Application Note

Document No.: AN1128

G32R501 Dual-core Emulation Guide

Version: V1.1

1 Introduction

The G32R5xx dual-core microcontroller (MCU) series is described in Table 1. The applicable product described in this document (referred to as G32R5xx MCU in this document) is based on the high-performance Arm® Cortex®-M52 32-bit RISC core. To fully utilize the dual-core architecture, the G32R501 series MCU requires specific development methods.

This debugging manual provides a guide for debugging custom applications on the G32R501 MCU, and it covers the following aspects:

- Basic principle of debugging G32R5xx dual-core MCU.
- How to use EWARM and MDK-ARM tool chains that support Geehy-Link debugging to debug dual-core devices.

For more information on the G32R501 MCU, please refer to the following documents:

- G32R501 Series Datasheet
- G32R501 Series User Manual

Table 1 G32R5xx Dual-core Models

General series	Specific supported product model
G32R5xx	G32R501DxCx7/G32R501DxYx7/G32R501DxYx8Q

Contents

1 Introduction 1

2 Basic Mechanism of Dual Cores 3

2.1 Functional Characteristics 3

2.2 Debug Access Port (DAP) 3

3 Debugging Support 5

4 MDK-ARM Dual-core Debugging Support 6

4.1 Dual-core Debugging on MDK-ARM 6

4.2 Steps for Dual-core Debugging Using GEEHY-LINK (WinUSB) 6

5 IAR EW for Arm Dual-core Debugging Support 12

5.1 Dual-core Debugging on IAR EW for Arm 12

5.2 Steps for Dual-core Debugging Using GEEHY-LINK (WinUSB) 12

6 Eclipse Dual-Core Debugging Support 18

6.1 Dual-Core Debugging on Eclipse 18

6.2 Instructions for Dual-Core Debugging Using GEEHY-LINK (WinUSB) 18

7 Revision 24

2 Basic Mechanism of Dual Cores

A multi-core processor is composed of heterogeneous core (meaning different cores) or isomorphic (same) core.

The dual cores in G32R5xx are of an asymmetric architecture. By default, CPU0 is set to master and can work normally, while CPU1 is set to slave. When the chip starts, CPU1 is set to hold and its clock is disabled. To make the slave work, CPU0 needs to enable its clock through registers and to set its boot address.

2.1 Functional Characteristics

The G32R501 series MCU can provide comprehensive and flexible debugging and performance analysis support.

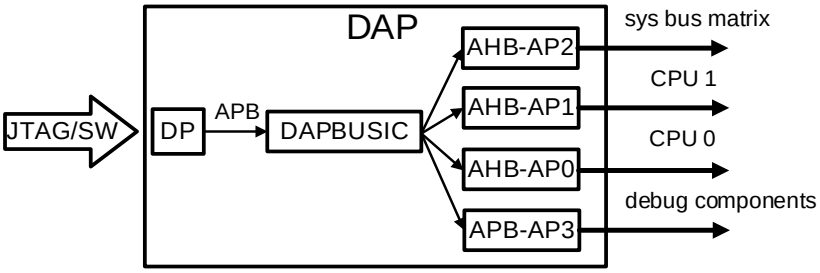
- **Independent breakpoint debugging:** Supports independent breakpoint debugging for each CPU core in the system, facilitating fine debugging of the multi-core system.
- **Code execution tracking:** Supports tracking of the code execution process, which is helpful for performance analysis and debugging.
- **JTAG debug port:** Provides a standard JTAG debugging interface, compatible with a wide range of debugging tools.
- **Serial wire debug port:** Supports Serial Wire Debug Port, to simplify debugging connection.

2.2 Debug Access Port (DAP)

The G32R501 MCU includes four access ports (AP) connected to the debug port (DP):

- AHB-AP0: CPU0 access port (AHB-AP) provides access to the integrated debugging and tracing functions in the CPU0 core through the AHB-Lite bus connected to the AHBD port of the processor.
- AHB-AP1: CPU1 access port (AHB-AP) provides access to the integrated debugging and tracing functions in the CPU1 core through the AHB-Lite bus connected to the AHBD port of the processor.
- AHB-AP2: Bus matrix access port. Allow access to the system bus matrix.
- APB-AP3: Debug access port. Allow access to external debugging components.

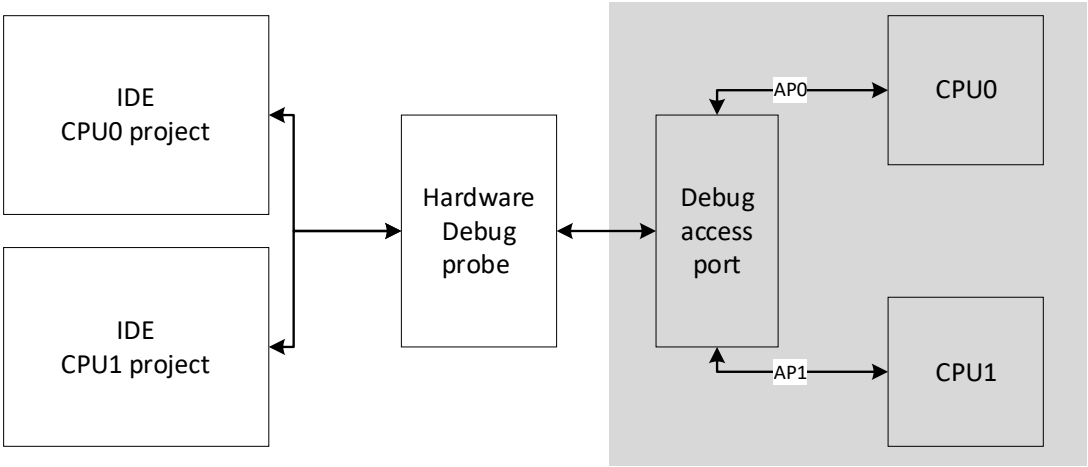
Figure 1 G32R5xx Debug Access Port (DAP)



3 Debugging Support

The dual-core debugging allows the use of a single hardware debugging probe to debug two cores simultaneously. The debugging information of two cores can be displayed in a single integrated development environment (IDE) graphical user interface (GUI), or an IDE GUI instance can be created for each core separately. The performance of the separated IDE GUI instance is shown in Figure 2.

Figure 2 Dual-core debugging IDE Diagram



To ensure smooth dual-core debugging, the debugger used must provide the following functions:

- An optional access port.
- The ability to connect multiple cores simultaneously using the same debugging probe.
- Visibility of all cores.
- Support cross triggering Arm® components.
- The possibility of switching access ports between different domains within the same debugging session to visualize the status of memory and peripheral devices.

4 MDK-ARM Dual-core Debugging Support

The latest version of MDK-ARM can be downloaded from its official website.

4.1 Dual-core Debugging on MDK-ARM

As described earlier, the dual-core asymmetric system in G32R5xx requires specific development tools, and the MDK-ARM IDE v5.40 and above support dual-core debugging of G32R5xx.

4.2 Steps for Dual-core Debugging Using GEEHY-LINK (WinUSB)

This section provides step-by-step instructions for using MDK-ARM v5.40 and GEEHY-LINK (WinUSB) debugging probes together with G32R5xx MCU.

Note:

The dual-core debugging of Arm® Cortex®-M52 is supported in MDK-ARM v5.40 and higher versions.

