AN14367

Debugging Cortex-A U-Boot and Native RTOS on i.MX 8M Plus and i.MX 93 EVKs

Rev. 1.0 — 1 October 2024 Application note

Document information

Information	Content
Keywords	AN14367, OpenOCD, Cortex-A debug, GUI, i.MX 93 EVK, i.MX 8M Plus, GDB, U-Boot debugging, RTOS debugging, Native RTOS, JLink, JTAG
Abstract	This application note describes how to debug the U-Boot and Native RTOS on Cortex A Core by using OpenOCD together with the command-line tool GDB or GUI tool Eclipse on i.MX 93 and i.MX 8M Plus EVK

1 Introduction

This application note focuses on the NXP MPU Cortex-A Core debugging method and flow. It provides two methods, one is to use the command-line debugging tool (GDB), the other one is to use the GUI debugging tool (Eclipse in this AN), and takes U-Boot and RTOS debugging as examples. There is no fixed binding relationship, and projects debugged using the JTAG flow can also be debugged using J-Link and vice versa.

i.MX 8M Plus EVK and i.MX 93 EVK provide a local JTAG port (a standard 10-pin JTAG header on the board is used for local debug, it is paralleled with JTAG from the FTDI chip) and a remote JTAG port (the A-bus of FT4232 is numerated as JTAG), so this AN provides two variants of hardware connection: a local JTAG port with the JLink debug probe or a remote debug port used directly.

This AN applies to:

- i.MX 8M Plus EVK and i.MX93 series EVK debugging
- U-Boot debugging
- With/without J-link debugging
- Cortex-A core RTOS debugging
- OpenOCD debugging
- GDB debugging
- OpenOCD + Eclipse debugging

2 Definitions, acronyms, and abbreviations

Table 1 describes the definitions, acronyms, and abbreviations used in this application note.

Table 1. Definitions, acronyms, and abbreviations

Acronyms	Meaning
AN	Application Note
BSP	Board Support Package
EVK	Evaluation Kit
GDB	GNU Project Debugger
GUI	Graphical User Interface
JTAG	Joint Test Action Group
MPSSE	Multi-Protocol Synchronous Serial Engine
OpenOCD	Open On-Chip Debugger
RTOS	Right Real-Time Operating System

3 Hardware and software setup

This chapter describes the software and hardware required for U-Boot and RTOS debugging that are to be prepared before proceeding to the next step.

3.1 Hardware materials

- For i.MX 8M Plus EVK
  - i.MX 8M Plus EVK board
  - Host PC with Ubuntu18/Ubuntu 20/ Ubuntu 22

  - i.MX 8M Plus board power cable
  - micro-USB debug cable
  - USB-Type C cable
  - Micro SD card
  - (Only for J-Link debugging flow) A SEGGER J-Link debug probe
- For i.MX 93 EVK
  - i.MX93EVK board
  - Host PC with Ubuntu18/Ubuntu 20/ Ubuntu 22
  - i.MX93 EVK board power cable
  - USB-Type C debug cable
  - USB-Type C cable
  - Micro SD card
  - (Only for J-Link debugging flow) A SEGGER J-Link debug probe

3.2 Hardware connection

If a remote JTAG port is used, the following connection is enough for OpenOCD debug:

1. Insert the Micro SD card to the card slot on the EVK board (in this process, the image is booted from the Micro SD card);
2. Connect the USB debug cable to the debug port on the EVK board for debugging, connect the other end to the Host PC with Ubuntu. Do not use a USB hub to connect to avoid signal interference;
3. Connect the USB-Type C cable to the USB port on the EVK board for flashing the image, connect the other end to the host PC with Ubuntu;
4. Connect the power cable to the power interface.
5. (Optional) If J-Link is used, an extra step is to connect the J-Link probe to a local JTAG port. For details, refer to Figure 1

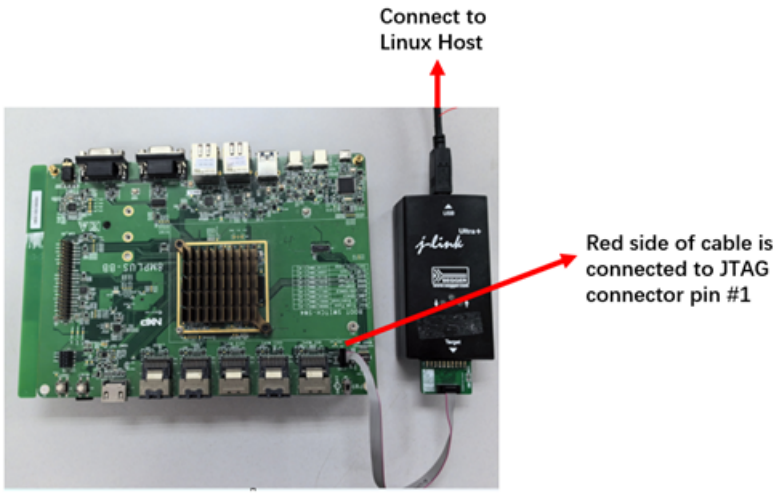

Figure 1. J-Link hardware connection on i.MX 8M PLUS board

3.3 Software setup

A Linux PC host is needed to set up the debugging environment. Ubuntu 20 or Ubuntu 22 is recommended because the example in this AN has been verified on them.

3.3.1 Software setup on Ubuntu PC

Refer to Table 2 to install the following software on an Ubuntu host PC: bcu, uuu, OpenOCD, and Tabby (or other serial console tools).

Install the GDB client for command-line debugging or Eclipse for GUI debugging. For J-Link debugging, install the J-Link probe driver and extra patch:

Table 2. Software requirements on Ubuntu PC

BCU	BCU is used for board debug interface enablement. The current version is 1.1.92. For Ubuntu_18: `$ wget https://github.com/nxp-imx/bcu/releases/download/bcu_1.1.92/bcu_Ubuntu18 --no-check-certificate` For Ubuntu_20: `$ wget https://github.com/nxp-imx/bcu/releases/download/bcu_1.1.92/bcu_Ubuntu20 --no-check-certificate` For Ubuntu_22: `$ wget https://github.com/nxp-imx/bcu/releases/download/bcu_1.1.92/bcu_Ubuntu22` The latest bcu is available at the following link: https://github.com/nxp-imx/bcu/releases.
OpenOCD	OpenOCD is used for debugging remote targets that work with GDB. There are cfg files for i.MX 8M Plus, but for i.MX 93, the patch must be applied. `$ git clone https://github.com/openocd-org/openocd.git` `$ sudo apt-get install make libtool pkg-config autoconf automake texinfo` `$ sudo apt-get install libusb-1.0-0-dev libftdi-dev libftdi1-2` `$ sudo apt-get install autotools-dev build-essential swig cmake python-dev libconfuse-dev libboost-all-dev libtool-bin libjaylink-dev` `$ cd openocd` `$ git checkout 12ff36bd19e4f25dd7505c46a77d9f2c47dc350a` `$./bootstrap` `$./configure --enable-ftdi --enable-openjtag --enable-jlink --prefix=/usr/local/share` `$ make` **`$ sudo make install`** Save the following debugging scripts from the OpenOCD 93 debugging scripts and OpenOCD 8MP debugging scripts to the specified directory. ***Note:*** *By default, OpenOCD debugging scripts are installed in the target or board directory. Debugging script `nxp_imx93-evk-reset.cfg` includes an additional initial reset that is needed when a rebuilt image is downloaded and debugged.* `# cp -a imx93.cfg openocd/tcl/target/imx93.cfg` `# cp -a nxp_imx93-evk.cfg openocd/tcl/board/nxp_imx93-evk.cfg` `# cp -a nxp_imx93-evk-reset.cfg openocd/tcl/board/nxp_imx93-evk-reset.cfg` `# cp -a nxp_imx8mp-evk.cfg openocd/tcl/board/nxp_imx8mp-evk.cfg` `# cp -a imx8mp.cfg openocd/tcl/target/imx8mp.cfg`
GDB	GDB is the main debugging tool that can be installed by using the `apt` command. `$ sudo apt install -y gdb-multiarch`

Table 2. Software requirements on Ubuntu PC*...continued*

uuu	uuu is used for the BSP image or bootloader image downloading, the current version is 1.5.125. `$ wget https://github.com/nxp-imx/mfgtools/releases/download/uuu_1.5.125/uuu` `$ chmod +x uuu`
Eclipse IDE	Eclipse IDE is the GUI platform, download, and install Eclipse IDE on Ubuntu. The current test software version is 4.29.0. The download link is：https://www.eclipse.org/downloads/.
Tabby	Tabby is a serial console tool on Ubuntu. The download link is: https://tabby.sh/. You can also use your favorite console tool.
J-Link Debugging flow (Optional):	
J-Link driver and patch	1. Download the J-Link driver (J-link Version: V7.96) from https://www.segger.com/downloads/jlink/. 2. Install the driver: `# dpkg -i JLink_Linux_V796f_x86_64.deb`. 3. Download the SEGGER J-Link patch for i.MX 93 and save it in directory `/opt/SEGGER/JLink`. ***Note:*** *For step-to-step guidelines, refer to `readme` in the downloaded folder.*
libjaylink	If “libjaylink” is not present during building, install the library manually by following the steps below, then build OpenOCD again: `$ git clone https://gitlab.zapb.de/libjaylink/libjaylink.git` `$ cd libjaylink` `$ ./autogen.sh` `$ ./configure` `$ make && sudo make install`

4 U-Boot OpenOCD debugging

This chapter introduces the method of using JTAG for U-Boot debugging.

