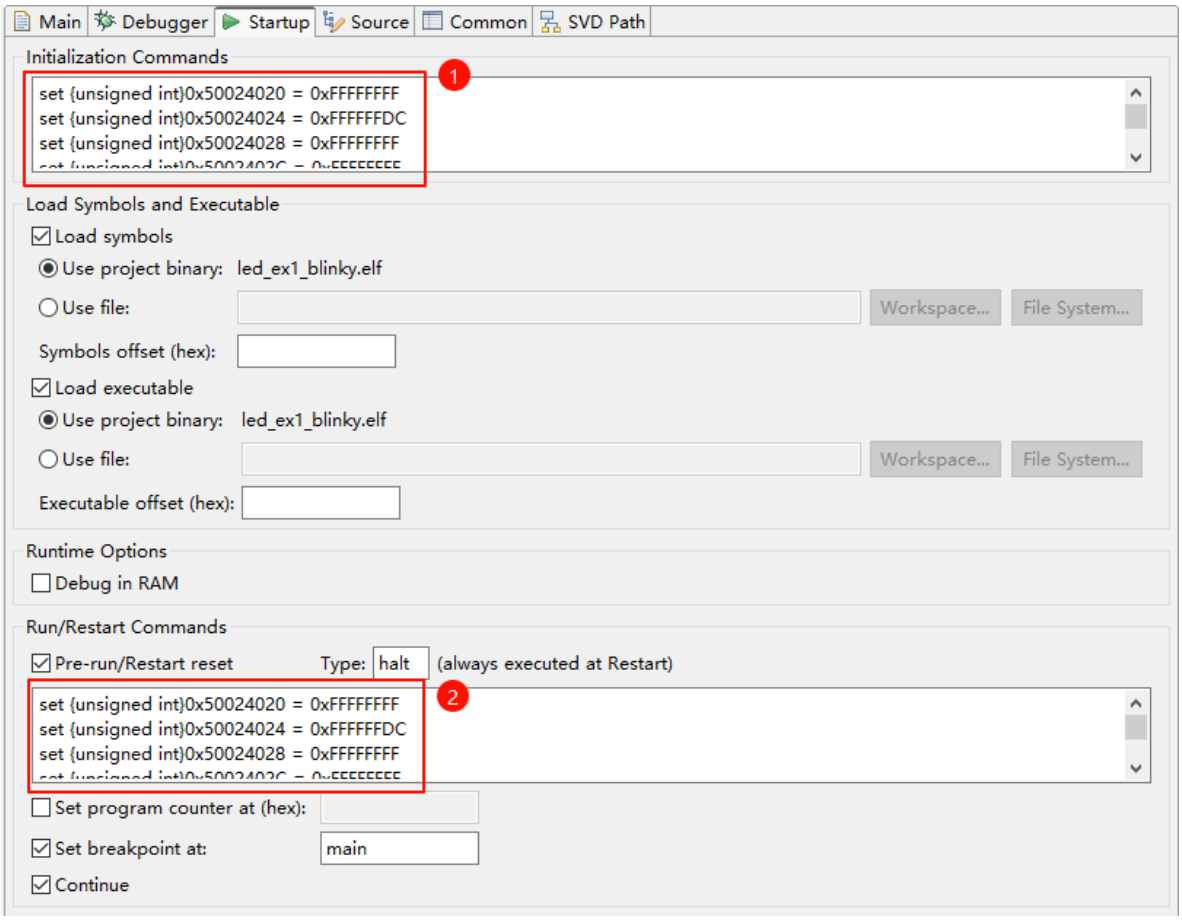
   1) Add the DCS KEY in the Initialization Commands and Run/Restart Commands sections (this needs to correspond with pyocd_user.py and the chip side).

   2) The content added in both sections should be consistent. Refer to the following:

```
set {unsigned int}0x50024020 = 0xFFFFFFFF

set {unsigned int}0x50024024 = 0xFFFFFFDC

set {unsigned int}0x50024028 = 0xFFFFFFFF

set {unsigned int}0x5002402C = 0xFFFFFFFF

set {unsigned int}0x500240A0 = 0xFFFFFFFF

set {unsigned int}0x500240A4 = 0xFFFEDFFF

set {unsigned int}0x500240A8 = 0xFFFFFFFF

set {unsigned int}0x500240AC = 0xFFFFFFFF

set $t0 = *(unsigned int *)0x08000000

set $sp=$t0
```

```
set $t1 = *(unsigned int *)0x08000004

set $pc=$t1

set $xpsr=$xpsr|(1<<24)
```

Figure 72 Configure Startup



5. Finally, click the "Apply" button at the bottom right corner of the tab to apply all configuration items.

5.5.2. Dual-Core Emulation Configuration

For dual-core emulation configuration, please refer to the 《AN1128_G32R501 Dual-Core Emulation Guide》.

