Application Note

Document No.: AN1128

G32R501 Dual-core Debug Guide

Version: V1.2

1 Introduction

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

Note:

（1） This application note applies **only to G32R501 dual-core packages (for example Dx)** (see Table 1).

（2） **G32R502 and devices without CPU1 are not applicable** to the dual-core debug steps in this document.

（3） Slave boot APIs, registers, and IPC example details are authoritative in **AN1135** and the sources under “driverlib/g32r501/examples/eval/ipc/”. This document focuses on dual-core debug connection steps on three toolchains.

This guide covers debugging custom dual-core applications on the G32R501D MCU, including:

- Basic principles of G32R5xx dual-core MCU debugging.
- How to debug dual-core devices with EWARM, MDK-ARM, and Eclipse toolchains that support Geehy-Link.

For more device information, refer to:

- G32R501 Series Datasheet
- G32R501 Series User Manual
- AN1135 G32R501 IPC Application Note

G32R5xx dual-core models are listed in Table 1.

Table 1 G32R5xx Dual-core Models

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

Note: Prefer opening the SDK IPC dual projects under “G32R5xx_SDK\driverlib\g32r501\examples\eval\ipc\” first, then use this document to understand the settings. Hand-built projects must fully implement the checkpoints in §4.2 / §5.2 / §6.2.

Contents

2 Basic Mechanism of Dual Cores

A multi-core processor consists of heterogeneous cores (different cores) or isomorphic cores (identical cores).

The dual cores in G32R5xx use an asymmetric architecture. By default, CPU0 is master and runs normally, while CPU1 is slave. At chip start, CPU1 is held and its clock is disabled. To run the slave, CPU0 must enable its clock through registers and set its boot address (typically wrapped as “APP_bootCPU1” in examples, calling “SysCtl_setCPU1BootAddress” and “SysCtl_enableBootCPU1”).

2.1 Functional Characteristics

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

- Independent breakpoint debugging: Supports independent breakpoint debugging for each CPU core.
- Code execution tracing: Supports tracing code execution for performance analysis and debugging.
- JTAG debug port: Provides a standard JTAG interface compatible with a wide range of tools.
- Serial Wire Debug Port: Supports SWD to simplify the debug connection.

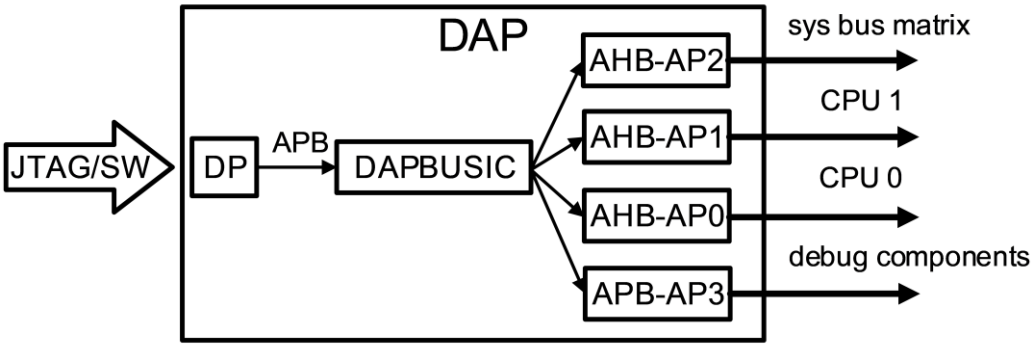
2.2 Debug Access Port (DAP)

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

- AHB-AP0: CPU0 access port (AHB-AP) provides access to integrated debug and trace features in CPU0 via the AHB-Lite bus connected to the processor AHBD port.
- AHB-AP1: CPU1 access port (AHB-AP) provides access to integrated debug and trace features in CPU1 via the AHB-Lite bus connected to the processor AHBD port.
- AHB-AP2: Bus matrix access port. Allows access to the system bus matrix.
- APB-AP3: Debug access port. Allows access to external debug components.

The G32R5xx Debug Access Port (DAP) is shown in Figure 1.

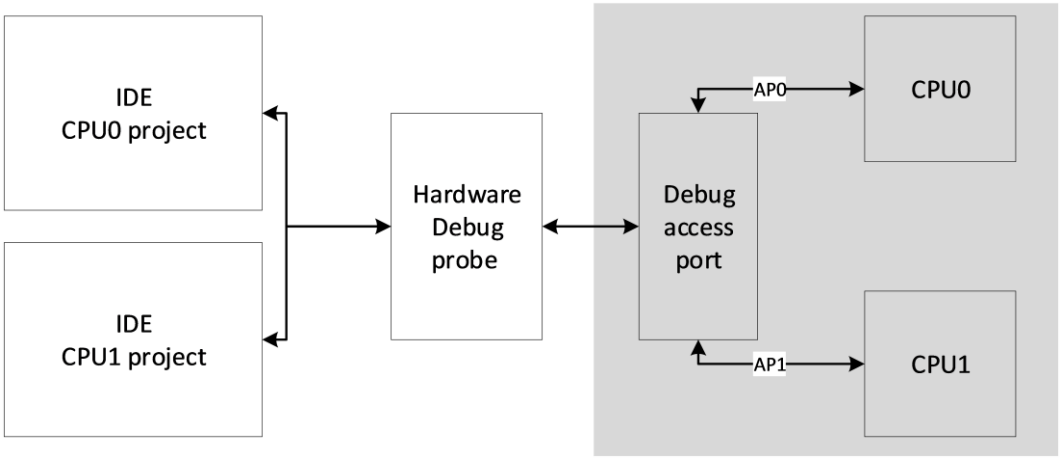
Figure 1 G32R5xx Debug Access Port (DAP)



3 Debugging Support

Dual-core debugging allows a single hardware debug probe to debug both cores simultaneously. Debug information for both cores can be shown in one IDE GUI, or a separate IDE GUI instance can be created for each core. Separated IDE GUI instances are illustrated in Figure 2.

Figure 2 Dual-core Debugging IDE Diagram



To ensure smooth dual-core debugging, the Debugger must provide:

- Selectable access ports.
- Ability to connect multiple cores simultaneously with the same probe.
- Visibility of all cores.
- Support for cross-triggering Arm® components.
- Ability to switch access ports between domains in the same session to view memory and peripherals.

4 MDK-ARM Dual-core Debugging Support

The latest 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 tools. 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 a GEEHY-LINK (WinUSB) probe with a G32R5xx MCU.

Note:

（1） Dual-core debugging of Arm® Cortex®-M52 is supported in MDK-ARM v5.40 and higher.

（2） Before the following operations, install G32R5xx device support (“Geehy.G32R5xx_DFP.x.x.x.pack”).

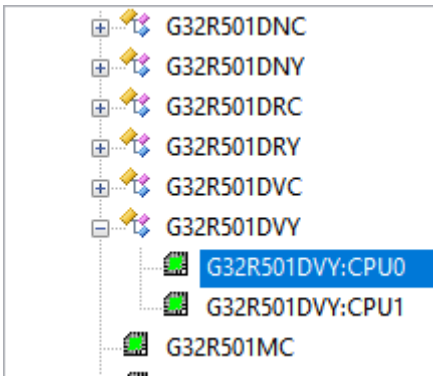
（3） In this example, create one project per core.

（4） Example programs: “G32R5xx_SDK\driverlib\g32r501\examples\eval\ipc\”.

4.2.1 Create and Configure the CPU0 Project

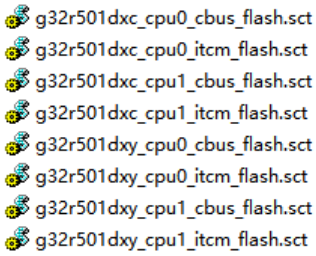
1. Open MDK-ARM and create a new project.
2. Select the correct device: Project → Options for Target → Device, choose G32R501 of the dual-core series (Figure 3), and select the part ending with “CPU0”.

Figure 3 G32R501 MDK Chip Selection (partial)



3. Configure the “.sct” file and select the **dual-core** configuration (CPU0 script must be “g32r501dxy_cpu0_cbus_flash.sct” or the matching cpu0 dual-core template, Figure 4).

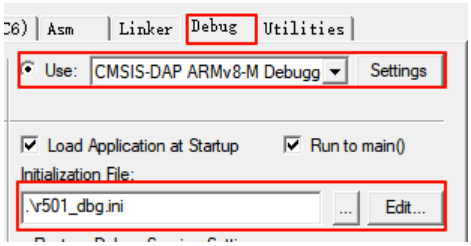
Figure 4 Use Dual-core Configuration



4. Configure the Debugger and debug script: Project → Options for Target → Debug.

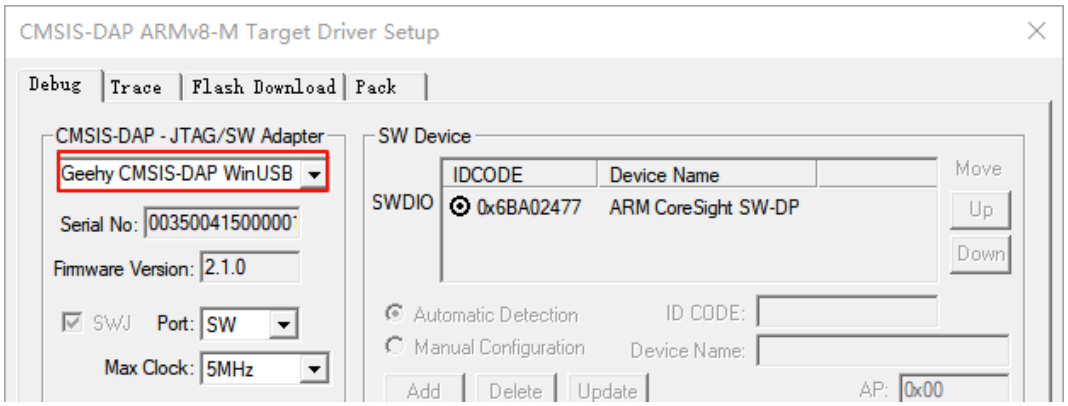
- Select Debugger “CMSIS-DAP ARMv8-M Debugger”.
- Select debug script **“r501_dbg.ini”**.