4.1 OpenOCD Debugging with JTAG Introduction

Most i.MX 8 and i.MX 9 series EVKs are equipped with a Joint Test Action Group (JTAG) interface and an FTDI chip allowing debugging for Cortex-A core and Cortex-M core. Open On-Chip Debugger (OpenOCD) is a tool that utilizes the JTAG interface to perform chip debugging. J-link is not needed under this combination. For i.MX 93 EVK (taken as an example), the general debugging process outline is:

1. The i.MX 93 EVK has an FTDI 4232H chip that has Multi-Protocol Synchronous Serial Engine (MPSSE) supporting USB signals to JTAG signals;
2. bcu controls RC_nSEL to enable FTDI chip to transmit JTAG signal through USB debug cable;
3. OpenOCD starts running as a server waiting for the connections from GDB or Telnet clients and handling the commands issued through those channels;
4. Users use the GDB client to connect to the server, through the corresponding port(3333 for Cortex-A on i.MX 93 by default) for debugging.

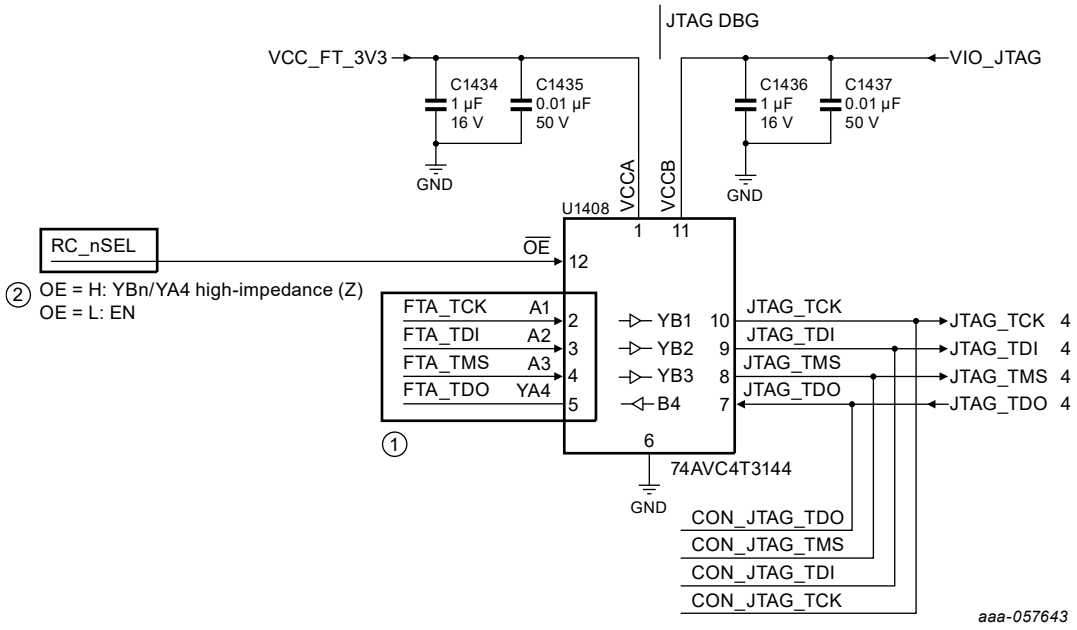

Figure 2. Remote debug interface on i.MX 93 EVK

4.2 Debugging U-Boot with Eclipse IDE

This chapter introduces the preparation work before starting, which is necessary. By executing it step by step, unnecessary troubles can be avoided. The basic steps of debugging are described in the later sections.

4.2.1 Software requirements

Table 3. Software requirements

Build (get) the Linux BSP image	Linux BSP release L6.1.36_2.1.0 is used in this application note. Specifically: • i.MX 8M Plus: `imx-image-full-imx8mpevk.wic` • i.MX 93: `imx-image-full-imx93evk.wic` Download the official release BSP from nxp.com or refer to i.MX Yocto Project User's Guide (nxp.com) to build on a host Ubuntu PC by yourself.
Download the Linux BSP image	Method 1: download it to the board using uuu (change the bootmode to serial download mode and use a USB-Type C cable): `$ sudo ./uuu -b sd imx-image-full-imx8mpevk.wic # for 8mp` `$ sudo ./uuu -b sd imx-image-full-imx93evk.wic # for 93` Method 2: flash it to the SD card directly: `$ sudo dd if=.wic of=/dev/sd[x] bs=1M status=progress conv=fsync` ***Note:*** *Check your card reader partition and replace sd[x] with your corresponding partition.*
Setup U-Boot	U-Boot is the project that is debugged. Before building the U-Boot: 1. Use your cross-compile toolchain, refer to i.MX Linux User's Guide (nxp.com) for more toolchain information. 2. Choose one of two defconfig files (for 8MP or for 93). `$ git clone https://github.com/nxp-imx/uboot-imx.git`

Table 3. Software requirements*...continued*

	`$ cd uboot-imx` `$ git checkout lf-6.1.36-2.1.0` `$ source /opt/fsl-imx-internal-xwayland/6.1-langdale/environment-setup-armv8a-poky-linux` `$ make imx8mp_evk_defconfig # for 8MP` `$ make imx93_11x11_evk_defconfig # for 93` `$ make`
Build the bootloader	Build `flash.bin` using the built files from the U-Boot project in the imx-mkimage project, including: • `u-boot.bin` • `spl/u-boot-spl.bin` The Building must be done on the host Ubuntu PC. The command in imx-mkimage is: `$ make SOC=iMX8MP flash_evk # for 8mp` `$ make SOC=iMX9 REV=A1 flash_singleboot_m33 # for 93` Refer to: i.MX Linux User's Guide for more `flash.bin` building information.
Download the bootloader	Method 1: Download it to the board by using uuu (change the bootmode to serial download mode and use a USB-Type C cable): `$ sudo ./uuu -b sd flash.bin` Method 2: Flash it to the SD card directly: `$ sudo dd if=flash.bin of=/dev/sd[x] bs=1k seek=32` ***Note:*** *Check your card reader partition and replace sd[x] with your corresponding partition.*

4.2.2 Eclipse OpenOCD Configuration for U-Boot debugging

In terms of usage, OpenOCD can be used in the terminal as an independent tool or integrated into certain GUI software, for example, Eclipse. Compared to terminal debugging, the benefits of integrating into a GUI include:

- Viewing the call stack and variables more intuitively;
- Setting the breakpoints and single-step debug in the code directly;
- Using the advantages of the GUI platform to view and modify the code more conveniently.

The following are Eclipse OpenOCD Configuration steps:

1. Open Eclipse IDE, select a directory as a workspace, then click **Launch.**

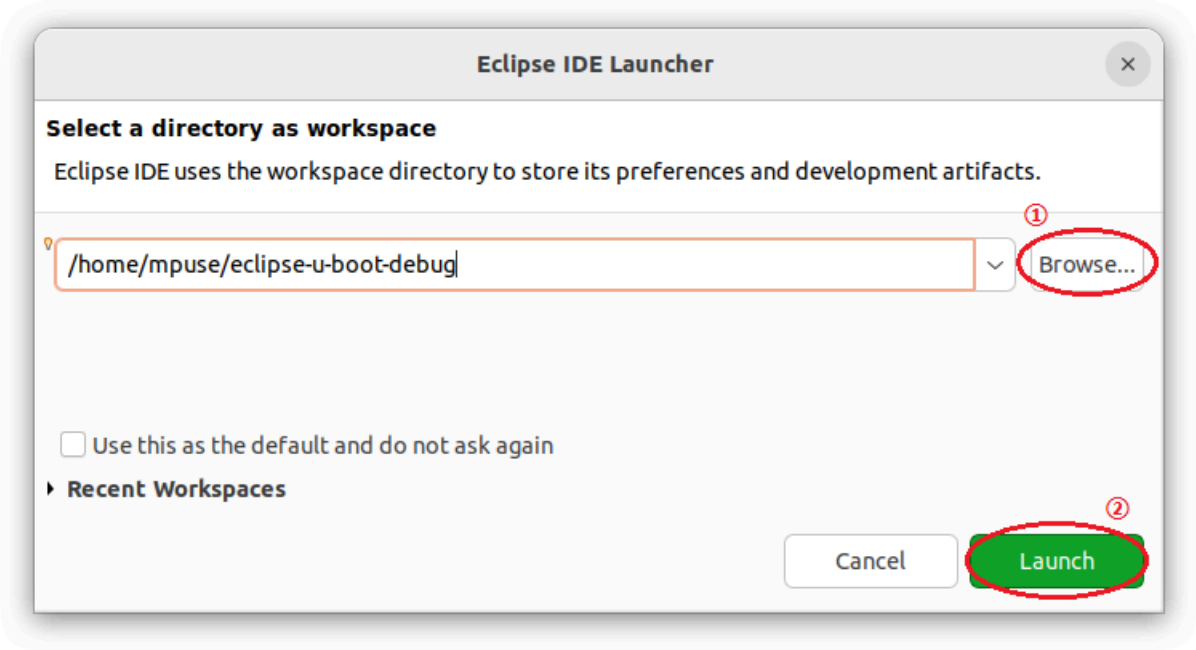

Figure 3. Launch a workspace

2. In the main menu, left-click **File** and then click **Open Projects from Files Systems**.

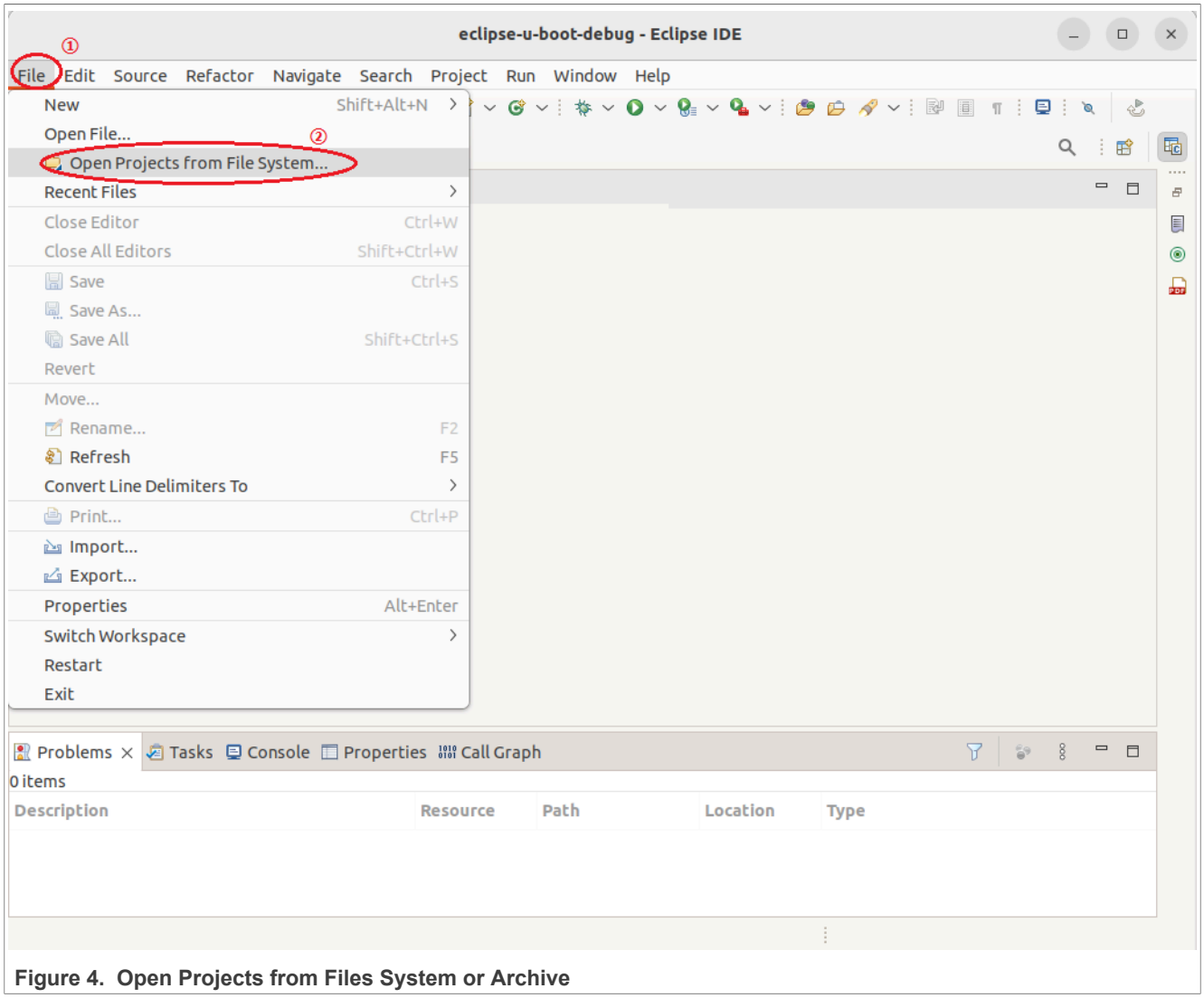

Figure 4. Open Projects from Files System or Archive

3. On the **Import Projects from Files System or Archive** tab, click **Directory…** . To browse the folder, choose the path of the U-Boot source code. Confirm that the project already includes this uboot-imx folder, then click **Finish**.

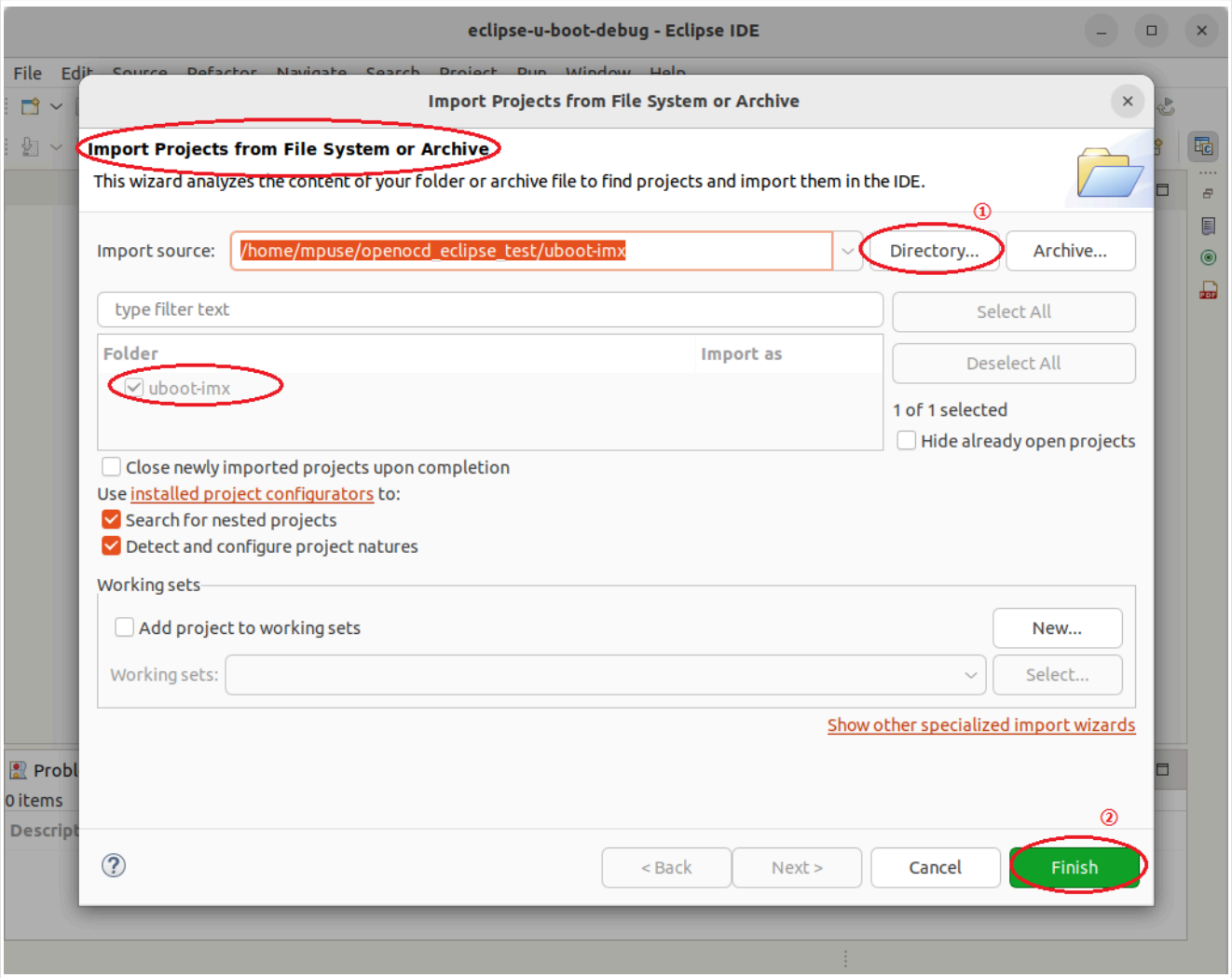

Figure 5. Choose uboot-imx source code directory

4. Click **Run**, then click **Debug Configurations**.

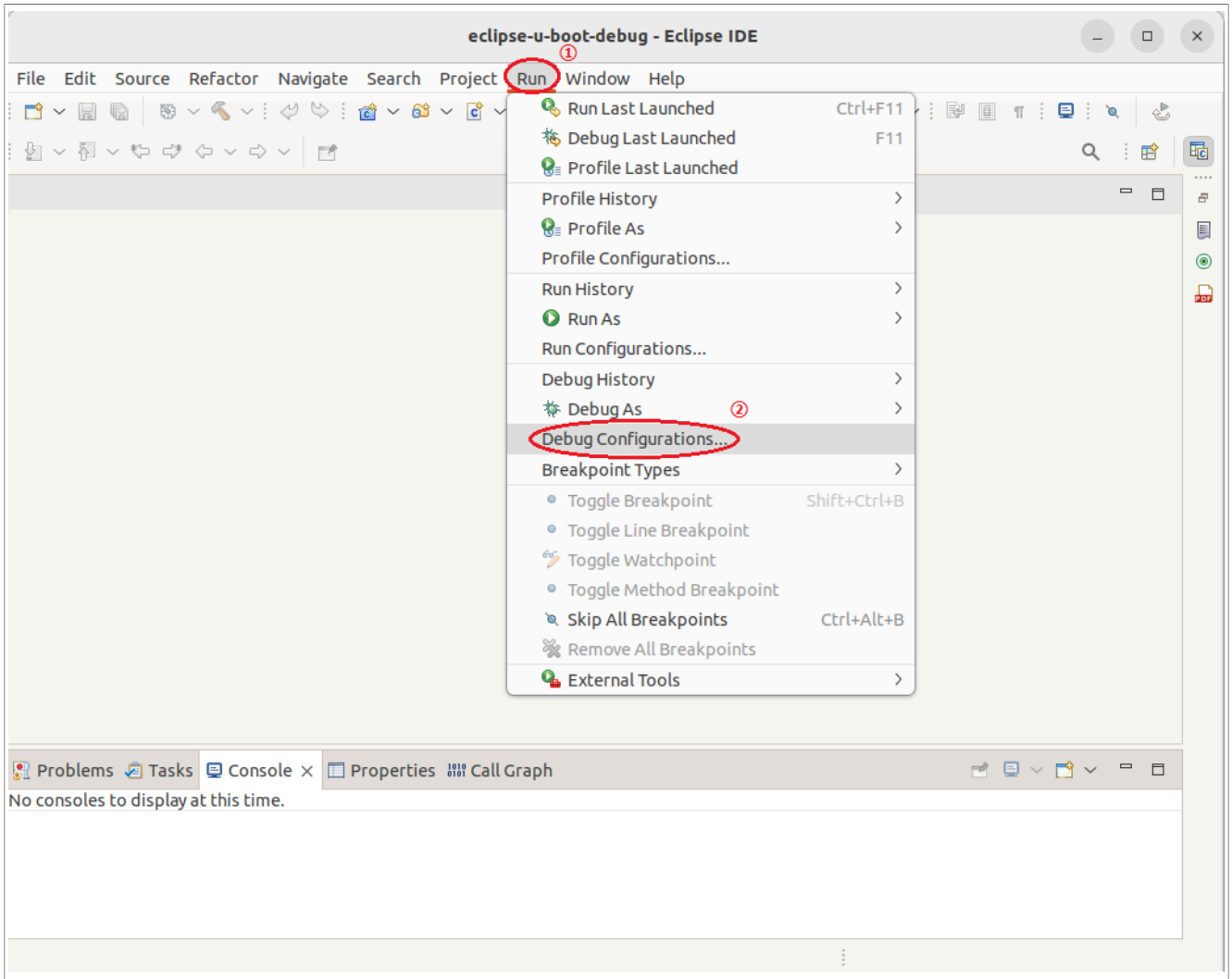

Figure 6. Open the Debug Configurations tab

5. Create a **GDB OpenOCD Debugging** interface and name the configuration in **Name** as shown in Figure 7
   In the **Main** tab in **Project,** name the project，in **C/C++ Application**, select the U-Boot file compiled from the U-Boot source code.