*Before performing the following operations, please install the G32R5xx chip support (**Geehy.G32R5xx_DFP.x.x.x.pack**).*

In this example, a project needs to be created for each core.

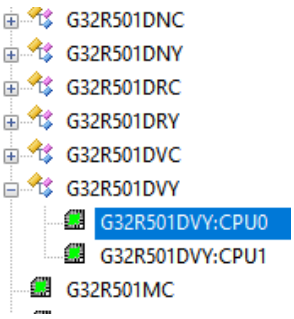
(For the example program, refer to G32R5xx_SDK\driverlib\g32r501\examples\eval\ipc\)

1. Create a new project, and configure debugging settings for CPU0 project:

   a) Open MDK-ARM and create a new project.

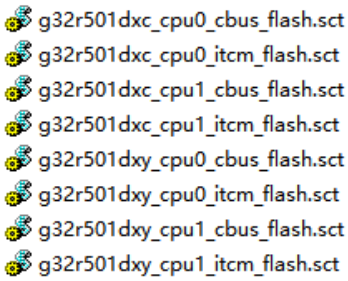
   b) Select the correct device, Project → Options for Target → Device, choose the G32R501(Figure 3) of dual-core series, and select the model with the word "CPU0" at the end.

Figure 3 G32R501 MDK Chip Selection (partial)



   c) Configure the .sct file and select the dual-core configuration.

Figure 4 Use Dual-core Configuration

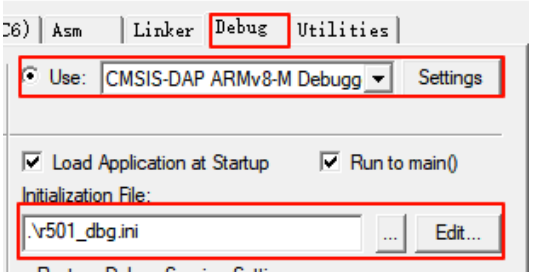


d) Configure the debugger and debugging script: Project → Options for Target → Debug

i. Select the debugger as "CMSIS-DAP ARMv8-M Debugger".

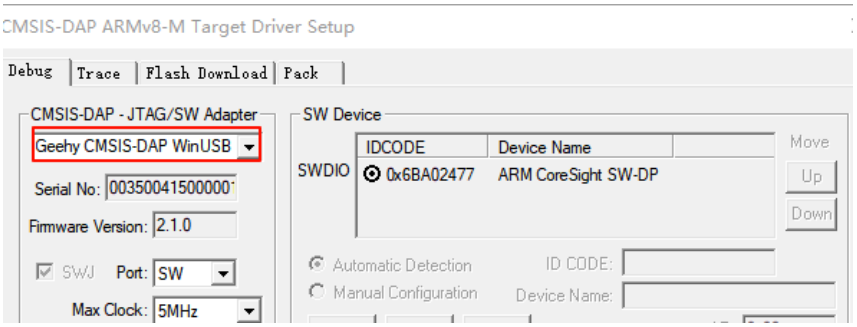
ii. Select the debugging script as “r501_deg.ini”

Figure 5 Configure Debugger and Debugging Script



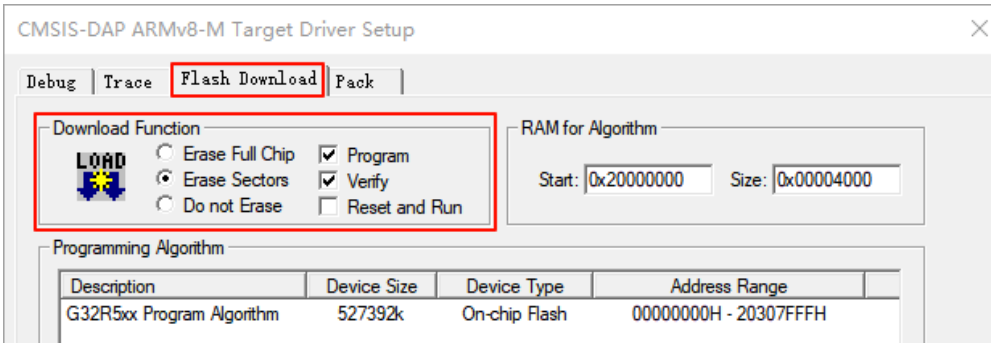
e) Configure the debugger as GEEHY-LINK.

Figure 6 Select the Debugger as GEEHY-LINK



f) Configure the program download method.

Figure 7 CPU0 Program Download Configuration

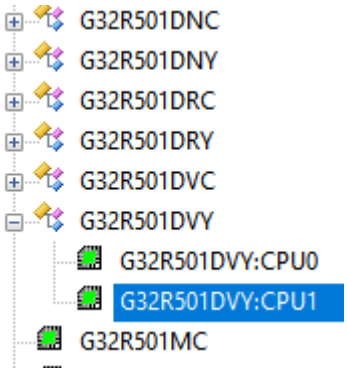


2. Create a new project, and configure debugging settings for CPU1 project:

   a) Open MDK-ARM and create a new project.

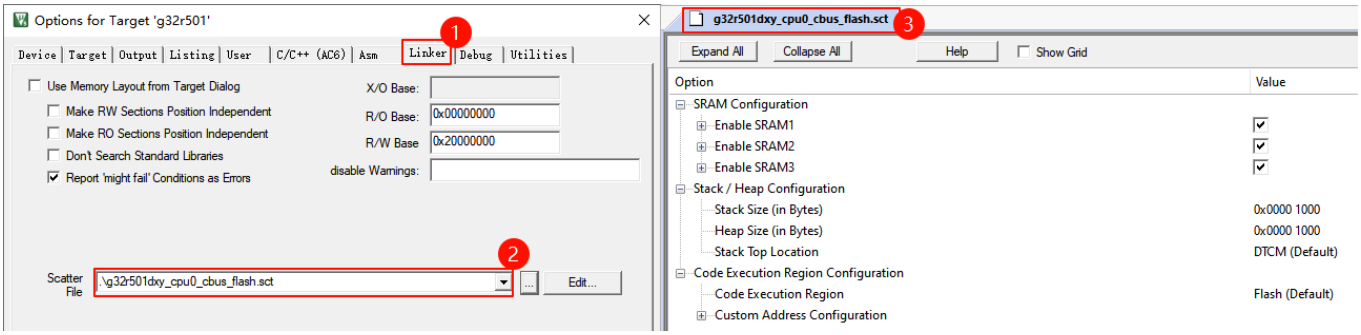
   b) Select the correct device, Project → Options for Target → Device, choose the G32R501(Figure 8) of dual-core series, and select the model with the word "CPU1" at the end.

Figure 8 CPU1 Project Selection



   c) Configure the .sct file and select the CPU1 configuration.

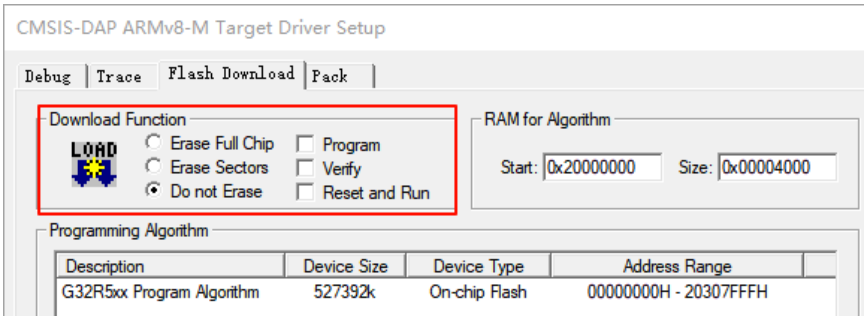
Figure 9 CPU1 .sct File Configuration



   d) Please refer to the previous chapter for debugger configuration.

   e) The configuration program does not require downloading.