Figure 5 Configure Debugger and Debugging Script



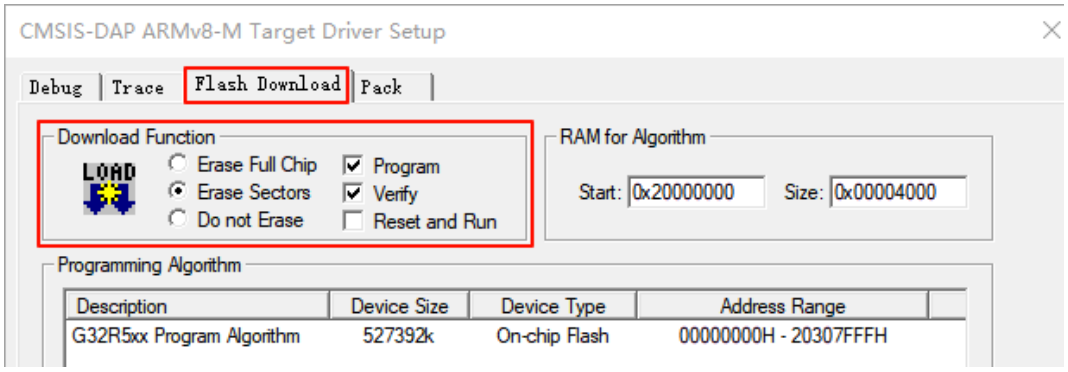
5. Configure the Debugger as GEEHY-LINK (Figure 6).

Figure 6 Select the Debugger as GEEHY-LINK



6. Configure the program download method (Figure 7).

Figure 7 CPU0 Program Download Configuration



7. **CPU0 application must-haves (aligned with examples; all required):**

- Implement or port **"APP_bootCPU1"** (or equivalently call "SysCtl_setCPU1BootAddress" + "SysCtl_enableBootCPU1").
- Reserve the "cpu1_code" section and embed the CPU1 image via ".incbin" (or an equivalent linker method); see "Download the CPU1 image via the CPU0 project" below.
- Place the dual-core debug breakpoint **after the "APP_bootCPU1(...)" call** (consistent with §6.2 Eclipse).
- See **AN1135** and IPC example sources for boot API details.

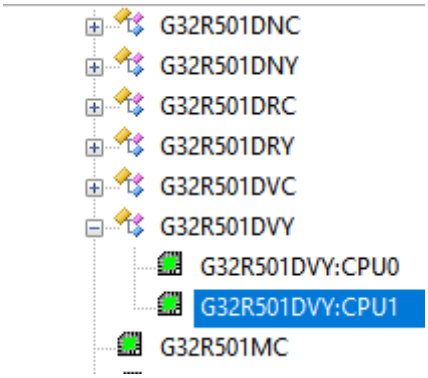
Example (same pattern as "ipc_ex1_mailbox_interrupt" CPU0 source):

```
void APP_bootCPU1(uint32_t cpu1Address)
{
    SysCtl_setCPU1BootAddress(cpu1Address);
    SysCtl_enableBootCPU1();
}
/* In main, after obtaining cpu1_imageStartAddr: */
APP_bootCPU1(cpu1_imageStartAddr);
```

4.2.2 Create and Configure the CPU1 Project

1. Open MDK-ARM and create a new project.
2. Select the correct device: Project → Options for Target → Device, choose G32R501 of the dual-core series (Figure 8), and select the part ending with "CPU1".

Figure 8 CPU1 Project Selection



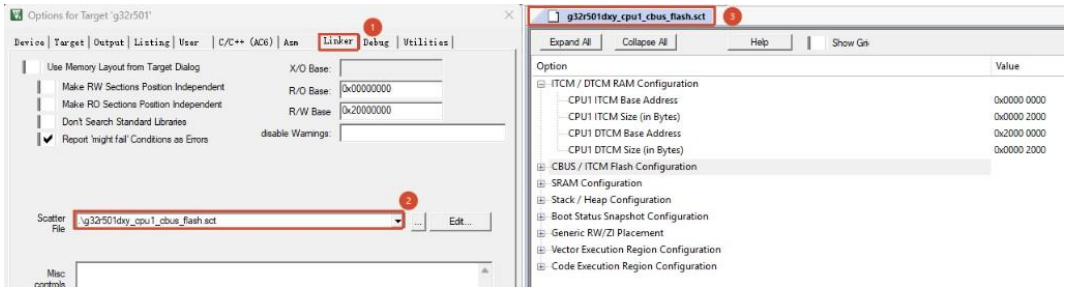
3. Configure the ".sct" file and **select the CPU1 script**, for example "g32r501dxy_cpu1_cbus_flash.sct" (Figure 9). **Do not** select the CPU0 script "g32r501dxy_cpu0_cbus_flash.sct".

CPU0 / CPU1 scatter files are compared in Table 2.

Table 2 CPU0 / CPU1 Scatter File Mapping

Project	Scatter file	Notes
CPU0	"g32r501dxy_cpu0_cbus_flash.sct"	Dual-core Flash split and "cpu1_code" section
CPU1	"g32r501dxy_cpu1_cbus_flash.sct"	ROM in CPU1 Flash window; **never** use the cpu0 script

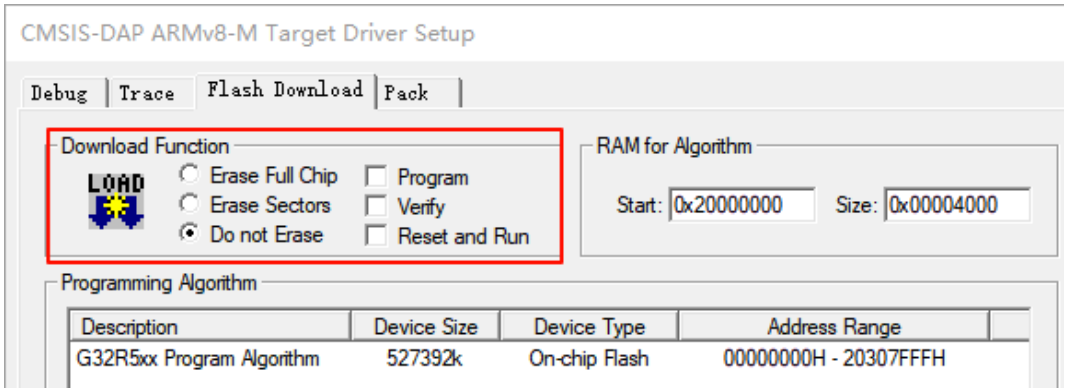
Figure 9 CPU1 .sct File Configuration



4. Keep the CPU1 project Debug page consistent with CPU0: same probe type (for example CMSIS-DAP / GEEHY-LINK) and Initialization File; Device set to CPU1; cancel Flash download for this project as described below.

5. Configure the CPU1 project for **no download** (CPU0 programs Flash, Figure 10).

Figure 10 CPU1 Program Download Configuration



4.2.3 Download the CPU1 Image via the CPU0 Project

Because Flash operations during download are performed by CPU0, the CPU0 project must include the CPU1 binary at link time.

Required order:

1. Build the **CPU1** project first to generate “cpu1_image.bin”.
2. Then build and download the **CPU0** project (which embeds that bin).

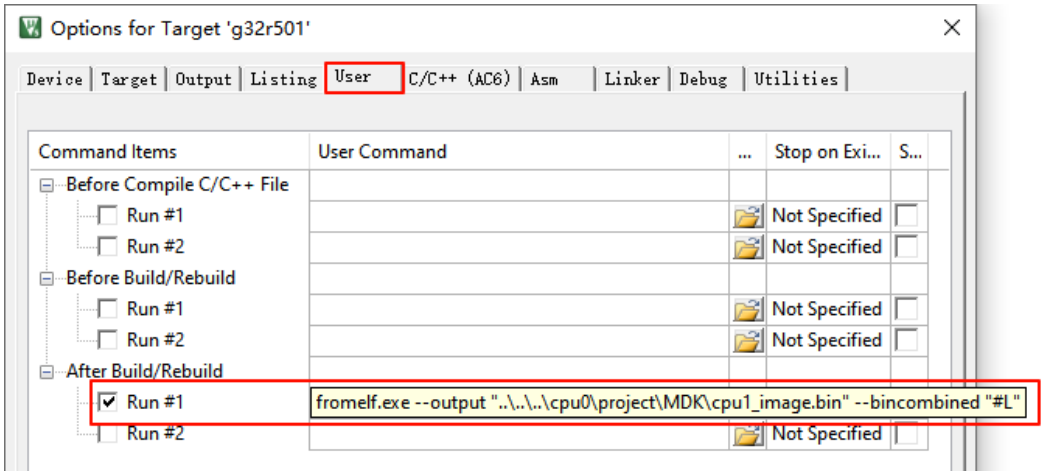
Configuration steps:

1. Configure the CPU1 project to generate a bin file under Project → Options for Target → User → After Build/Rebuild. Example (aligned with current “ipc_ex1_mailbox_interrupt”; adjust relative depth for your tree):

```
fromelf --bin --output "..\..\..\cpu0\source\cpu1_image.bin" "#L"
```

Keep the relative path writable to CPU0-side “source/cpu1_image.bin” (Figure 11).