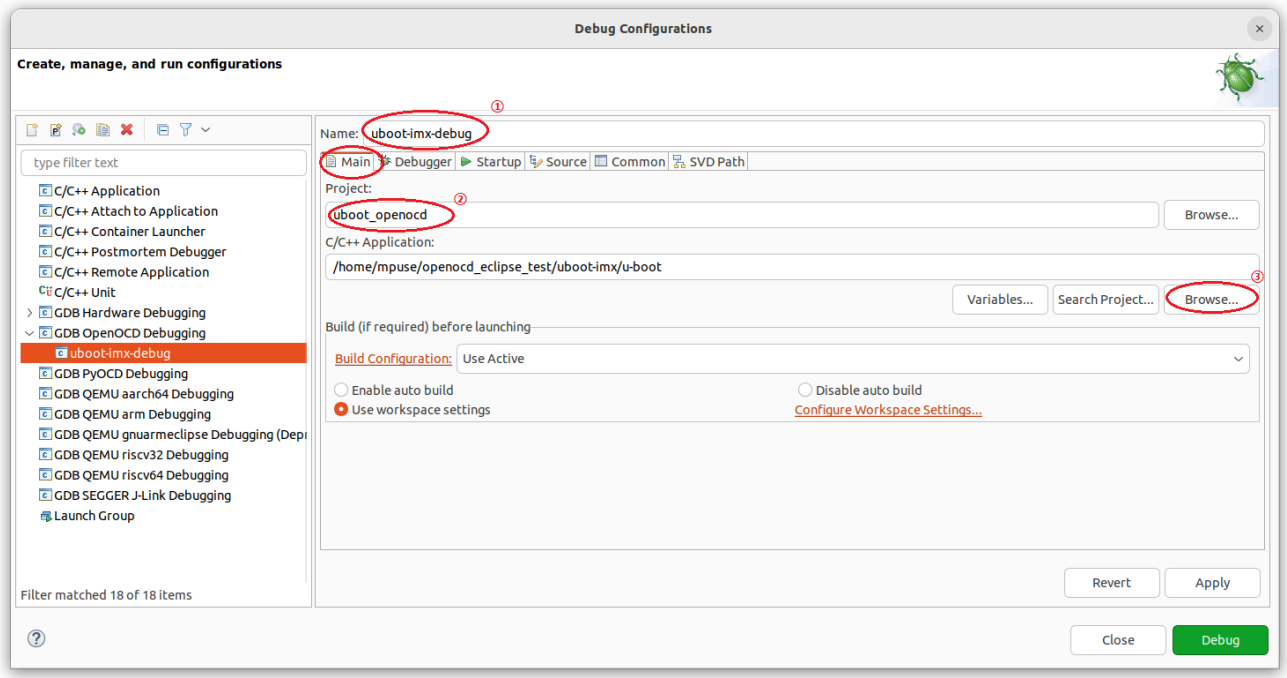

Figure 7. Configure the main tab

6. In the **Debugger** tab, OpenOCD setup section, click the **Browse…** button， select the built openocd location. Enter the following commands in **Config Options**.

   Note: *Modify the command according to your actual path*：

   For i.MX 8M Plus

```
-f
/home/mpuse/openocd_eclipse_test/openocd/tcl/interface/ftdi/imx8mp-evk.cfg -f
/home/mpuse/openocd_eclipse_test/openocd/tcl/board/nxp_imx8mp-evk.cfg -s
/home/mpuse/openocd_eclipse_test/openocd/tcl/
```

   For i.MX 93

```
-f
/home/mpuse/openocd_eclipse_test/openocd/tcl/interface/ftdi/imx93-evk.cfg -f
/home/mpuse/openocd_eclipse_test/openocd/tcl/board/nxp_imx93-evk.cfg -s
/home/mpuse/openocd_eclipse_test/openocd/tcl/
```

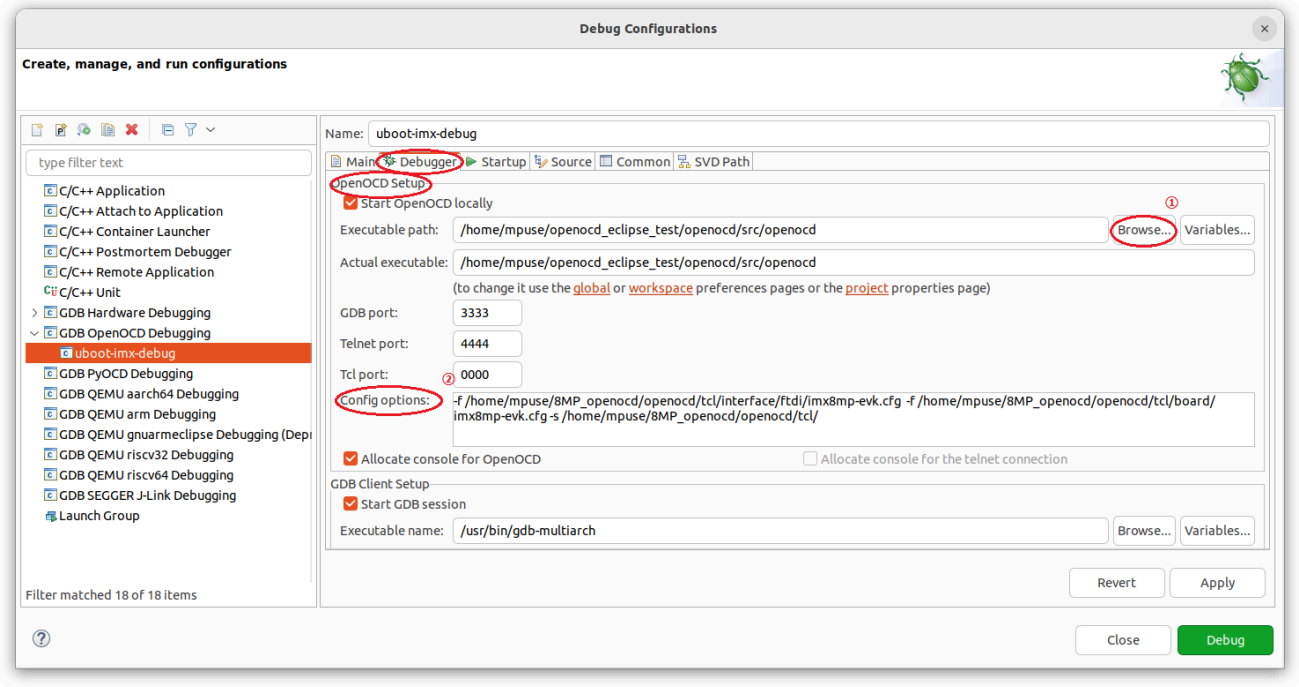

Figure 8. Configure the Debugger tab - 1

7. In the **GDB Client Setup** section, click **Browse** to choose the GDB (use `which gdb-multiarch` to check the gdb path)

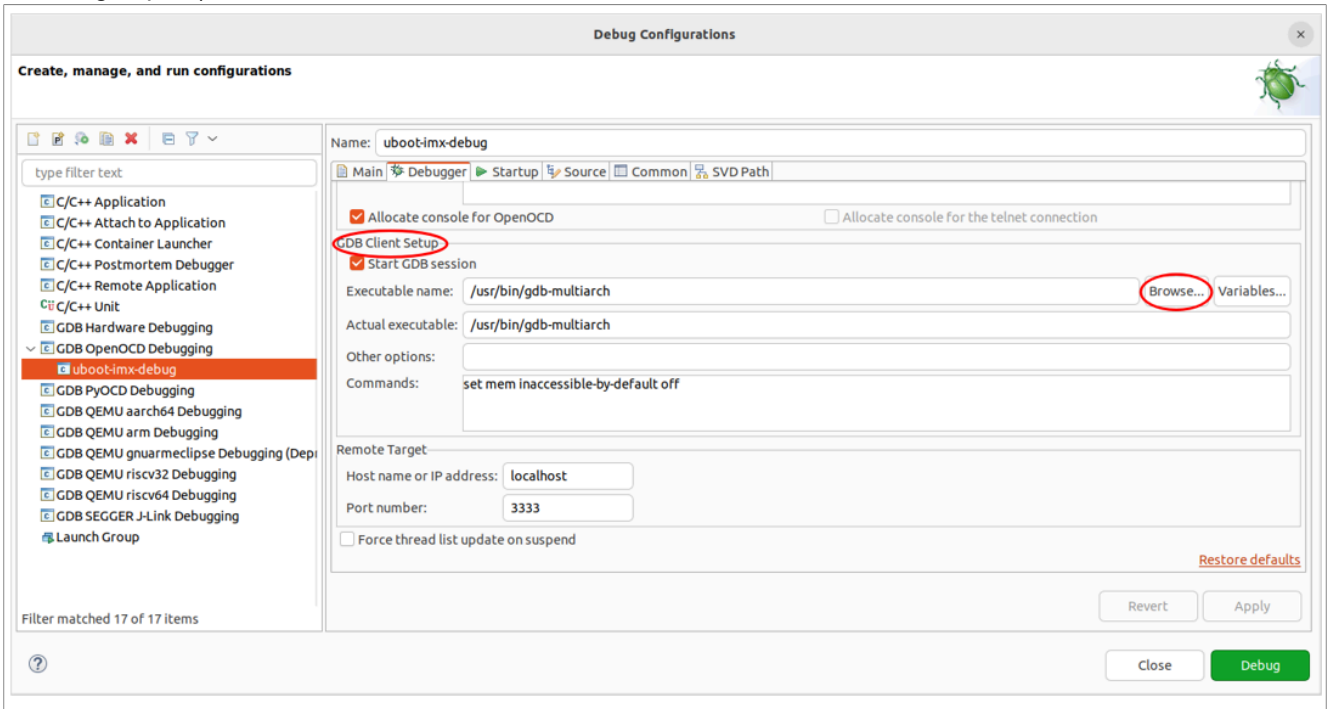

Figure 9. Configure the Debugger tab - 2

8. In the **Startup** tab, click **load Symbol and Executable**.
   - Make sure that **Initial Reset** has been selected, and the Type is **init**.
   - Check the **Load Symbols**, then use the U-Boot compiled from the U-Boot source folder.
   - The **Symbol offset(hex)** offset is necessary. The value can be got at the U-Boot stage:

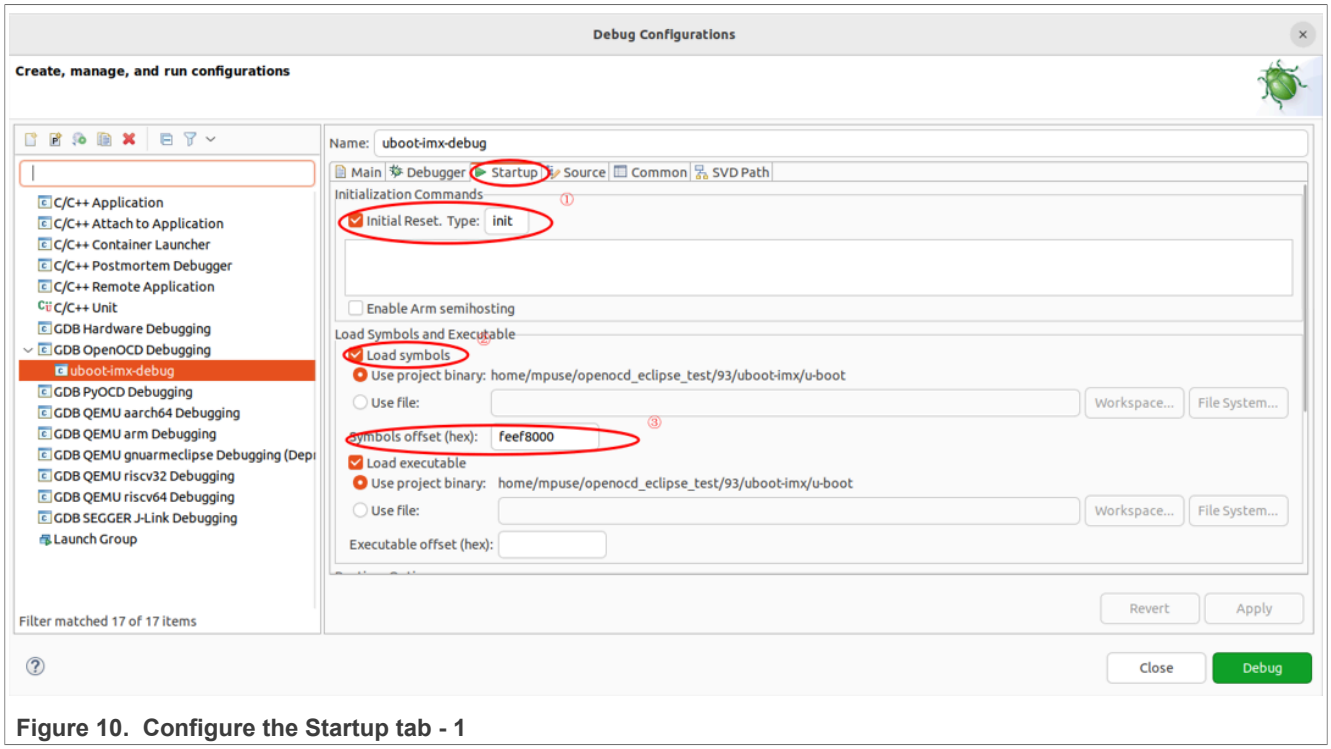

Figure 10. Configure the Startup tab - 1

9. Type `bdinfo` at the U-Boot stage to check the **relocaddr** and use it as an offset.

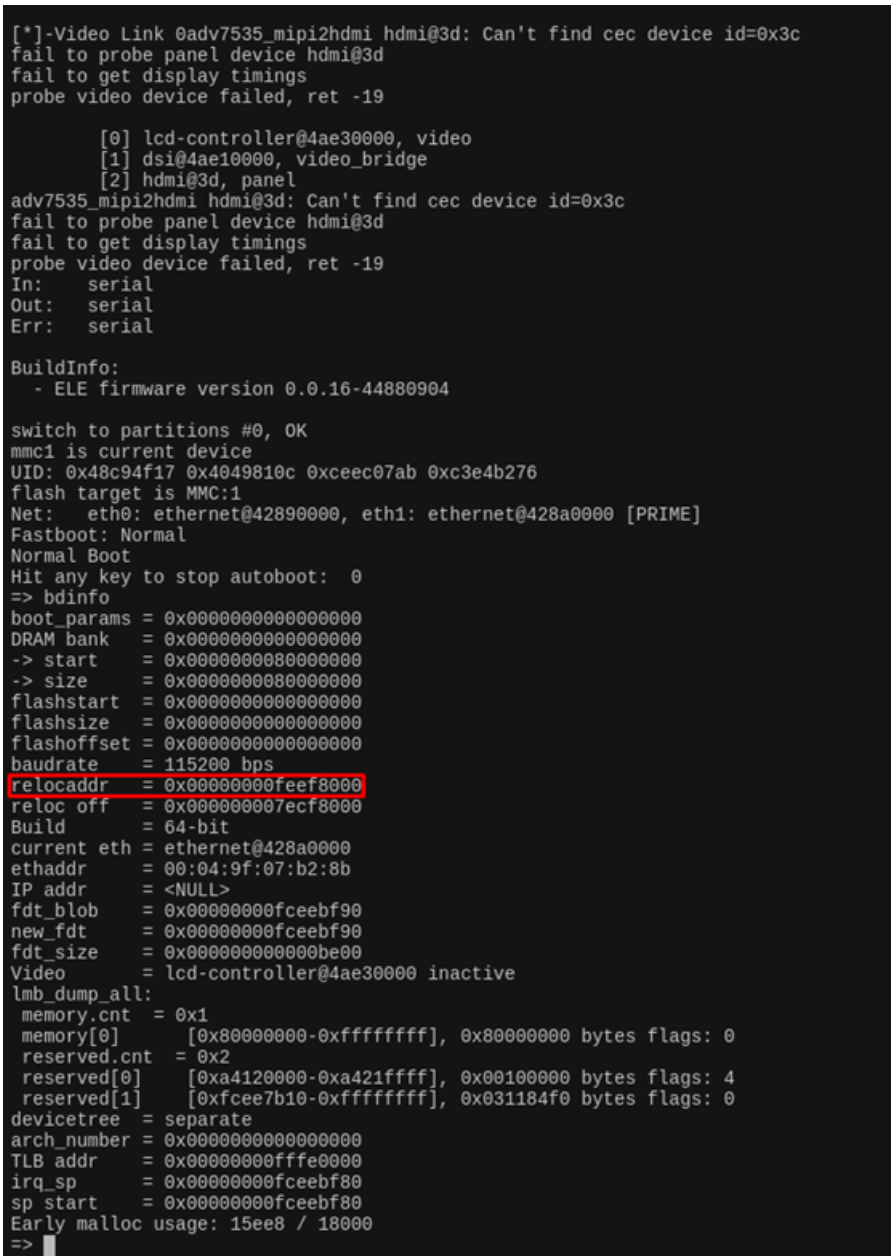

Figure 11. Type bdinfo to obtain the relocaddr value

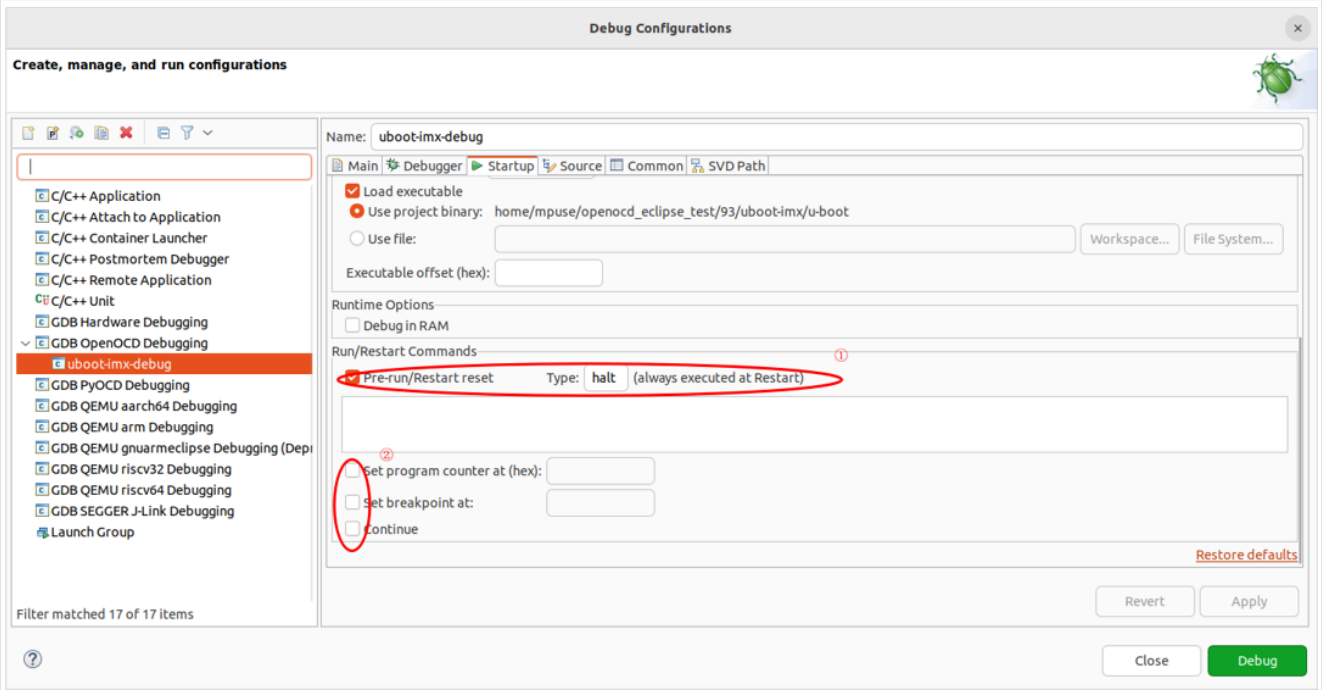

Figure 12. Configure the Startup tab - 2

Run/Restart Commands

- Make sure **Pre-run/Restart reset** has been selected, and the type is **halt**.
- **Set program counter at (hex), Set breakpoint at, Continue** do not need to be selected.
- Click **Apply** and **Close**.

4.2.3 Eclipse debugging steps

Note: *Disconnect other boards with the FTDI JTAG interface to avoid cross-impact.*

1. Open the serial console software, choose the third COM (Cortex-A);
2. Power on the board through the ON/OFF switch;
3. Stop at the U-Boot stage, check the time when you build U-Boot to make sure builds `u-boot.bin` and `u-boot-spl.bin` are used.

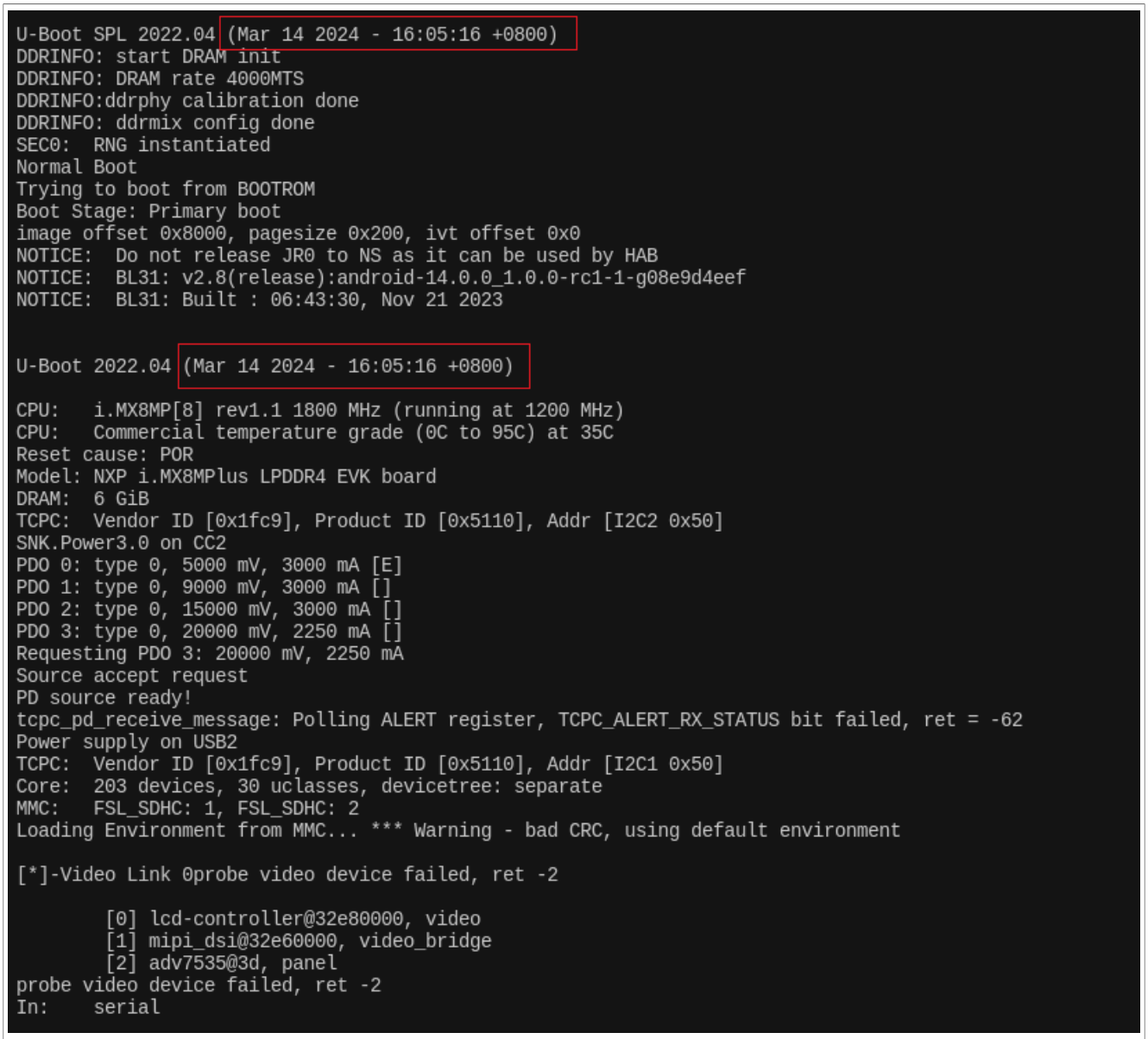

Figure 13. Check the SPL and U-Boot build time

4. Enable the JTAG remote debug interface with bcu:

```
sudo ./bcu_Ubuntu20 set_gpio remote_en 1 -board=imx8mpevk
sudo ./bcu_Ubuntu20 set_gpio remote_en 1 -board=imx93evk11b1
```

Note: *When the board must be restarted, first use BCU to disable the JTAG interface. If DEBUG is still needed, re-enable the JTAG interface.*

5. (Optional) Disable the JTAG remote debug interface with BCU:
Attention: *After running the following commands, JTAG remote debug will no longer be available.*

```
sudo ./bcu_Ubuntu20 set_gpio remote_en 0 -board=imx8mpevk
sudo ./bcu_Ubuntu20 set_gpio remote_en 0 -board=imx93evk11b1
```

6. Debugging pathway test

Before using OpenOCD GUI debugging, use OpenOCD on a terminal to check if the debugging pathway is available.

```
$ sudo src/openocd -s ./tcl -f interface/ftdi/imx93-evk.cfg -f board/
nxp_imx93-evk.cfg
Open On-Chip Debugger 0.11.0+dev-00651-g9de084e00 (2022-04-26-15:55)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : clock speed 1000 kHz
Info : JTAG tap: imx93.cpu tap/device found: 0x0892801d (mfg: 0x00e
 (Freescale (Motorola)), part: 0x8928, ver: 0x0)
Info : imx93.a55.0: hardware has 6 breakpoints, 4 watchpoints