Figure 10 CPU1 Program Download Configuration



3. Download CPU1 program files using CPU0 project.

Since the Flash operations in the downloading process are completed by CPU0, the CPU0 project needs to download the binary files of the programs that CPU1 needs to run through the CPU0 project during compilation.

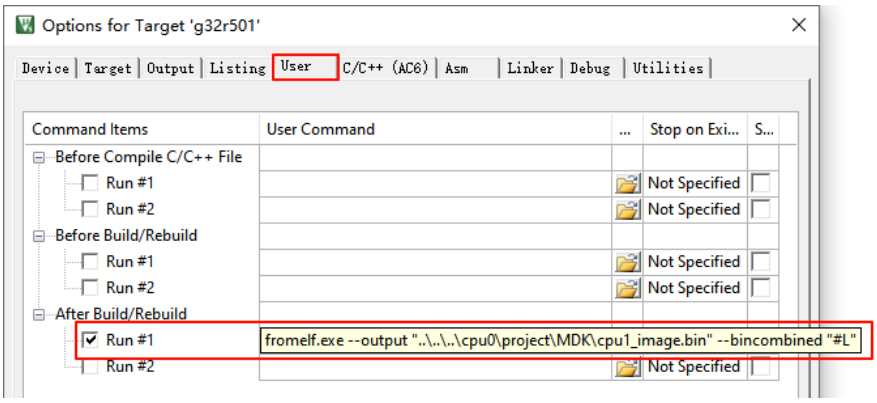
a) Configure CPU1 project to generate bin files, select Project → Options for Target → User → After Build/Rebuild, and configure the command for generating bin files.

The example uses:

fromelf.exe --output "..\..\..\cpu0\project\MDK\cpu1_image.bin" --bincombined "#L"

When using commands, pay attention to the path of the project.

Figure 11 Project→Options for Target→User→After Build/Rebuild



b) Call the bin file of the corresponding CPU1 in the CPU0 project. The example program is as follows:

```
__attribute__((__used__, section("cpu1_code")))
void G32R501_incbin(void)
{
    __asm(".incbin \"cpu1_image.bin\"");
}
```

c) If a custom .sct file is used, the user needs to specify the cpu1_code segment in the .sct file, and the definition of this segment needs to be consistent with the program space where CPU1 runs.

4. Regarding the example .sct file. The examples .sct used in this chapter are g32r501dxy_cpu0_cbus_flash.sct and g32r501dxy_cpu1_cbus_flash.sct in DK\device_support\g32r501\common\sct\.

a) In g32r501dxy_cpu0_cbus_flash.sct, after dual-core configuration is selected, the G32R5xx Flash will be divided into two. The location of the cpu1_code segment will be declared.

Figure 12 CPU1 Program Running Segment Settings of .sct in CPU0

```
#if __CORE_CONFIG == 1
LR_ROM_CPU1 __RO_BASE+__RO_SIZE/2 __RO_SIZE/2{
  ER_ROM_CPU1 __RO_BASE+__RO_SIZE/2 __RO_SIZE/2{
    .ANY (cpu1_code)
  }
}
#endif
```

The allocation of Flash for dual-core configuration is as follows:

Table 2 CPU0/1 Running Space Settings

Start address of flash	Size	Core used
0x08000000	0x050000	CPU0
0x08050000	0x050000	CPU1

b) In g32r501dxy_cpu1_cbus_flash.sc, the use configuration of Flash corresponds to the dual-core setting of r501_cpu0_flash_link.sct.

Figure 13 .sct Program Running Segment Settings of CPU1

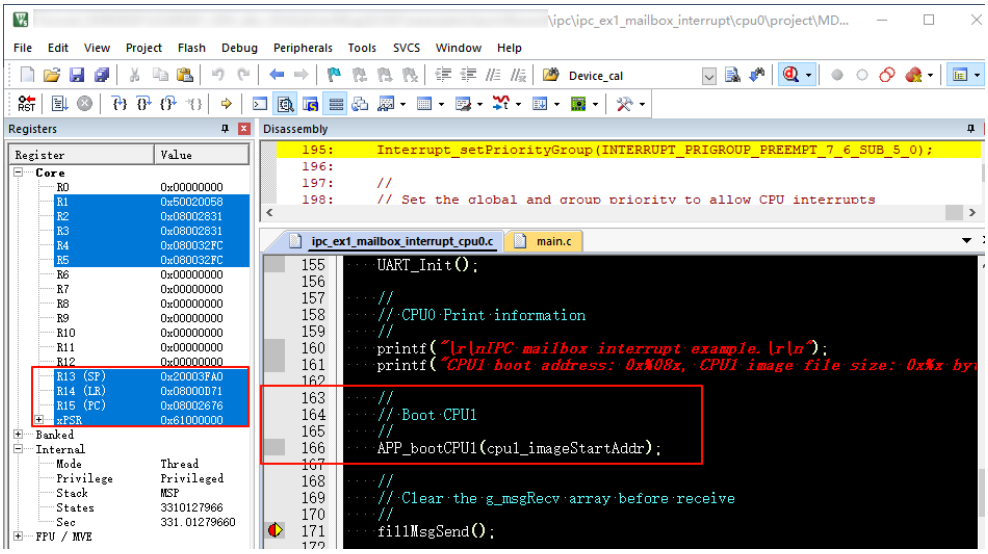
```
LR_ROM __RO_BASE+__RO_SIZE/2 __RO_SIZE/2 {
  ER_ROM __RO_BASE+__RO_SIZE/2 __RO_SIZE/2 {
    *.o (RESET, +First)
    *(InRoot$$Sections)
    .ANY (+RO)
    .ANY (+XO)
  }
```

5. Start dual-core debugging

After completing normal debugger configuration and program compilation correctly, dual-core debugging configuration can be started.

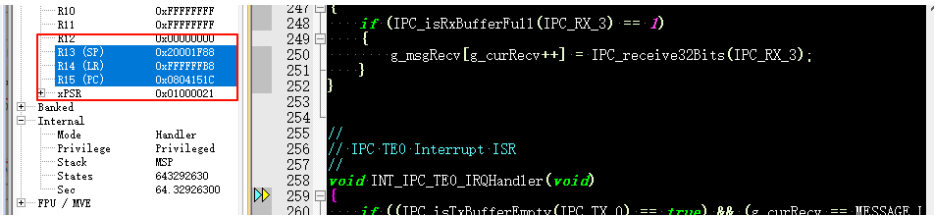
a) Start CPU0 project debugging, and set the breakpoints after setting the statement of setting CPU1 boot.