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



2. Embed the CPU1 bin in the CPU0 project. Example (MDK path sketch; must match this project User/incbin):

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

3. If a custom ".sct" is used, define the "cpu1_code" section so that it matches the CPU1 run address window.

4.2.4 Example `.sct` Files

Example ".sct" files used in this chapter are "g32r501dxy_cpu0_cbus_flash.sct" and "g32r501dxy_cpu1_cbus_flash.sct" under "SDK\device_support\g32r501\common\sct\".

Note: Linker-script screenshots in this chapter correspond to the current "g32r501dxy_cpu0_cbus_flash.sct" / "g32r501dxy_cpu1_cbus_flash.sct" files under "device_support/g32r501/common/sct/". If a local UI still shows a legacy name such as "r501_cpu0_flash_link.sct", select the current scripts above. If screenshot pixels differ from the current Configuration Wizard fields, follow the SDK script contents.

1. In "g32r501dxy_cpu0_cbus_flash.sct", after dual-core configuration is selected, Flash is split into two and the "cpu1_code" location is declared (Figure 12). Flash allocation for dual-core is shown in Table 3.

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

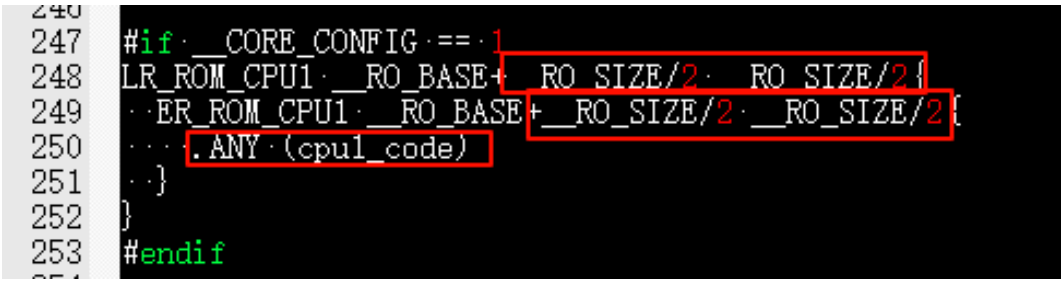


Table 3 CPU0/1 Running Space Settings

Flash start address	Size	Core used
0x08000000	0x050000	CPU0
0x08050000	0x050000	CPU1

2. In "g32r501dxy_cpu1_cbus_flash.sct", the Flash window must match the dual-core split in CPU0's "g32r501dxy_cpu0_cbus_flash.sct" (Figure 13).

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)
  }
```

4.2.5 Start Dual-core Debugging

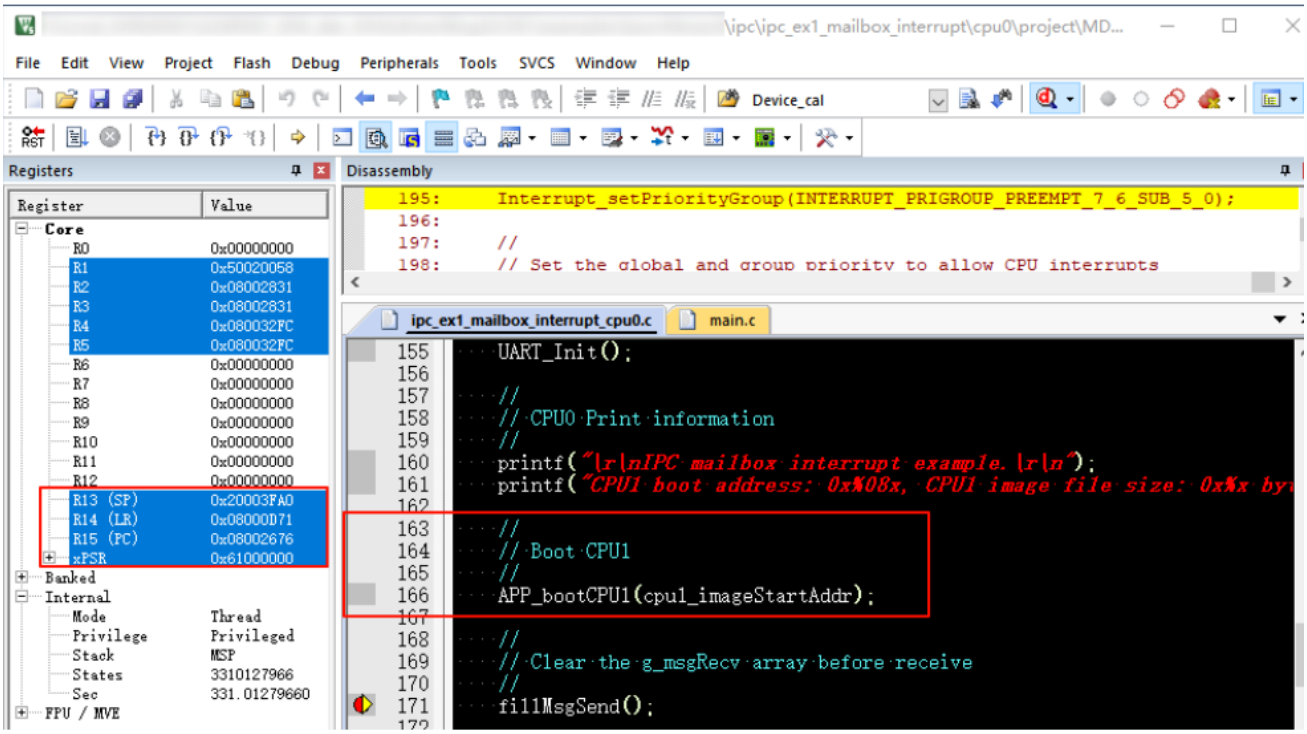
After Debugger configuration and a successful build:

1. Start CPU0 project debugging and set a breakpoint **after the "APP_bootCPU1(cpu1_imageStartAddr);" call**. Taking "ipc_ex1_mailbox_interrupt" as an example:

```
//
// Boot CPU1
//
APP_bootCPU1(cpu1_imageStartAddr);
```

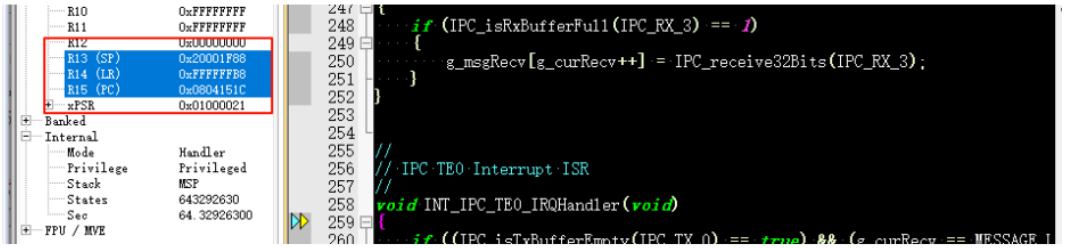
The breakpoint location is as shown in Figure 14.

Figure 14 CPU0 Boot Debugging and CPU1 Boot



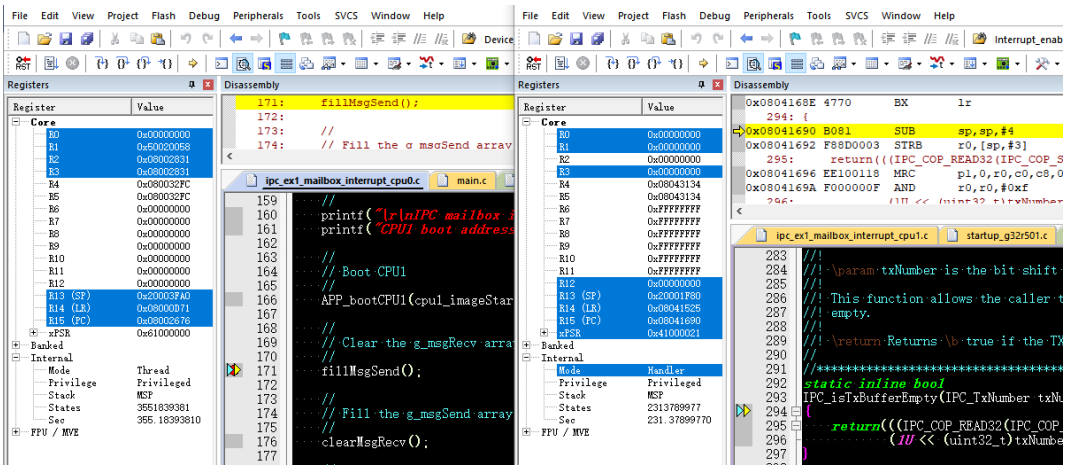
2. Start CPU1 project debugging (Figure 15).

Figure 15 CPU1 Interrupt Debug Sketch (IPC Handler)



3. The final result is shown in Figure 16.

Figure 16 CPU0/1 Debugging Interface



5 IAR EW for Arm Dual-core Debugging Support

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

5.1 Dual-core Debugging on IAR EW for Arm

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 a GEEHY-LINK (WinUSB) probe with a G32R5xx MCU.

Note:

（1） Dual-core debugging of Arm® Cortex®-M52 is supported in IAR EW for Arm 9.60.2 and higher.

（2） Before the following operations, install G32R5xx support (“G32R5xx_AddOn_v1.0.0.exe”).

（3） Create one project per core.

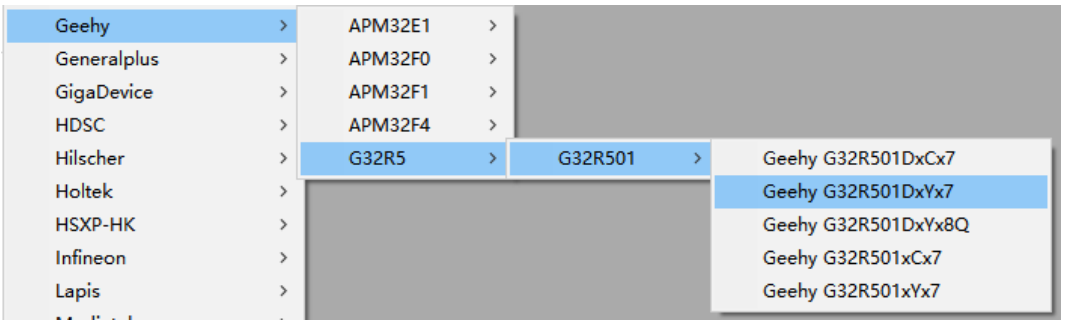
（4） Example programs: “G32R5xx_SDK\driverlib\g32r501\examples\eval\ipc\”.

（5） IAR device selection does not distinguish CPU0/CPU1; select the dual-core device and differentiate cores with **“.icf”** files.

5.2.1 Configure CPU0 / CPU1 Projects

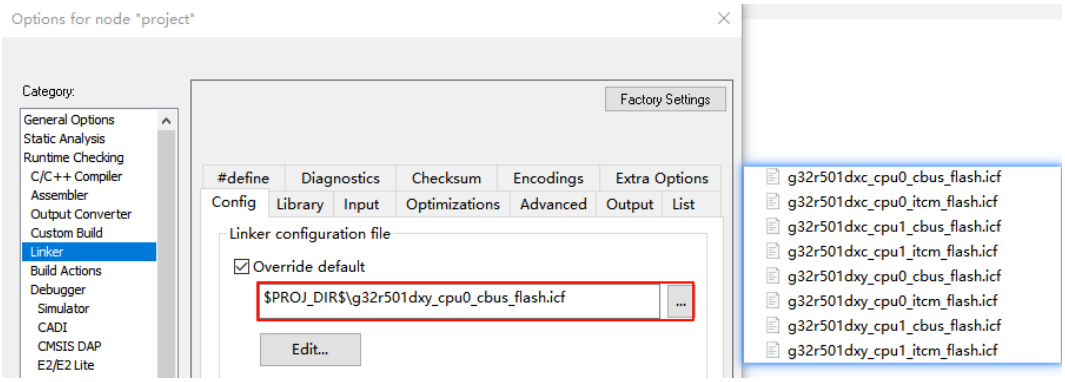
1. Open IAR EW for Arm and create a new project.
2. Select the correct device: General Options → Target → Device, choose G32R501 of the dual-core series (Figure 17).

Figure 17 G32R501 IAR Chip Selection (partial)



3. Configure the **“.icf”** file. Use different “.icf” files for different CPU projects. For common evaluation-board orderables, use “g32r501dxyx7_cpu0_cbus_flash.icf” (CPU0) and “g32r501dxyx7_cpu1_cbus_flash.icf” (CPU1); for the “dxyx8q” package use “g32r501dxyx8q_cpu0_cbus_flash.icf” / “g32r501dxyx8q_cpu1_cbus_flash.icf” (Figure 18).

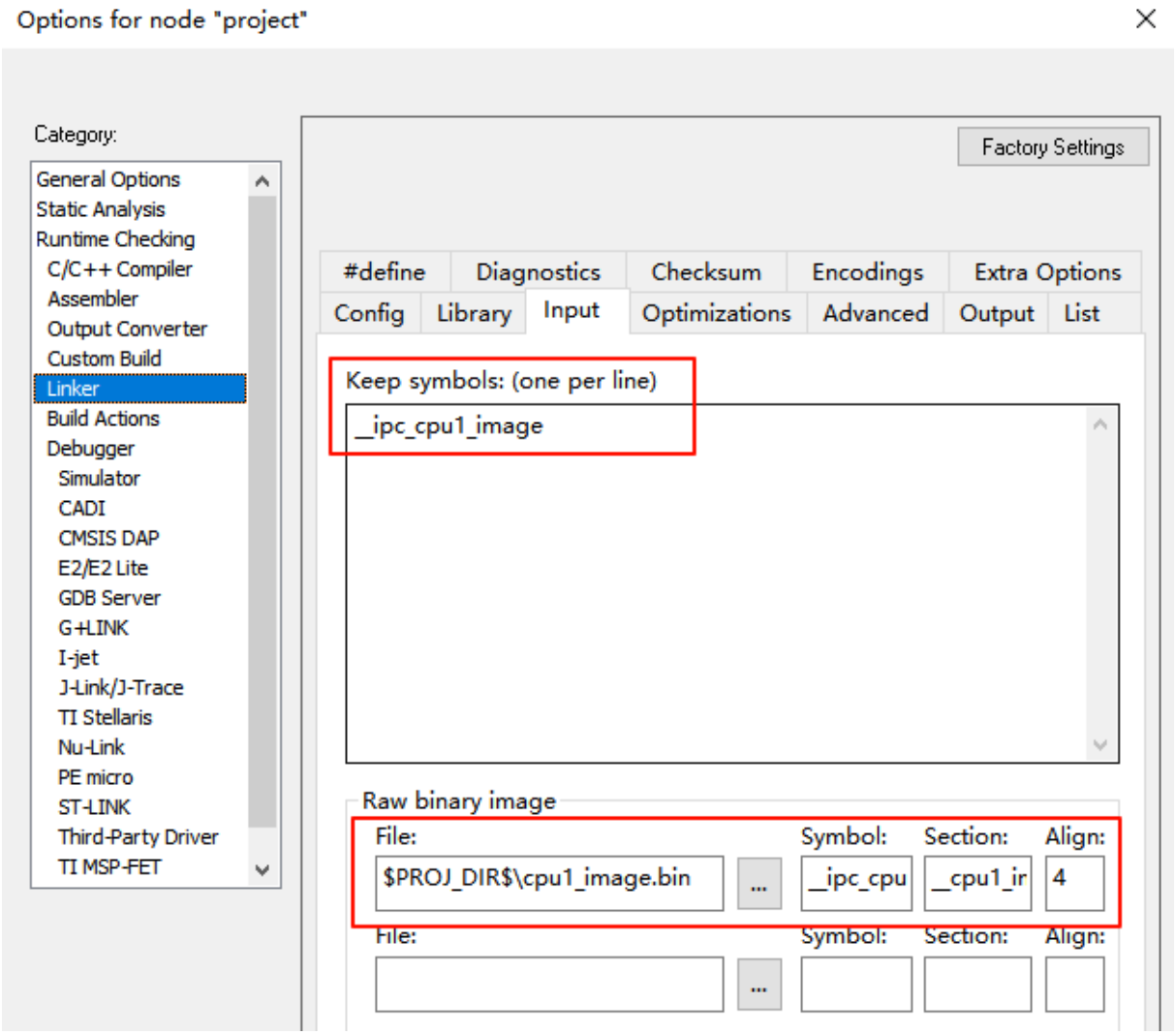
Figure 18 Use Dual-core Configuration



4. Configure the CPU0 project to include the CPU1 image: Linker → Input.

- Keep “ipc_cpu1_image” from being optimized away.
- Load the bin path and related settings (Figure 19).

Figure 19 CPU0 Linker Input Embedding cpu1image.bin



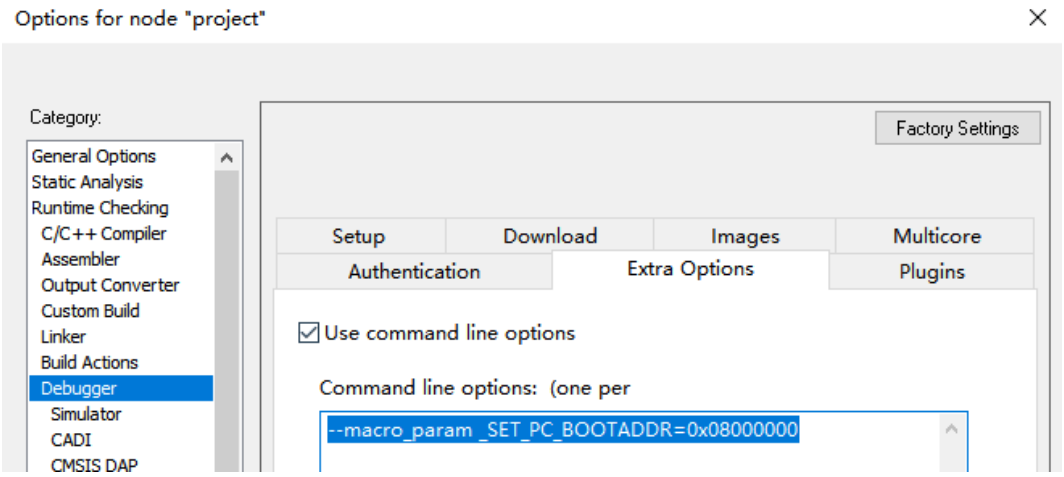