Info : starting gdb server for imx93.a55.0 on 3333
Info : Listening on port 3333 for gdb connections
Info : starting gdb server for imx93.m33 on 3334
Info : Listening on port 3334 for gdb connections
Info : gdb port disabled
```

7. Open a new terminal to test the GDB connection

```
$ gdb-multiarch
(gdb) set architecture auto
(gdb) target remote localhost:3333
Remote debugging using localhost:3333
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x00000000feefa200 in ?? ()
(gdb) symbol-file u-boot
Reading symbols from u-boot...
(gdb) set $offset = ((gd_t *)$x18)->relocaddr
(gdb) symbol-file
Discard symbol table from `/home/mpuse/openocd_eclipse_test/93/uboot-imx/u-
boot'? (y or n) y
No symbol file now.
(gdb) add-symbol-file u-boot $offset
add symbol table from file "u-boot" at
 .text_addr = 0xfeef8000
(y or n) y
Reading symbols from u-boot...
(gdb) bt
#0  ?? () at arch/arm/cpu/armv8/exceptions.S:139
#1  0x0000000000000000 in ?? ()
```

8. (Optional) Open another terminal to test the Telnet connection,

```
$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> resume
> halt
imx93.a55.0 halted in AArch64 state due to debug-request, current mode: EL2H
cpsr: 0x200002c9 pc: 0xfef4419c
MMU: enabled, D-Cache: enabled, I-Cache: enabled
> step
imx93.a55.0 halted in AArch64 state due to single-step, current mode: EL2H
cpsr: 0x200002c9 pc: 0xfef441a0
```

```
MMU: enabled, D-Cache: enabled, I-Cache: enabled
> step
imx93.a55.0 halted in AArch64 state due to single-step, current mode: EL2H
cpsr: 0x200002c9 pc: 0xfef441a4
MMU: enabled, D-Cache: enabled, I-Cache: enabled
> step
imx93.a55.0 halted in AArch64 state due to single-step, current mode: EL2H
cpsr: 0x200002c9 pc: 0xfef441a8
MMU: enabled, D-Cache: enabled, I-Cache: enabled>
```

9. If the previous step works well, it confirms the pathway test pass. To run the Eclipse OpenOCD debugging:
   - Close the terminal windows in step5;
   - Reboot the EVK and let it stop at the U-Boot stage again;
   - Make sure the Eclipse OpenOCD configuration is finished.

   If so, click the **debug** button, wait a few seconds for Eclipse to start the server, connect the target and start the debugging session.