Figure 14 CPU0 Boot Debugging and CPU1 Boot



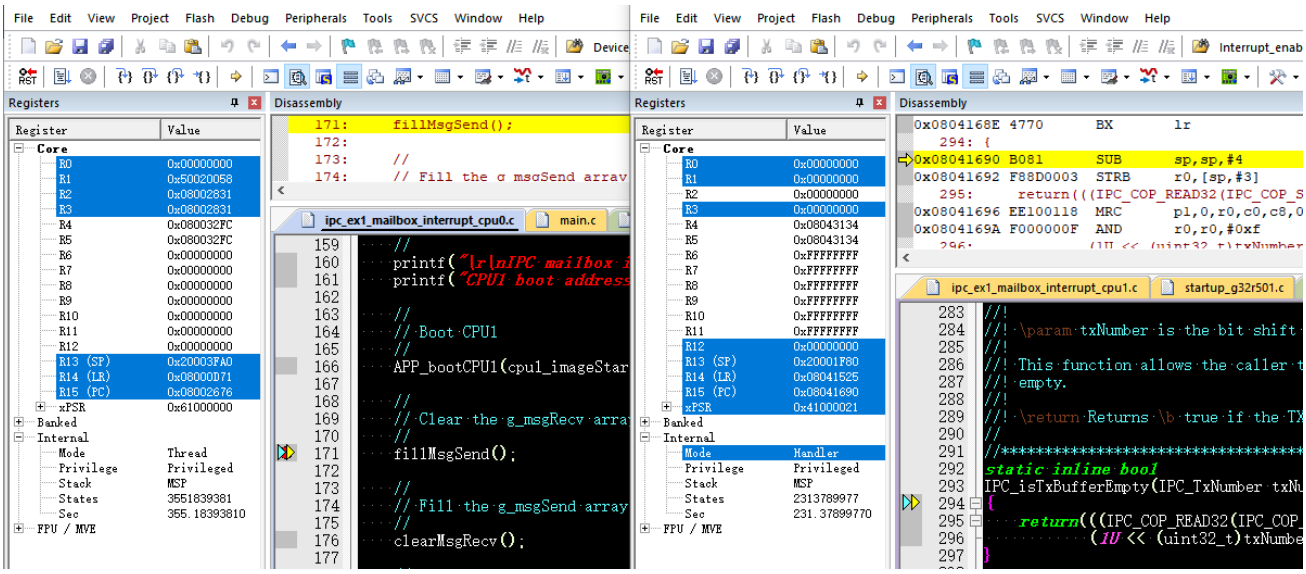
b) Start CPU1 project debugging.

Figure 15 CPU1 Boot Debugging



c) The final effect is shown in Figure 16.

Figure 16 CPU0/1 Debugging Interface



5 IAR EW for Arm Dual-core Debugging Support

The latest version of IAR EW for Arm can be downloaded from the official website of IAR.

5.1 Dual-core Debugging on IAR EW for Arm

As described earlier, the dual-core asymmetric system in G32R5xx requires specific development tools, and the IAR EW for Arm 9.60.2 and above support dual-core debugging of G32R5xx.

5.2 Steps for Dual-core Debugging Using GEEHY-LINK (WinUSB)

This section provides step-by-step instructions for using IAR EW for Arm 9.60.2 and GEEHY-LINK (WinUSB) debugging probes together with G32R5xx MCU.

Note:

The dual-core debugging of Arm® Cortex®-M52 is supported in IAR EW for Arm 9.60.2 and higher versions.

*Before performing the following operations, please install the G32R5xx chip support (**G32R5xx_AddOn_v1.0.0.exe**).*

In this example, a project needs to be created for each core.

(For the example program, refer to G32R5xx_SDK\driverlib\g32r501\examples\eval\ipc\)

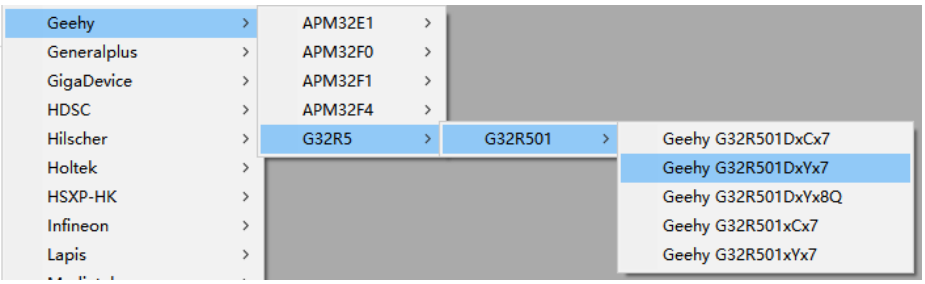
The G32R5xx chip does not distinguish between CPU0/CPU1 in the IAR EW for Arm project. When creating a new CPU0/CPU1 project, you only need to select the dual-core chip.

1. Create a new project, and configure debugging settings for CPU0/CPU1 project:

   a) Open IAR EW for Arm and create a new project.

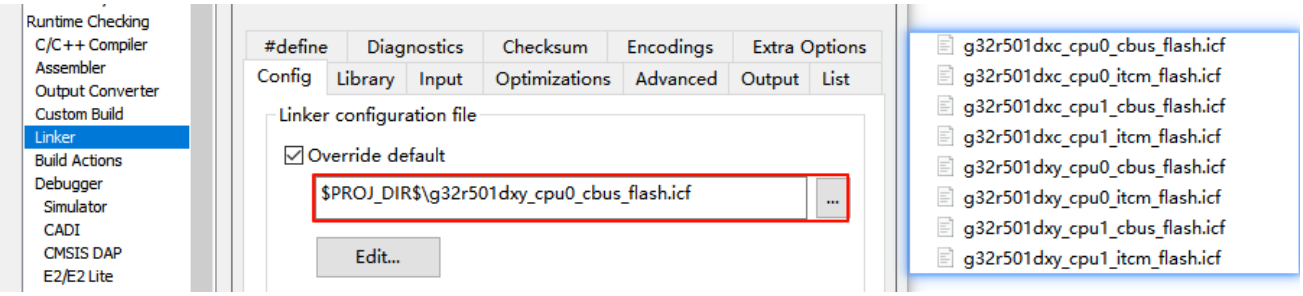
   b) Select the correct device, General Options→Target→Device, and select the G32R501 (Figure 17) of dual-core series.

Figure 17 G32R501 IAR Chip Selection (partial)



   c) Configure the .icf file and select the dual-core configuration. Please select different .icf files for different CPU projects. For example, the configuration file for CPU0 is “g32r501dxy_cpu0_cbus_flash.icf”.

Figure 18 Use Dual-core Configuration

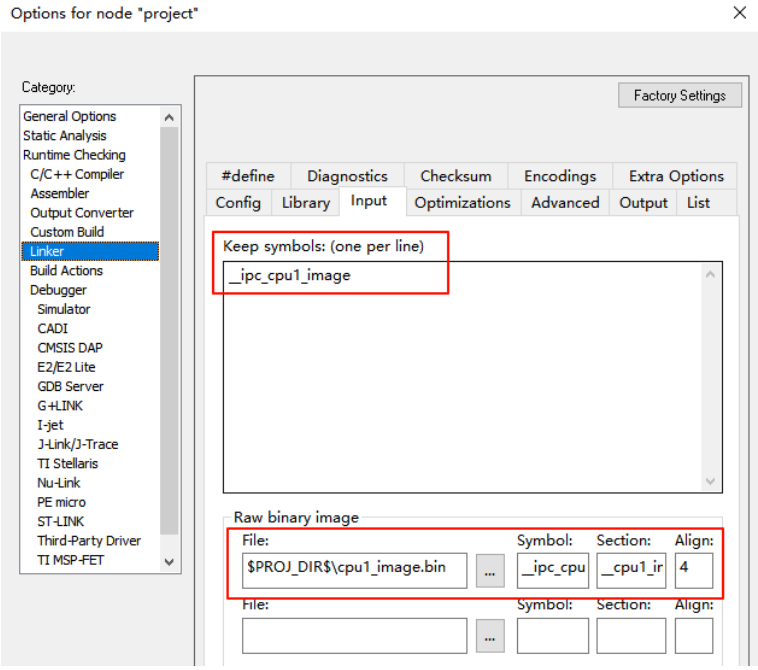


d) Configure CPU0 project to include the CPU1 running image: Linker → Input,

i. Set to keep compiling without optimizing “ipc_cpu1_image”.

ii. Load the bin file path and related configuration.