5. Configure the probe and boot address:

- For both projects, select Debugger “CMSIS-DAP”: Debugger → Setup.
- For CPU0, set the boot address: Debugger → Extra Options → enable “Use command line options”, and enter:

“--macro_param _SET_PC_BOOTADDR=0x08000000”

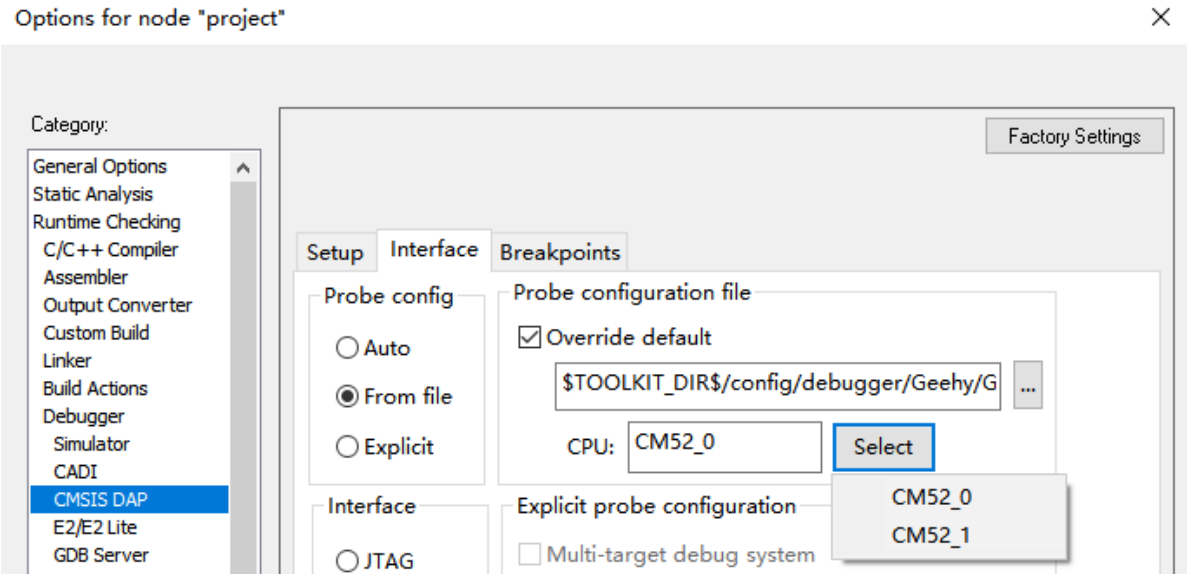
(address must match the actual CPU0 boot address, Figure 20).

Figure 20 Configure CPU0 Project Boot Address



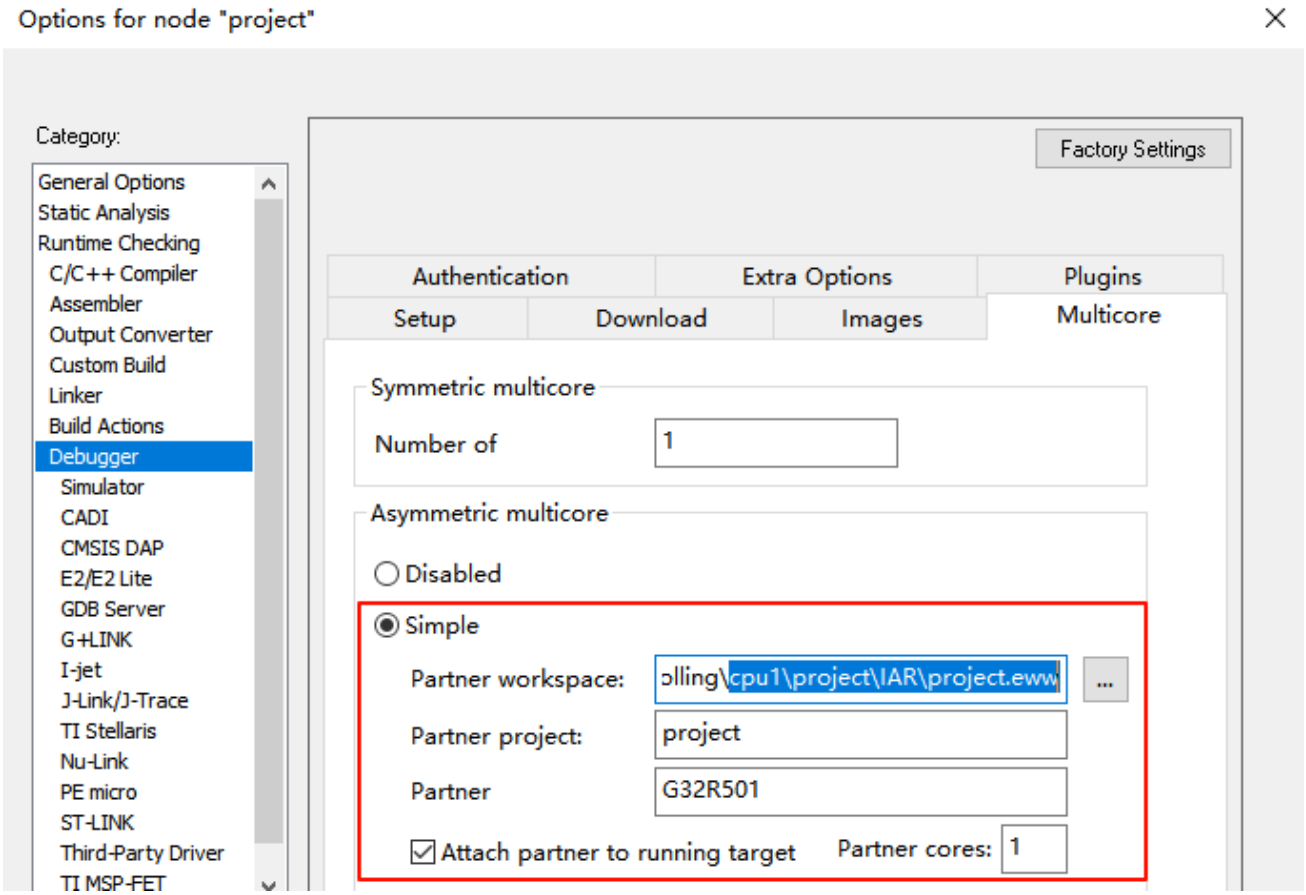
1. Configure the CPU0 AP: CMSIS DAP → Interface → Probe config → “From file” → “CM52_0” (Figure 21).

Figure 21 Configure CPU0/CPU1 Project Debug AP



2. Link the CPU1 project from CPU0: Debugger → Multicore → Asymmetric multicore → “Simple” → select Partner workspace / project / tag (Figure 22).

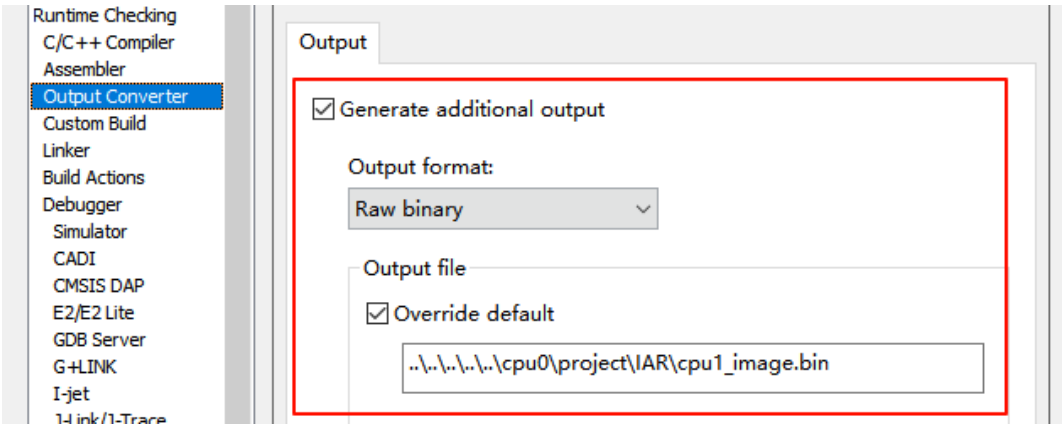
Figure 22 Link CPU1 Project from CPU0



5.2.2 Additional CPU1 Settings

1. Complete device selection, ".icf" settings, and debug settings.
2. Configure bin output to the directory expected by CPU0 (Figure 23).

Figure 23 Configure CPU1 Bin Output Directory

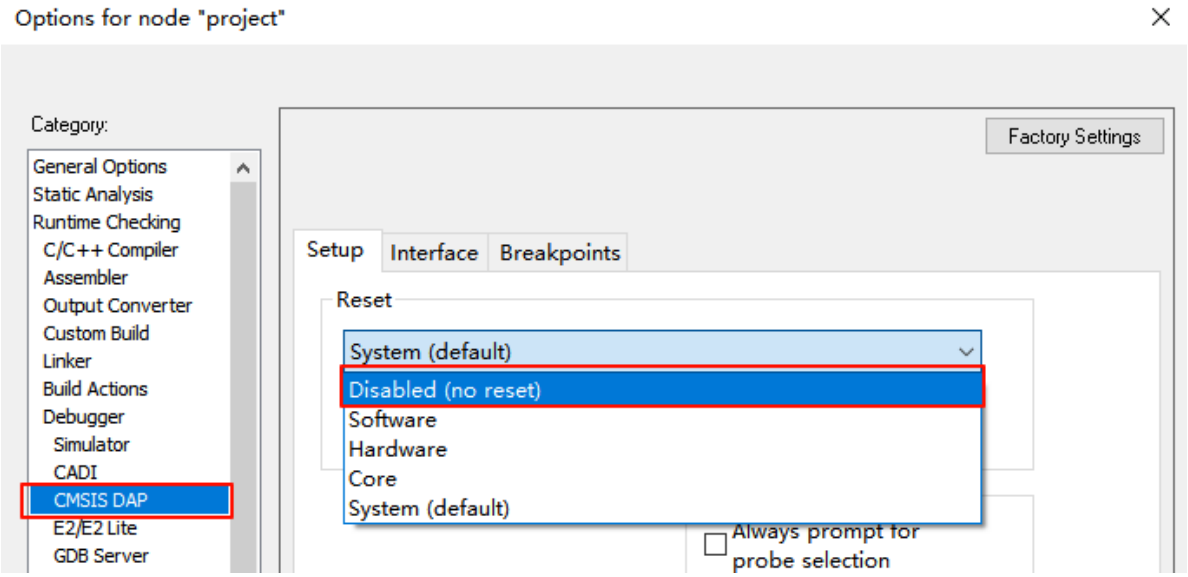


3. Configure the CPU1 AP: CMSIS DAP → Interface → Probe config → "From file" →

"CM52_1".

4. Disable reset on connect for CPU1: CMSIS DAP → Setup → Disabled (no reset) (Figure 24).

Figure 24 Configure CPU1 Connect Without Reset

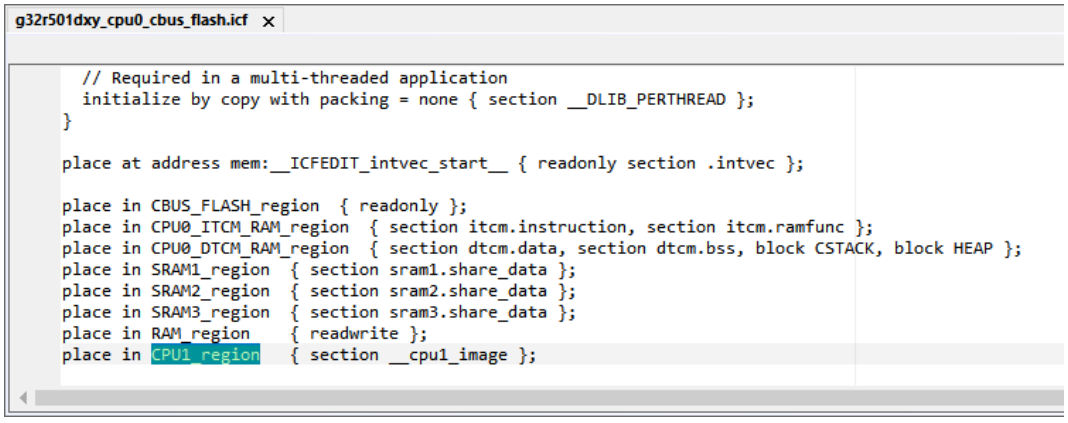


5.2.3 Example `.icf` Files

Example ".icf" files used in this chapter are under "SDK\device_support\g32r501\common\icf\", for example "g32r501dxyx7_cpu0_cbus_flash.icf" and "g32r501dxyx7_cpu1_cbus_flash.icf" (rename similarly for "dxyx8q").

In "g32r501dxyx7_cpu0_cbus_flash.icf", dual-core configuration splits Flash and declares the "cpu1_image" section (Figure 25). Flash allocation refers to Table 3.

Figure 25 CPU1 Image Segment Settings of .icf in CPU0

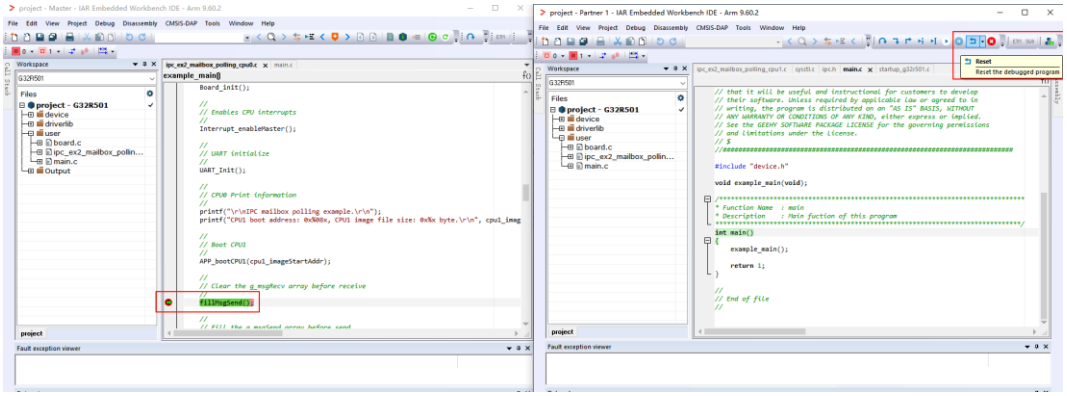


5.2.4 Start Dual-core Debugging

After configuration and a successful build:

1. Start CPU0 debugging; both CPUs start and two debug windows appear.
2. In the CPU0 project, set a breakpoint **after the "APP_bootCPU1(...)" call** and run to it; then connect CPU1 successfully.