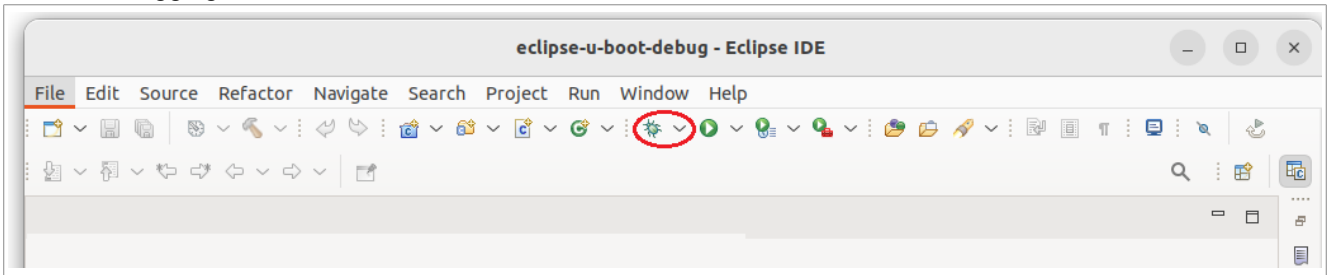

Figure 14. Debug button

- In case of a similar situation, press the **Resume** button as shown in Figure 15

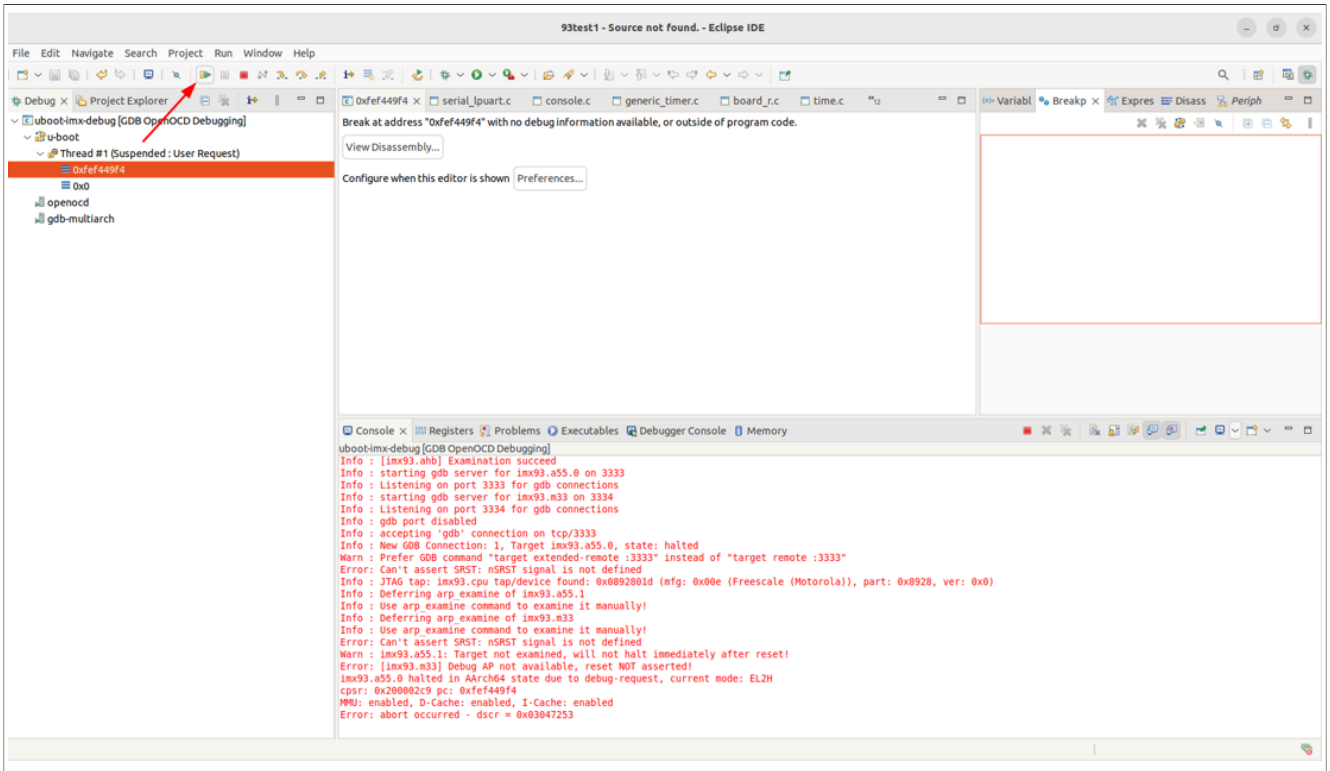

Figure 15. Resume

- Resume the debugging process
- Press **Suspend**

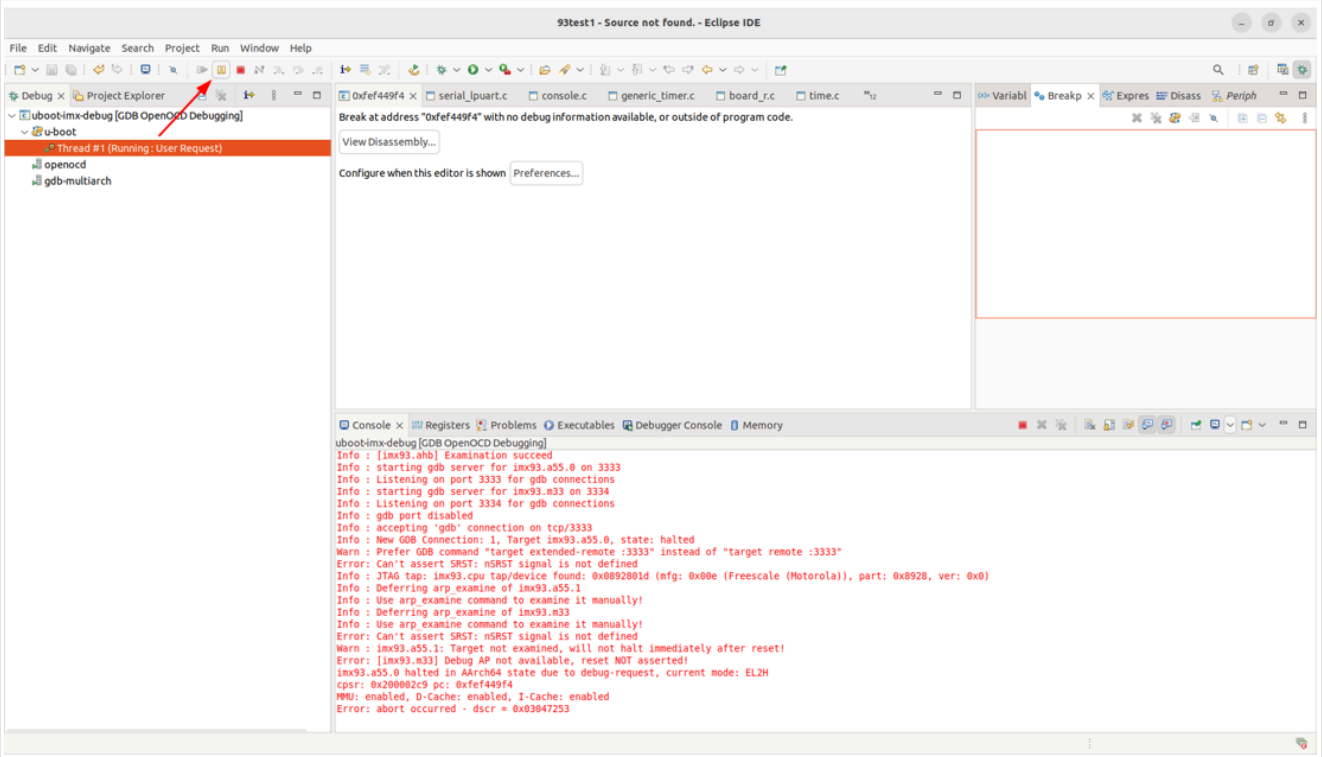

Figure 16. Suspend

- Halt the debugging process

10. If everything goes well, you can see:

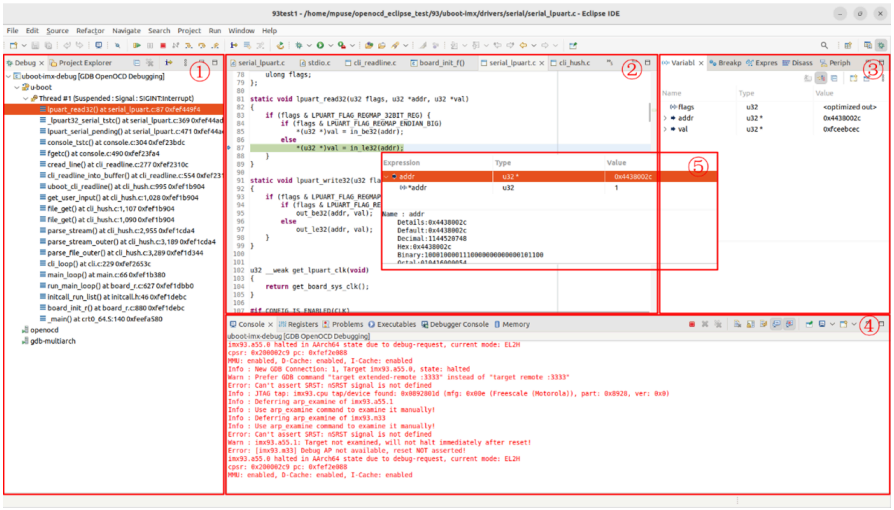

Figure 17. The main debug interface

On Figure 17, the sections are as follows:

a. The **Call stack** tab is on the left

b. The **Source code** tab is in the middle.

c. The **Variables** tab is on the right.

d. The **Console log** window is at the bottom.
To see different information, switch tabs. For example, you can see the breakpoint you set on the left.