Figure 19 Use Dual-core Configuration



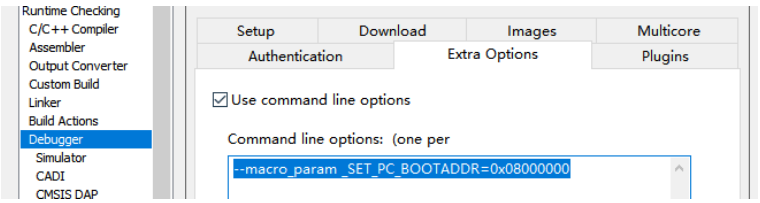
e) Configure the simulator and boot address:

i. Both CPU0 and CPU1 projects require it; select the debugger "CMSIS-DAP": Debugger→ Setup.

ii. Configure boot address for CPU0 project: Debugger → Extra Options → check "Use command line options" → enter in the text box:

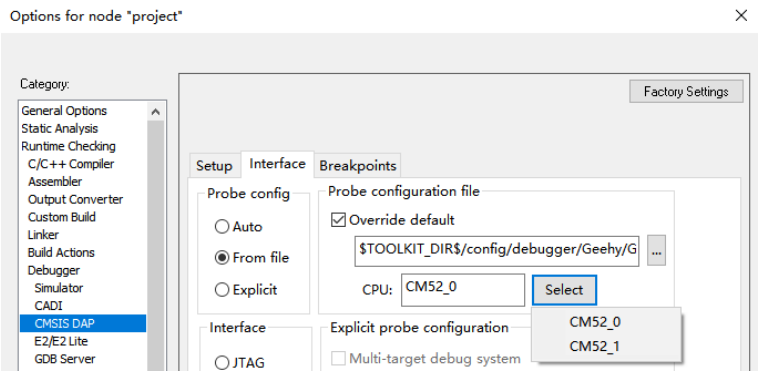
“--macro_param _SET_PC_BOOTADDR=0x08000000”. Note that the address here needs to correspond to the actual boot address of CPU0.

Figure 20 Configure Boot Address for CPU0 Project



f) Configure CPU0 project simulation AP port: CMSIS DAP→Interface→Probe config, select “From file”→ select “CM52_0” according to CPU.

Figure 21 Configure CPU0/CPU1 Project Simulation AP Port

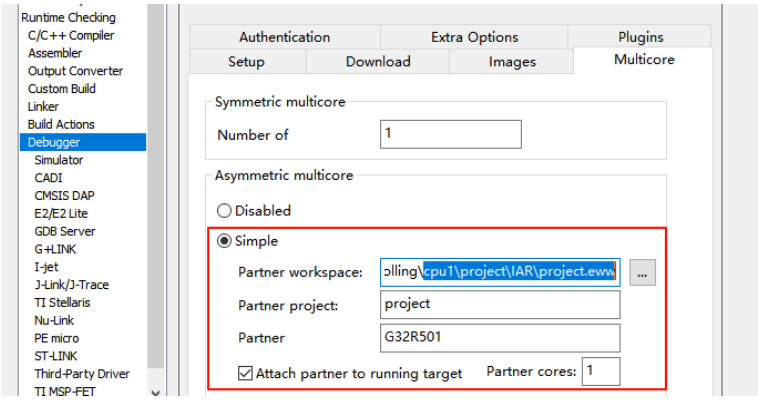


g) Configure CPU0 project to link CPU1 project. After the basic configuration of each project has been completed, CPU0 project can be used to link CPU1 project, so that the simulation of the two projects can be started simultaneously when starting the simulation.

Configuration process: Debugger→Multicore→Asymmetric multicore, check “Simple”→Select the file for CPU1 project at the Partner workspace→Fill in the CPU1 project name at the Partner project→Fill in the project tag that needs to be placed at the Partner.

The specific configuration is shown in the following figure.

Figure 22 Configure CPU0 project to link CPU1 project

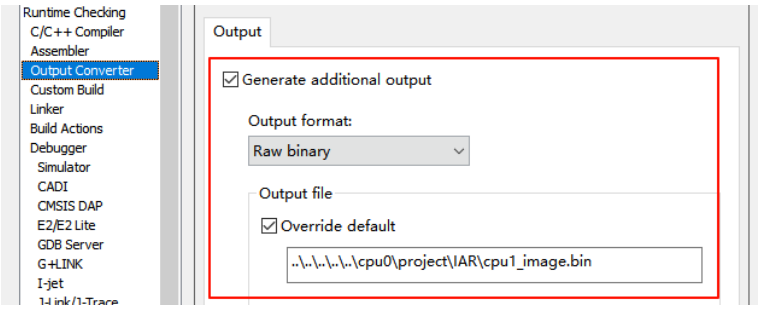


2. Additional settings for CPU1 project:

   a) Refer to the previous content to complete the correct chip selection, icf file settings, and simulation settings.

   b) Configure the output bin file to the specified directory (which needs to be consistent with the path of configuring CPU0 to include the CPU1 running image).

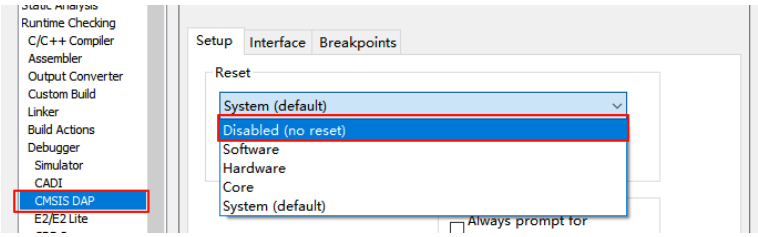
Figure 23 Configure the output bin file of CPU1 project to specified directory



   c) Configure CPU1 project simulation AP port: CMSIS DAP→Interface→Probe config, select “From file”→ select “CM52_1” according to CPU.

   d) The CPU1 project does not require additional configuration to not reset the MCU during simulation connection: CMSIS DAP → Setup → Disabled (no reset).

Figure 24 Not Resetting MCU When Configuring CPU1 Connection



3. Regarding the example .icf file. The examples .sct used in this chapter are g32r501dxy_cpu0_cbus_flash.icf and g32r501dxy_cpu1_cbus_flash.icf in SDK\device_support\g32r501\common\icf\.

   a) In g32r501dxy_cpu0_cbus_flash.icf, after dual-core configuration is selected, the G32R5xx Flash will be divided into two. In addition, the location of the cpu1_image segment will be declared.