3. After CPU1 is connected, press "Reset" in the CPU1 project so CPU1 returns to its start address.
4. Continue dual-core debug as needed (Figure 26).

Figure 26 IAR Dual-core Debug Interface



6 Eclipse Dual-Core Debugging Support

The latest Eclipse can be downloaded from the Eclipse website.

Note: This chapter uses pyOCD + GEEHY-LINK.

6.1 Dual-Core Debugging on Eclipse

Follow AN1126 Chapter 7 to enable pyOCD support for the G32R501 series.

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

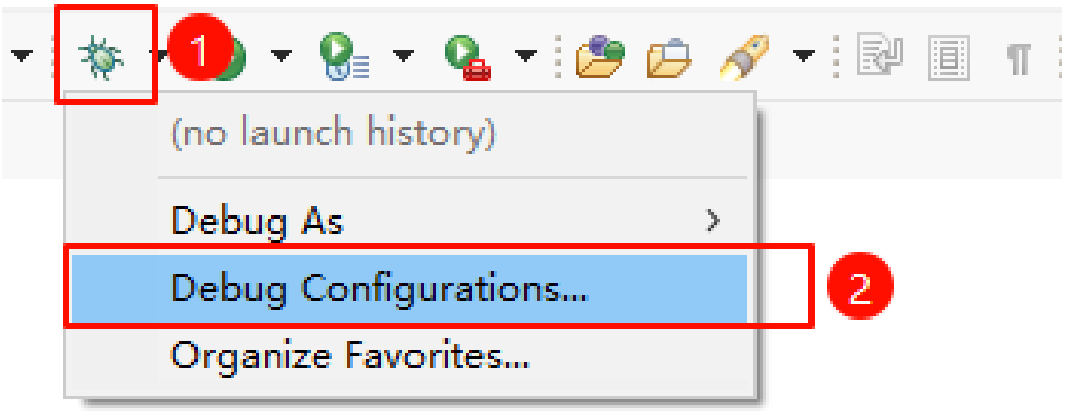
This section provides steps for Eclipse and GEEHY-LINK (WinUSB) with a G32R5xx MCU. Create one project per core. Example programs: “G32R5xx_SDK\driverlib\g32r501\examples\eval\ipc\”.

After importing projects and building successfully, configure debug settings as follows.

6.2.1 Create a Debug Configuration

1. Left-click the Debug icon to show debug configurations.
2. Select “Debug Configurations...”.
3. Select “GDB pyOCD Debugging”.
4. Select “New Configuration” (Figure 27).

Figure 27 New Configuration

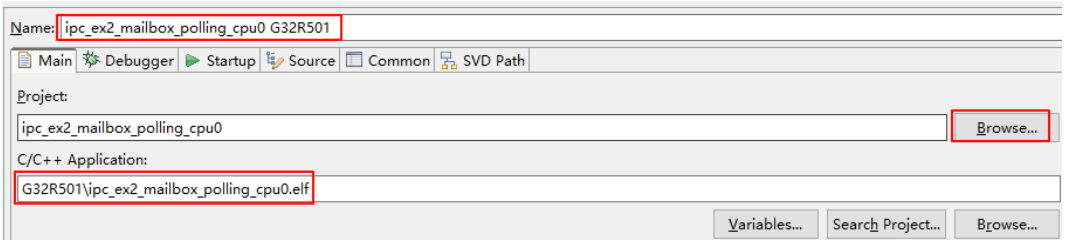


6.2.2 Configure the Main Tab

1. Name the debug configuration.
2. Browse to the corresponding project.

3. Select the ELF, for example “G32R501\ipc_ex2_mailbox_polling_cpu0.elf” (relative or absolute path, Figure 28).

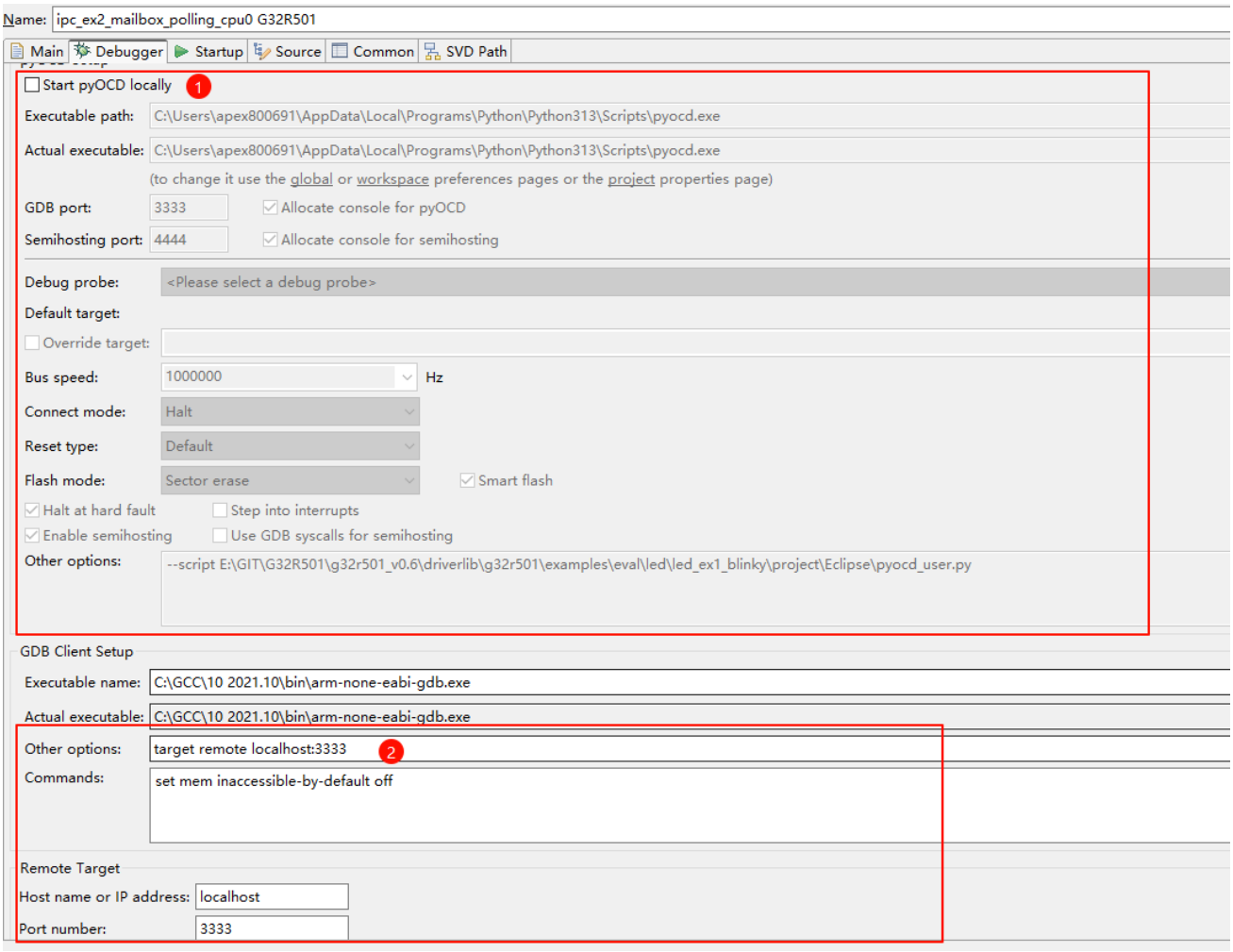
Figure 28 Configure Main



6.2.3 Configure the Debugger Tab

1. Disable starting the pyOCD GDB Server from Eclipse.
2. Point the GDB Client to the core port. Defaults are typically core 0: 3333 and core 1: 3334 (Figure 29).

Figure 29 Dual-core Debugger Tab



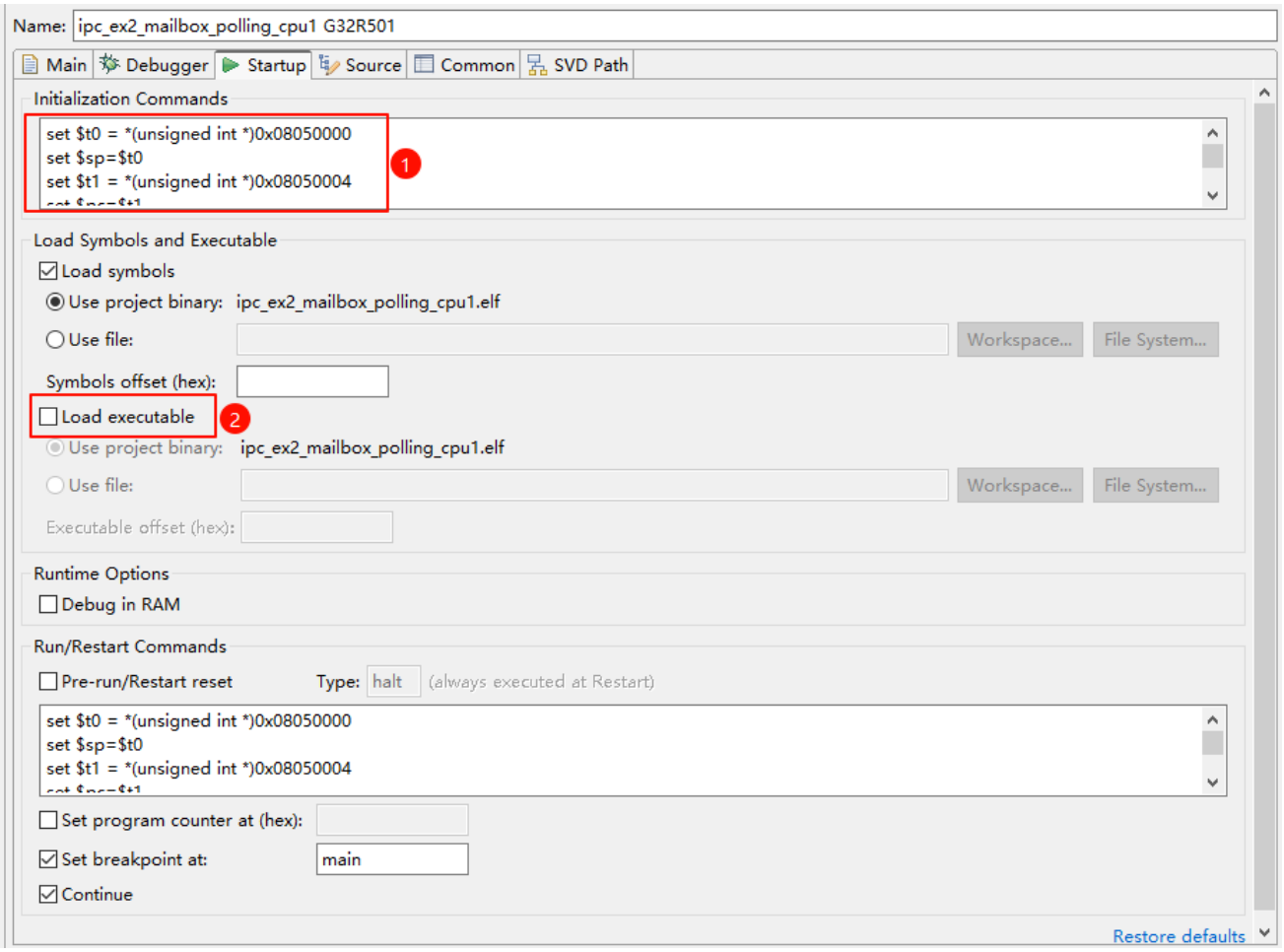
6.2.4 Configure the Startup Tab

- CPU0: Follow the single-core Startup settings in AN1126.
- CPU1:
  - In the Commands tab, remove decrypt sequences and keep only the SP/PC setup sequence. (Note: the example vector-table base is “0x08050000” and must match this project’s link layout. The base is the start of the vector table; SP/PC must be loaded from memory at that address—do **not** assign the base address itself to the registers.)