5.5.3. Program Fails to Run After Exiting Emulation in Ubuntu

5.5.3.1. Cause

In Ubuntu, Eclipse terminates the current thread before arm-none-eabi-gdb sends "resuming

core 0 [cortex_m]", preventing the chip from resetting properly.

5.5.3.2. Solution

Refer to the dual-core emulation configuration and start the pyOCD gdbserver solution in the terminal.

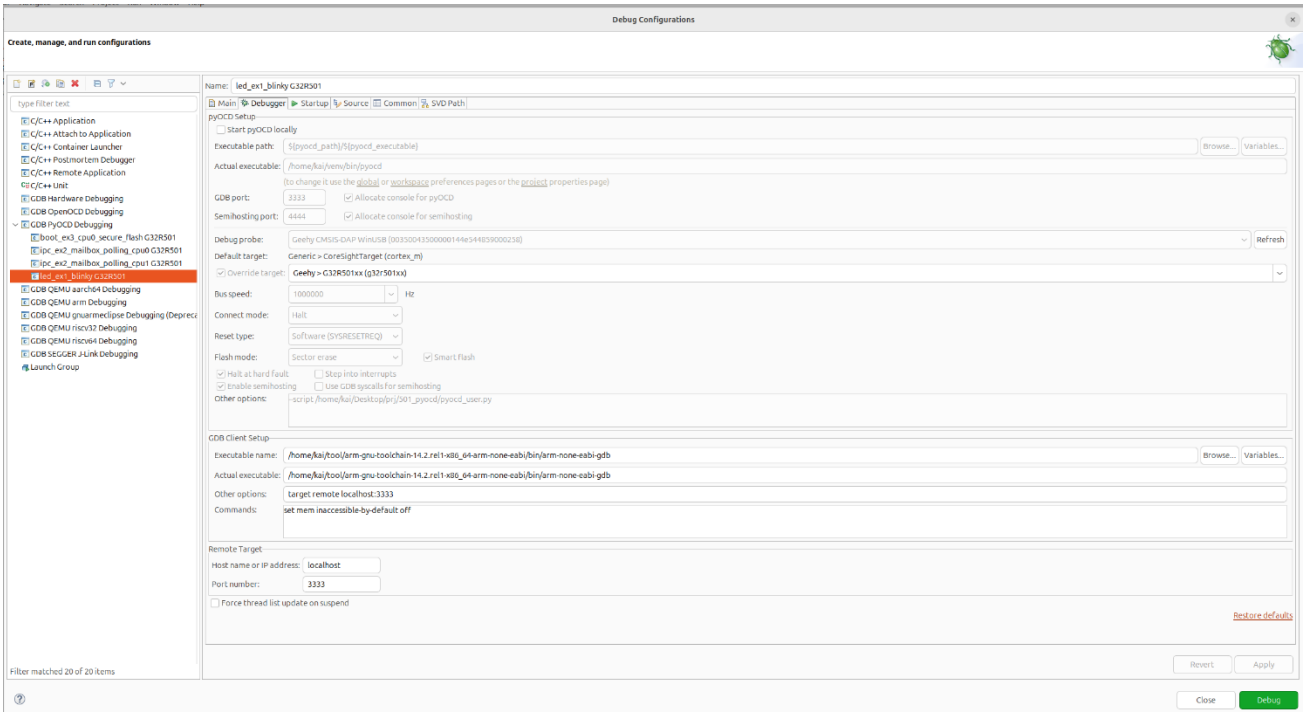
Start pyOCD gdbserver in the terminal:

Figure 73 Run pyOCD gdbserver

```
(venv) kai@Ubuntu:~/Desktop/prj/501_pyocd$ pyocd gdbserver
0001289 I Patched target_G32R501xx DECRYPT_KEYS via pyocd_user.py! [pyocd_user]
0001327 I Target type is g32r501xx [board]
0001366 I DP IDR = 0x6ba02477 (v2 rev6) [dap]
0001383 I AHB-AP#0 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001389 I AHB-AP#1 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001395 I AHB-AP#2 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001401 I AHB-AP#0 Class 0x1 ROM table #0 @ 0xe00ff000 (designer=a75:Arm China part=4d2) [rom_table]
0001406 I [0]<e000e000:SCS Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=2a04 devid=0:0:0> [rom_table]
0001409 I [1]<e0001000:DWT Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a02 devid=0:0:0> [rom_table]
0001411 I [2]<e0002000:BPU Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a03 devid=0:0:0> [rom_table]
0001414 I [3]<e0000000:ITM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=43 archid=1a01 devid=0:0:0> [rom_table]
0001417 I [5]<e0041000:ETM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=13 archid=4a13 devid=0:0:0> [rom_table]
0001422 I [6]<e0003000:??? class=9 designer=a75:Arm China part=d24 devtype=16 archid=0a06 devid=0:0:0> [rom_table]
0001425 I [7]<e0042000:CTI Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=14 archid=1a14 devid=40800:0:0> [rom_table]
0001428 I [8]<e0046000:PMC-100 class=9 designer=43b:Arm part=9ba devtype=55 archid=0a55 devid=145509d6:c105c04:0> [rom_table]
0001436 I AHB-AP#1 Class 0x1 ROM table #0 @ 0xe00ff000 (designer=a75:Arm China part=4d2) [rom_table]
0001442 I [0]<e000e000:SCS Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=2a04 devid=0:0:0> [rom_table]
0001447 I [1]<e0001000:DWT Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a02 devid=0:0:0> [rom_table]
0001450 I [2]<e0002000:BPU Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a03 devid=0:0:0> [rom_table]
0001455 I [3]<e0000000:ITM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=43 archid=1a01 devid=0:0:0> [rom_table]
0001459 I [5]<e0041000:ETM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=13 archid=4a13 devid=0:0:0> [rom_table]
0001464 I [6]<e0003000:??? class=9 designer=a75:Arm China part=d24 devtype=16 archid=0a06 devid=0:0:0> [rom_table]
0001467 I [7]<e0042000:CTI Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=14 archid=1a14 devid=40800:0:0> [rom_table]
0001470 I [8]<e0046000:PMC-100 class=9 designer=43b:Arm part=9ba devtype=55 archid=0a55 devid=145509d6:c105c04:0> [rom_table]
0001478 I CPU core #0: Star-MC2 r0p1, v7.0-M architecture [cortex_m]
0001478 I   Extensions: [DSP, FPU, FPU_DP, FPU_V5, MPU] [cortex_m]
0001478 I   FPU present: FPv5-D16-M [cortex_m]
0001479 I core is created and initialized. [target_G32R501xx]
0001484 I 4 hardware watchpoints [dwt]
0001486 I 8 hardware breakpoints, 1 literal comparators [fpb]
0001493 I 4 hardware watchpoints [dwt]
0001495 I 8 hardware breakpoints, 1 literal comparators [fpb]
r501 connect in did_connect...
0001550 I Semihost server started on port 4444 (core 0) [server]
0001569 I GDB server started on port 3333 (core 0) [gdbserver]
```

Eclipse Debugger Tab Configuration:

Figure 74



Note: The terminal path used to start the pyOCD gdbserver should include the single-core pyocd.yaml file.

6. Revision

Table 4 Document Revision History

Date	Version	Change History
January 2025	1.0	New
April 2025	1.1	Add Chapter 2.3.10.1, 4, 5

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “™” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property. Any information regarding the application of the product, Geehy hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in the sales order or the sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only 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 losses are caused to users due to user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT 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 PRODUCTS SUPPLIERS, GEEHY, GEEHY DISTRIBUTORS AND USERS. USERS (ON BEHALF OR 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 of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY'S PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED FOR USE AS CRITICAL COMPONENTS IN MILITARY, LIFE-SUPPORT, POLLUTION CONTROL, OR HAZARDOUS SUBSTANCES MANAGEMENT SYSTEMS, NOR WHERE FAILURE COULD RESULT IN INJURY, DEATH, PROPERTY OR ENVIRONMENTAL DAMAGE.

IF THE PRODUCT IS NOT LABELED AS "AUTOMOTIVE GRADE," IT SHOULD NOT BE CONSIDERED SUITABLE FOR AUTOMOTIVE APPLICATIONS. GEEHY ASSUMES NO LIABILITY FOR THE USE BEYOND ITS SPECIFICATIONS OR GUIDELINES.

THE USER SHOULD ENSURE THAT THE APPLICATION OF THE PRODUCTS COMPLIES WITH ALL RELEVANT STANDARDS, INCLUDING BUT NOT LIMITED TO SAFETY, INFORMATION SECURITY, AND ENVIRONMENTAL REQUIREMENTS. THE USER ASSUMES FULL RESPONSIBILITY FOR THE SELECTION AND USE OF GEEHY PRODUCTS. GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDES THE DOCUMENT AND PRODUCTS "AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT AND PRODUCTS (INCLUDING BUT NOT LIMITED TO LOSSES OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES). THIS COVERS POTENTIAL DAMAGES TO PERSONAL SAFETY, PROPERTY, OR THE ENVIRONMENT, FOR WHICH GEEHY WILL NOT BE RESPONSIBLE.

8. Scope of Application

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