Figure 25 CPU1 Program Running Segment Settings of .icf in CPU0

```
g32r501dxy_cpu0_cbus_flash.icf ×

  // Required in a multi-threaded application
  initialize by copy with packing = none { section __DLIB_PERTHREAD };
}

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in CBUS_FLASH_region  { readonly };
place in CPU0_ITCM_RAM_region  { section itcm.instruction, section itcm.ramfunc };
place in CPU0_DTCM_RAM_region  { section dtcm.data, section dtcm.bss, block CSTACK, block HEAP };
place in SRAM1_region  { section sram1.share_data };
place in SRAM2_region  { section sram2.share_data };
place in SRAM3_region  { section sram3.share_data };
place in RAM_region    { readwrite };
place in CPU1_region   { section __cpu1_image };
```

For the allocation of Flash for dual-core configuration, refer to Table 2 CPU0/1 Running Space Settings.Table 2 CPU0/1 Running Space Settings

4. Start dual-core debugging

   After completing normal debugger configuration and program compilation correctly, dual-core debugging configuration can be started.

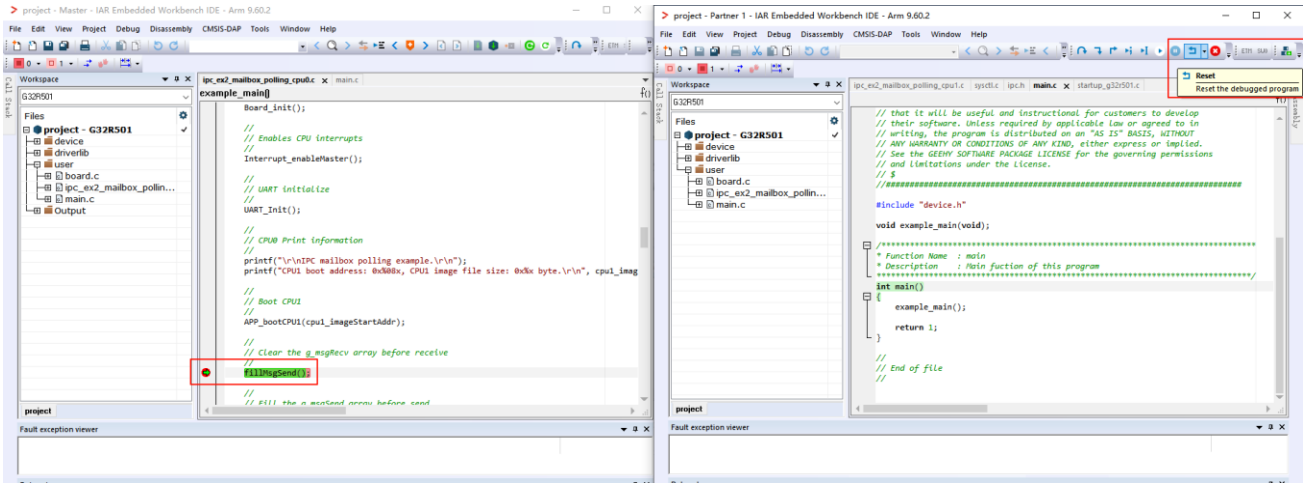
   a) Starting CPU0 project debugging will directly start two CPU. After starting the debugging window, two debugging windows can be seen:

      i. At this point, set a breakpoint after starting the CPU1 program in the CPU0 project, and run the program to this breakpoint. At this point, the CPU1 project will be able to connect to CPU1 normally.

      ii. After the CPU1 project is connected normally, please press its project "Reset" button to return the CPU1 project to its start address.

      iii. Subsequently, dual-core simulation can be made as needed.

Figure 26 IAR Dual-core Simulation Debugging Interface



6 Eclipse Dual-Core Debugging Support

The latest version of Eclipse can be downloaded from the official Eclipse website.

Note: The debugging method described in this chapter uses pyOCD+GEEHY-LINK for debugging.

6.1 Dual-Core Debugging on Eclipse

Follow the instructions in the "AN1126_G32R501 Instructions for Use of G32R501 IDE and Tool Chain" to enable pyOCD support for the G32R501 series chips.

6.2 Instructions for Dual-Core Debugging Using GEEHY-LINK (WinUSB)

This section provides a step-by-step guide for working with Eclipse and GEEHY-LINK (WinUSB) on the G32R5xx microcontroller.

In this example, a separate project needs to be created for each core.

(Example programs can be found at G32R5xx_SDK\driverlib\g32r501\examples\eval\ipc)

1. Import the projects, and after ensuring successful compilation, configure the debug tabs as follows:

   - Create a new debug configuration

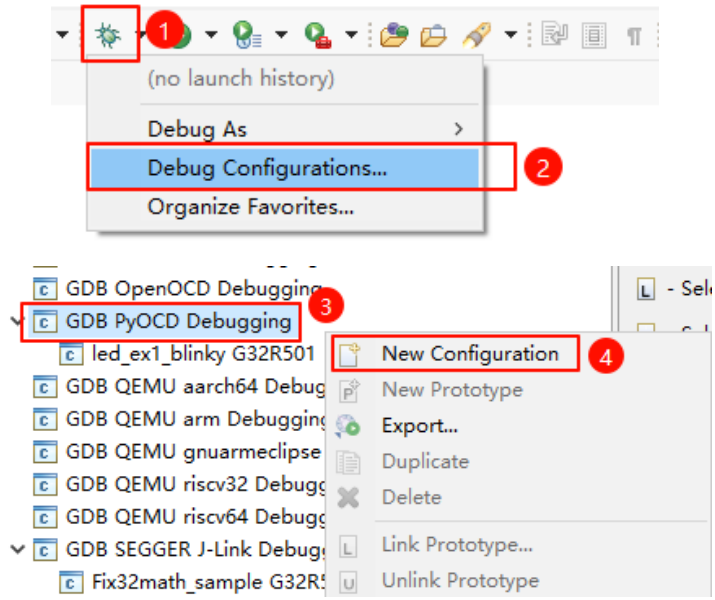
     1) Left-click the Debug icon to open the Debug configurations.

     2) Select “Debug Configurations...” from the menu.

     3) In the new window, right-click “GDB pyOCD Debugging.”

     4) Select “New Configuration” to create a new debug configuration.

Figure 27 New Configuration



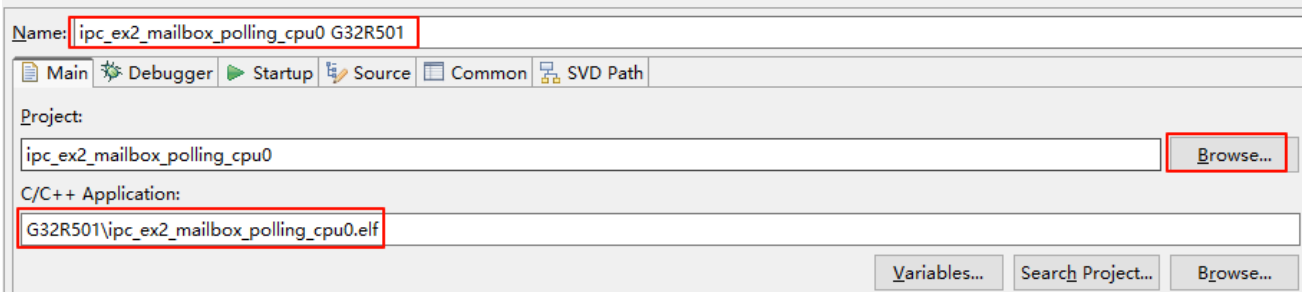
- Configure the Main tab

  1) Name the current debug configuration at the top.

  2) Click “Browse...” to select the project corresponding to the current debug configuration.