```
set $sp = *(unsigned int*)0x08050000
set $pc = *(unsigned int*)0x08050004
set $xpsr = $xpsr | (1<<24)
```

- Clear “Load executable” (Figure 30).

Figure 30 Startup Tab



6.2.5 Start pyOCD gdbserver and Dual-core Debugging

Start pyOCD gdbserver from a terminal:

```
pyocd gdbserver
```

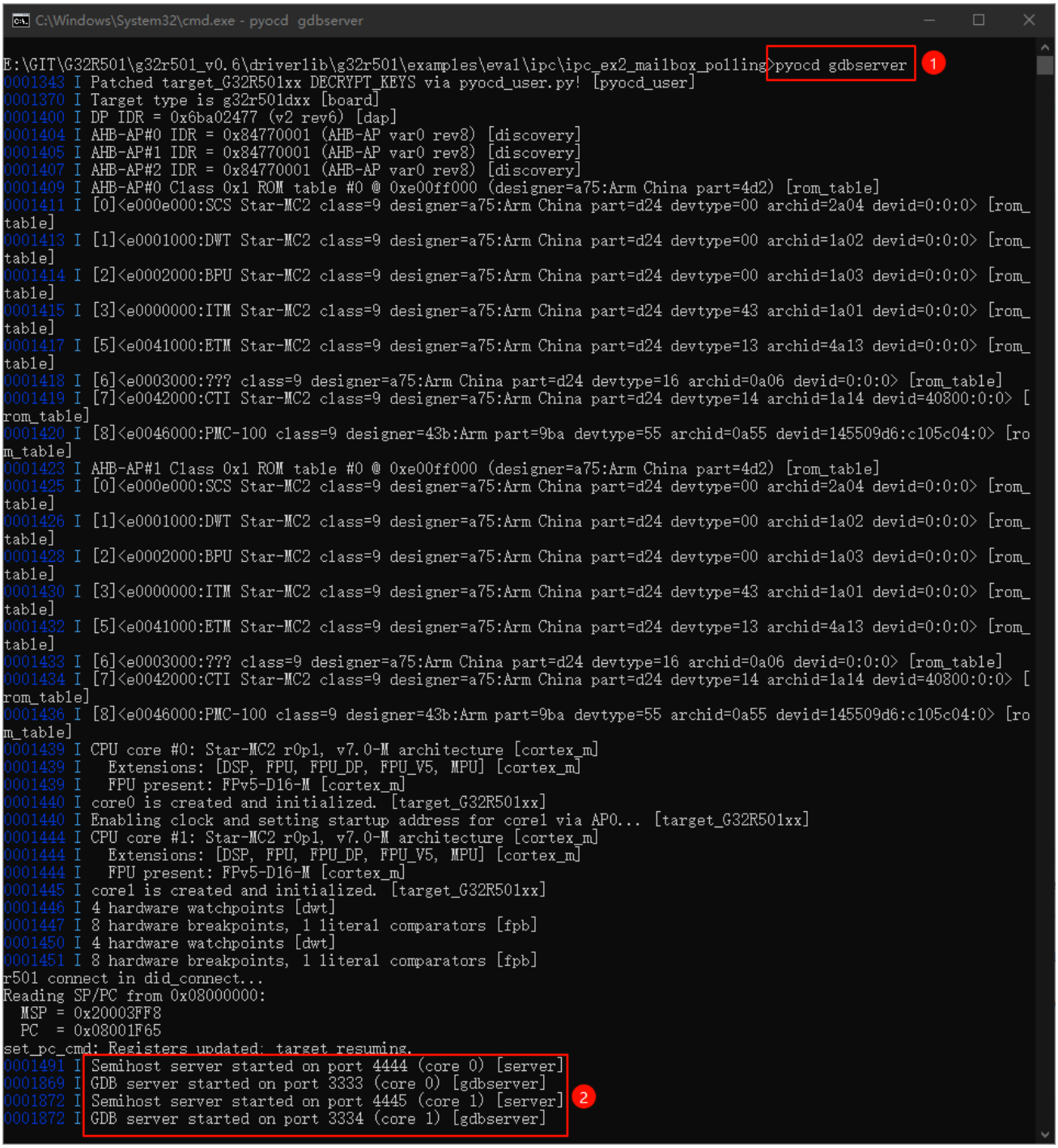
The working directory should contain files under “SDK/device_support/g32r501/common/pyOCD/”, including:

- Dual-core “pyocd.yaml”, for example:

```
target_override: g32r501dxx
frequency: 8000000  # Set 8 MHz SWD default for all probes
session:
  enable_multicore_debug: true
  persist: true
```

- “pyocd_user.py” (Figure 31).

Figure 31 Start pyocd gdbserver from Terminal



Start both debug sessions in Eclipse:

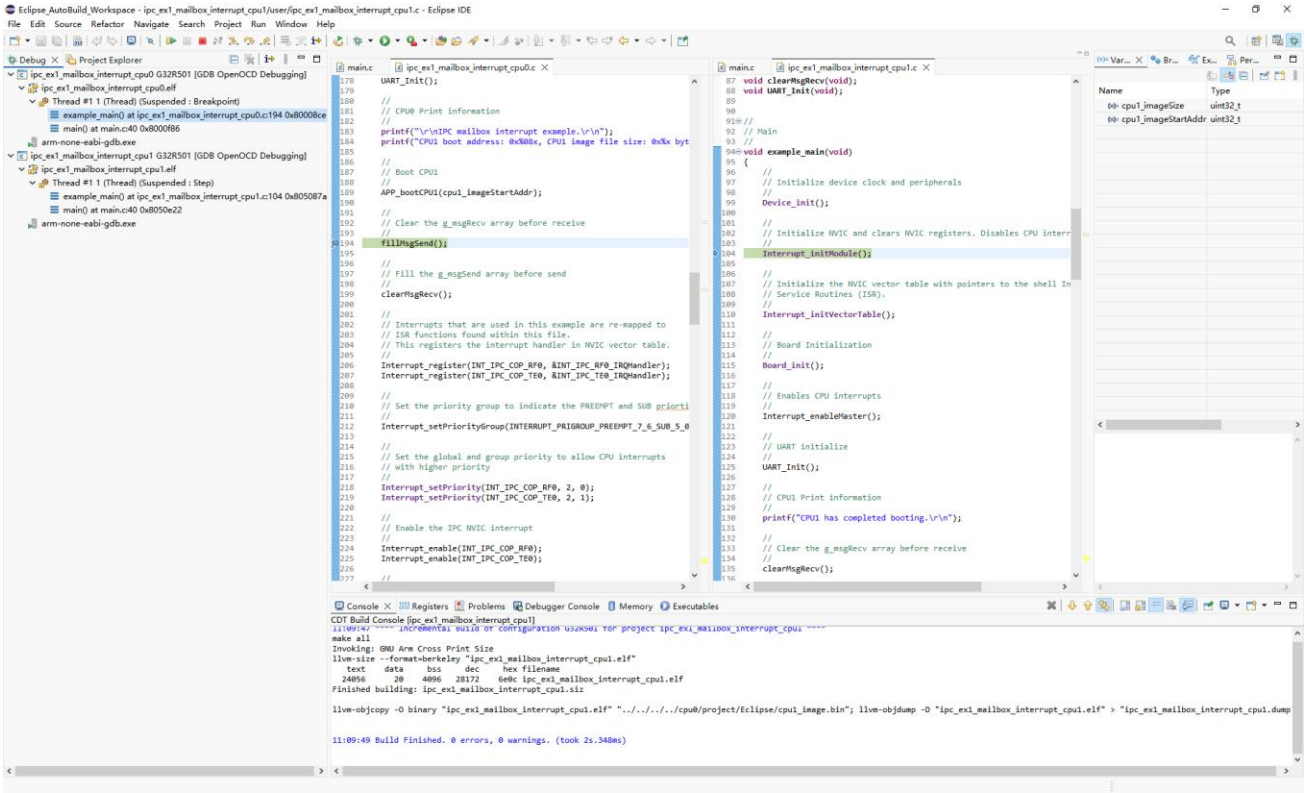
1. Start CPU0 debugging and set a breakpoint **after “APP_bootCPU1(cpu1_imageStartAddr);”**.

2. Start CPU1 debugging.

3. Control each core via its thread:

- Click the cpu0 thread to control CPU0 debug.
- Click the cpu1 thread to control CPU1 debug (Figure 32).

Figure 32 Eclipse Dual-core Debug Interface



7 Revision

Table 4 Document revision

Date	Version	Description
2025.1	1.0	● Initial release
2025.4	1.1	● Added Eclipse dual-core debugging support
2026.8	1.2	● Applicable products limited to G32R501Dx dual-core ● Linker script names aligned to current “g32r501dxy_” ● IAR examples updated to “g32r501dxyx7_” / “dxyx8q_.icf” ● Fixed “r501_dbg.ini” typo ● Updated dual-core debug screenshots ● Clarified Eclipse CPU1 Startup SP/PC must dereference the vector table ● Corrected document formatting

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 “TM” 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.