  3) Select the corresponding debug elf file, for example: G32R501\ipc_ex2_mailbox_polling_cpu0.elf. The example uses a relative path to the project file but absolute paths are also supported.

Figure 28 Configure Main

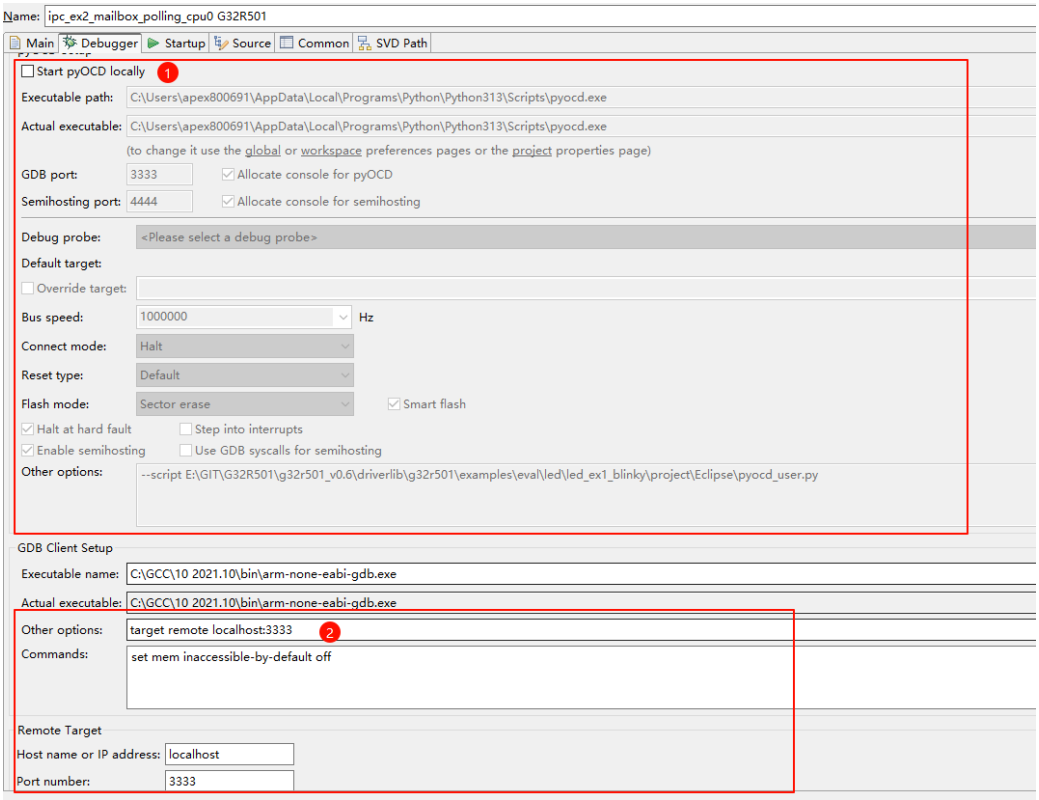


- Configure the Debugger tab

  1) Disable the setting for launching the pyOCD GDB Server by Eclipse.

  2) Set the GDB Client to connect to the appropriate core ports. The default ports are usually: core 0: 3333, core 1: 3334.

Figure 29 Dual-Core Simulation Debugger Tab



- Configure the Startup tab

  cpu0: The CPU0 Startup tab follows the single-core configuration as detailed in the "AN1126_G32R501 Instructions for Use of G32R501 IDE and Tool Chain".

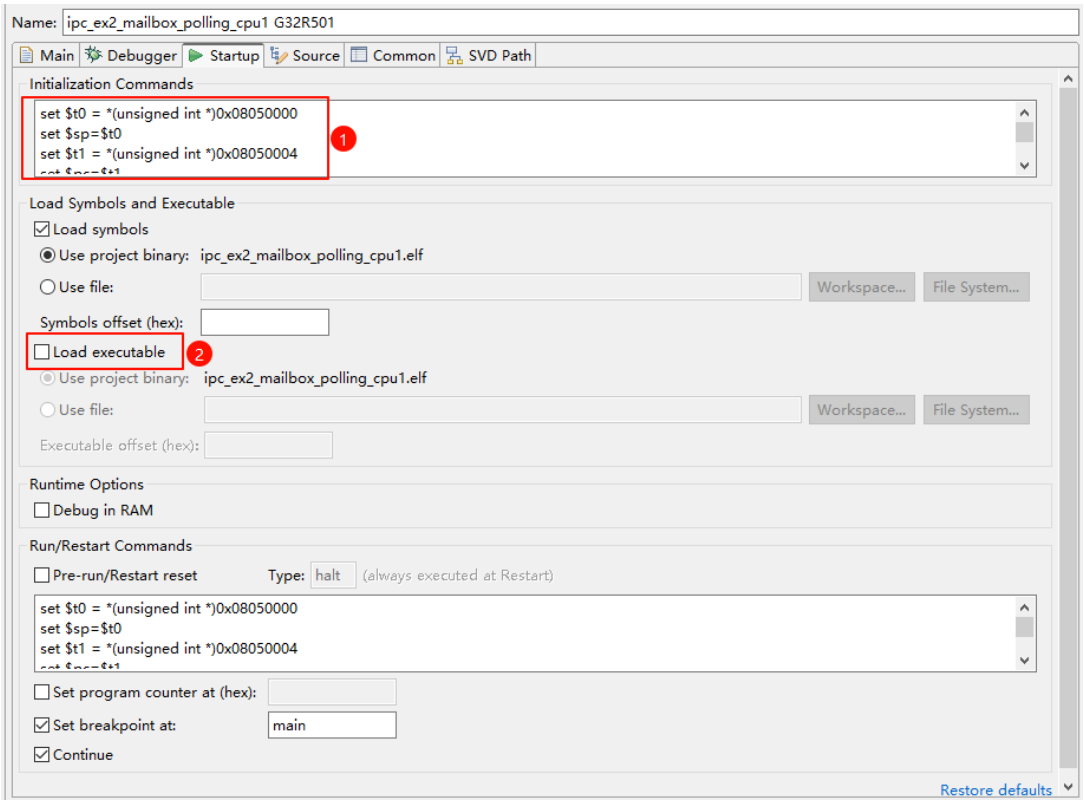
  cpu1：

  1) In the commander tab, remove the decryption sequence, leaving only the PC setup commands. (Note that the start address of the cpu1 program must be modified. The example uses 0x08050000.)

```
set $t0 = *(unsigned int *)0x08050000

set $sp=$t0

set $t1 = *(unsigned int *)0x08050004

set $pc=$t1

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

  2) Uncheck “Load executable”.

Figure 30 Startup Tab



2. Start the pyOCD gdbserver from the terminal with the command:

```
pyocd gdbserver
```

- The directory where pyOCD gdbserver is started should contain the following files:

  1) pyocd.yaml, the dual-core version, refer to SDK/device_support/g32r501/common/pyOCD/

```
target_override: g32r501dxx

frequency: 8000000 # Set 8 MHz SWD default for all probes


session:

  enable_multicore_debug: true

  persist: true
```

  2) pyocd_user.py, refer to SDK/device_support/g32r501/common/pyOCD/

Figure 31 Terminal Start pyocd gdbserver

```
C:\Windows\System32\cmd.exe - pyocd gdbserver

E:\GIT\G32R501\g32r501_v0.6\driverlib\g32r501\examples\eval\ipc\ipc_ex2_mailbox_polling>pyocd gdbserver  1
0001343 I Patched target_G32R501xx DECRYPT_KEYS via pyocd_user.py! [pyocd_user]
0001370 I Target type is g32r501dxx [board]
0001400 I DP IDR = 0x6ba02477 (v2 rev6) [dap]
0001404 I AHB-AP#0 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001405 I AHB-AP#1 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001407 I AHB-AP#2 IDR = 0x84770001 (AHB-AP var0 rev8) [discovery]
0001409 I AHB-AP#0 Class 0x1 ROM table #0 @ 0xe00ff000 (designer=a75:Arm China part=4d2) [rom_table]
0001411 I [0]<e000e000:SCS Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=2a04 devid=0:0:0> [rom_
table]
0001413 I [1]<e0001000:DWT Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a02 devid=0:0:0> [rom_
table]
0001414 I [2]<e0002000:BPU Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a03 devid=0:0:0> [rom_
table]
0001415 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]
0001418 I [6]<e0003000:??? class=9 designer=a75:Arm China part=d24 devtype=16 archid=0a06 devid=0:0:0> [rom_table]
0001419 I [7]<e0042000:CTI Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=14 archid=1a14 devid=40800:0:0> [
rom_table]
0001420 I [8]<e0046000:PMC-100 class=9 designer=43b:Arm part=9ba devtype=55 archid=0a55 devid=145509d6:c105c04:0> [ro
m_table]
0001423 I AHB-AP#1 Class 0x1 ROM table #0 @ 0xe00ff000 (designer=a75:Arm China part=4d2) [rom_table]
0001425 I [0]<e000e000:SCS Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=2a04 devid=0:0:0> [rom_
table]
0001426 I [1]<e0001000:DWT Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a02 devid=0:0:0> [rom_
table]
0001428 I [2]<e0002000:BPU Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=00 archid=1a03 devid=0:0:0> [rom_
table]
0001430 I [3]<e0000000:ITM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=43 archid=1a01 devid=0:0:0> [rom_
table]
0001432 I [5]<e0041000:ETM Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=13 archid=4a13 devid=0:0:0> [rom_
table]
0001433 I [6]<e0003000:??? class=9 designer=a75:Arm China part=d24 devtype=16 archid=0a06 devid=0:0:0> [rom_table]
0001434 I [7]<e0042000:CTI Star-MC2 class=9 designer=a75:Arm China part=d24 devtype=14 archid=1a14 devid=40800:0:0> [
rom_table]
0001436 I [8]<e0046000:PMC-100 class=9 designer=43b:Arm part=9ba devtype=55 archid=0a55 devid=145509d6:c105c04:0> [ro
m_table]
0001439 I CPU core #0: Star-MC2 r0p1, v7.0-M architecture [cortex_m]
0001439 I   Extensions: [DSP, FPU, FPU_DP, FPU_V5, MPU] [cortex_m]
0001439 I   FPU present: FPv5-D16-M [cortex_m]
0001440 I core0 is created and initialized. [target_G32R501xx]
0001440 I Enabling clock and setting startup address for core1 via AP0... [target_G32R501xx]
0001444 I CPU core #1: Star-MC2 r0p1, v7.0-M architecture [cortex_m]
0001444 I   Extensions: [DSP, FPU, FPU_DP, FPU_V5, MPU] [cortex_m]
0001444 I   FPU present: FPv5-D16-M [cortex_m]
0001445 I core1 is created and initialized. [target_G32R501xx]
0001446 I 4 hardware watchpoints [dwt]
0001447 I 8 hardware breakpoints, 1 literal comparators [fpb]
0001450 I 4 hardware watchpoints [dwt]
0001451 I 8 hardware breakpoints, 1 literal comparators [fpb]
r501 connect in did_connect...
Reading SP/PC from 0x08000000:
  MSP = 0x20003FF8
  PC  = 0x08001F65
set_pc_cmd: Registers updated: target resuming.
0001491 I Semihost server started on port 4444 (core 0) [server]
0001869 I GDB server started on port 3333 (core 0) [gdbserver]
0001872 I Semihost server started on port 4445 (core 1) [server]  2
0001872 I GDB server started on port 3334 (core 1) [gdbserver]
```

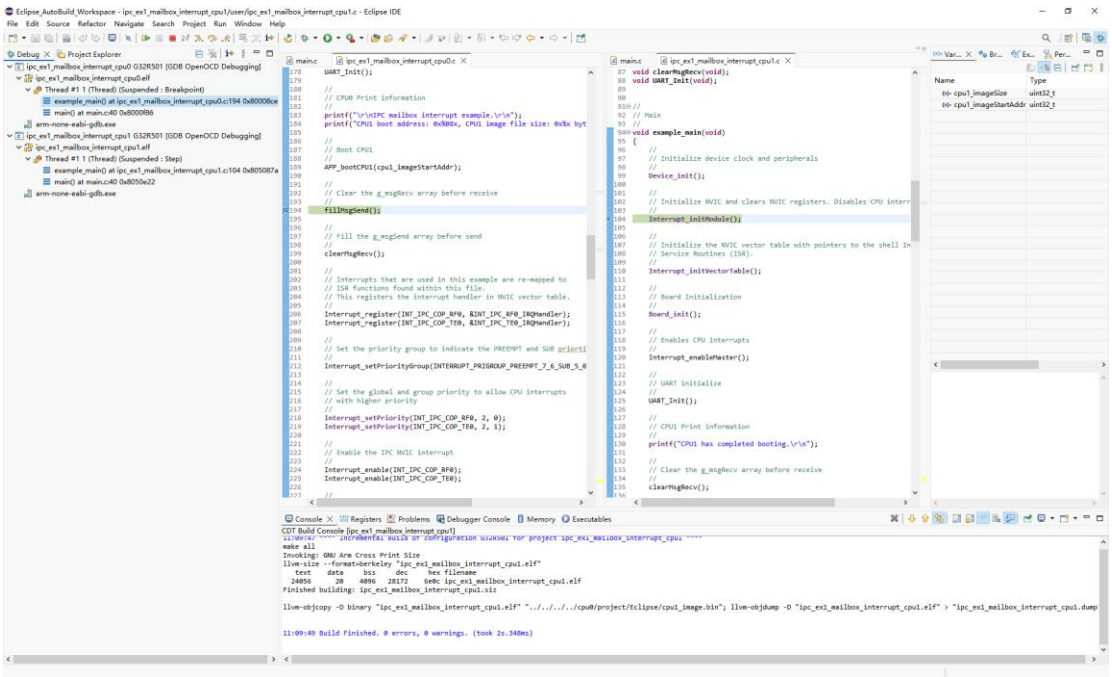
3. Start two debug sessions in Eclipse separately.

- Launch debugging for the CPU0 project and set a breakpoint after the statement that starts CPU1. In the example, set the breakpoint after the statement APP_bootCPU1(cpu1_imageStartAddr);
- Launch debugging for the CPU1 project.
- Perform dual-core debugging as needed.

    1) Click the thread corresponding to cpu0 to control cpu0 simulation.

    2) Click the thread corresponding to cpu1 to control cpu1 simulation.

Figure 32 Eclipse Simulation Debugging Interface



7 Revision

Table 3 Document Revision History

Date	Version	Change History
January 2025	1.0	New
April 2025	1.1	Add the new section about Eclipse dual-core debugging support.

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.