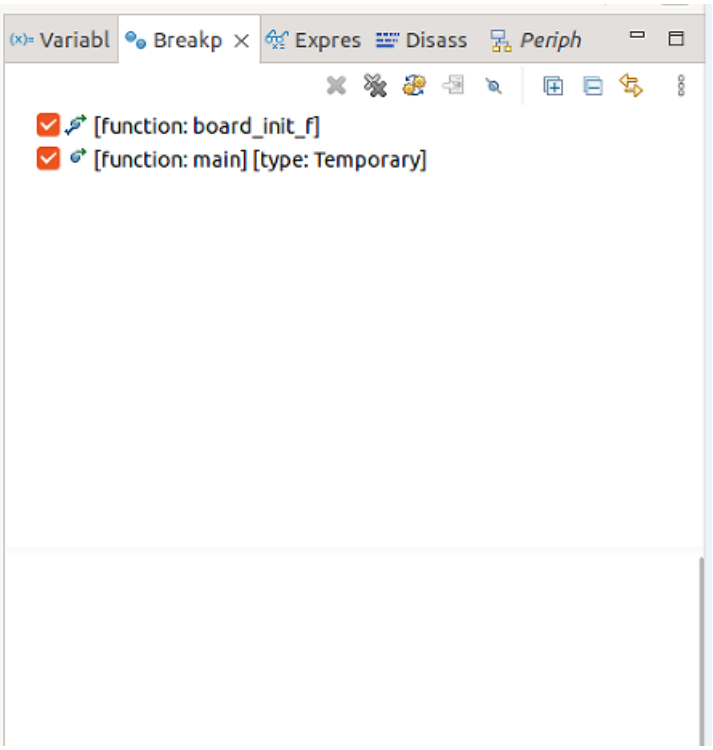

Figure 18. Breakpoint tab on the right side of main interface

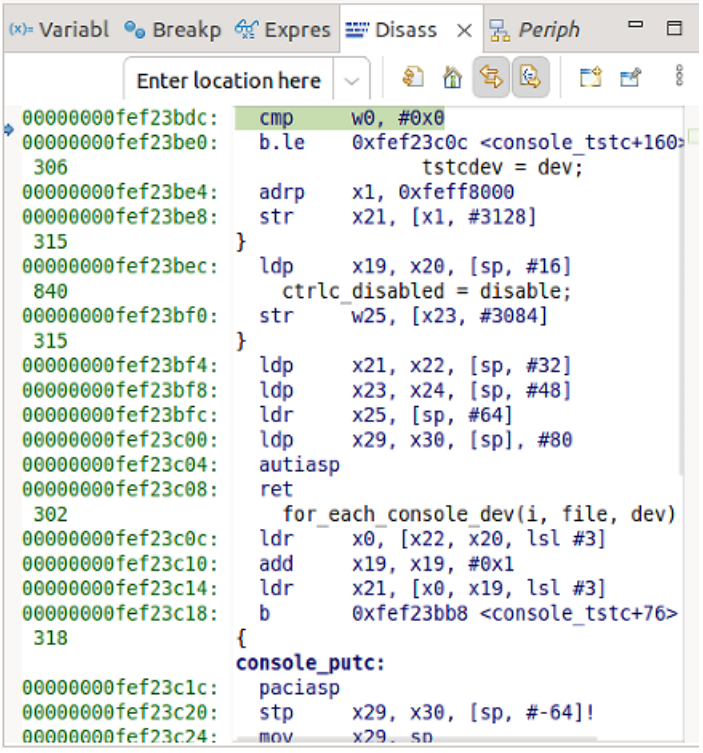

Figure 19. Disassembly tab on the right side of main interface

e. Variable information can be accessed by hovering the mouse over a variable.

Note: *For the errors in this region:*

Error 1: Cannot assert SRST: nSRST signal is not defined. We did not define nSRST in the cfg files, but the nSRST gpio does indeed exist, it is behind an I2C GPIO expander. Use bcu to reset the chip: `bcu reset sd/emmc/usb/... -board=imx93evk11`
Error 2: [imx93.m33] The target is not examined, it will not halt after reset! There is no m33 image in our build `flash.bin`*, this error will be reported.*

11. (Optional) You can use terminal debug mode, press **Window**, **Show View**, and choose **Debugger Console**

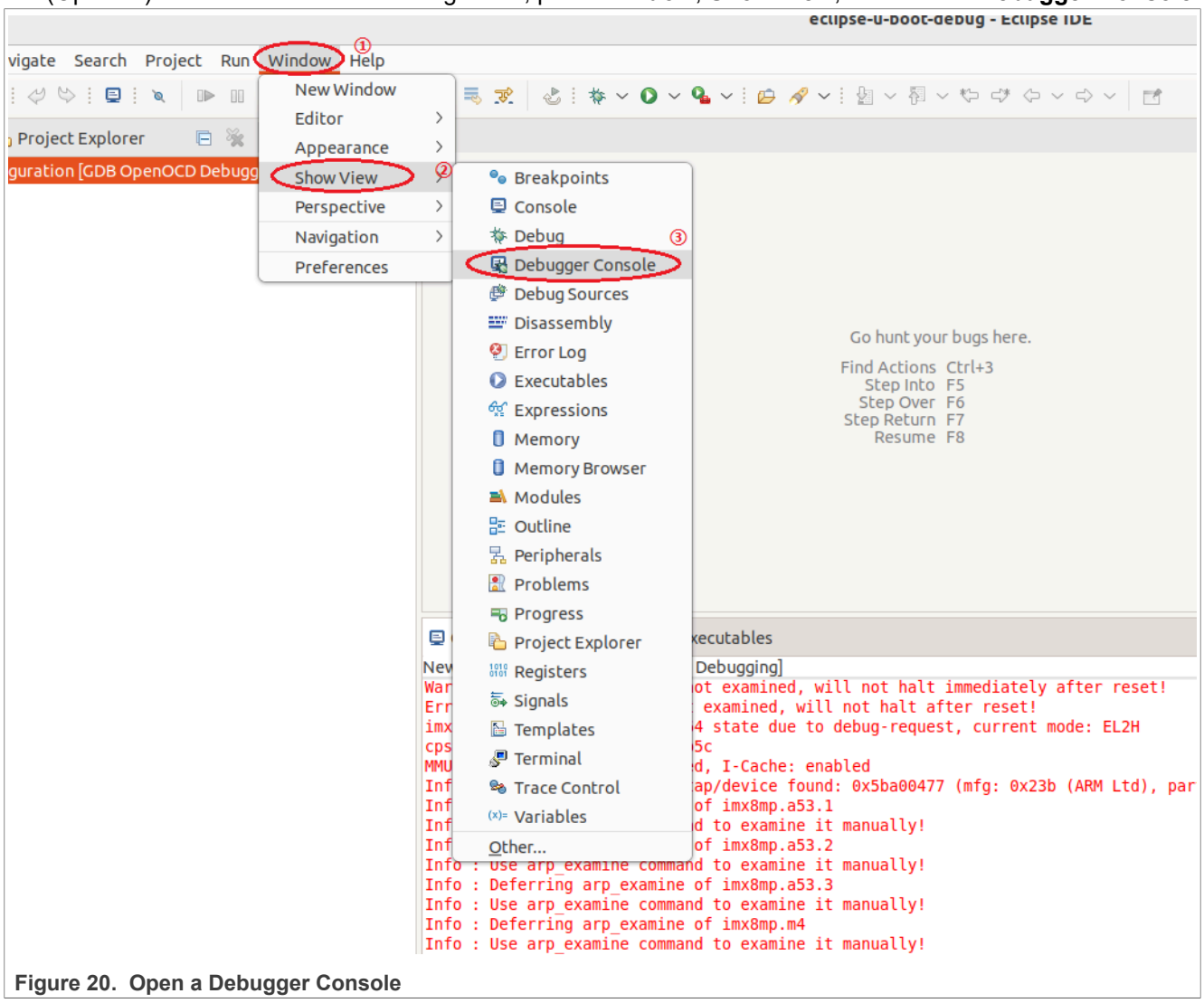

Figure 20. Open a Debugger Console

12. (Optional) The terminal debugging can be done in **Debugger Console**, like this：

```
b board_init_f
c
```

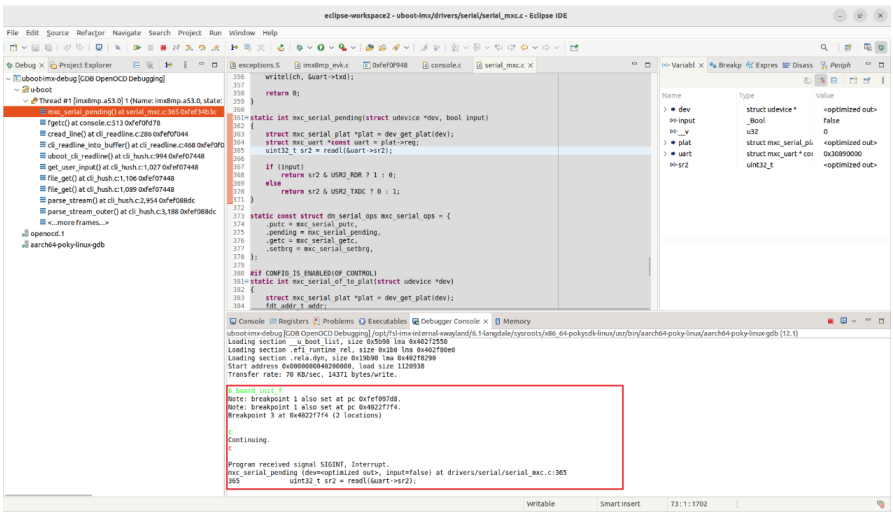

Figure 21. Use GDB command in Debugger Console

13. (Optional) Click the button to switch the Console.
    This gdb traces window contains lots of gdb configuration information, for example, the log in the red box corresponding to the setting in the **Startup** tab, use it to debug your customized configuration.

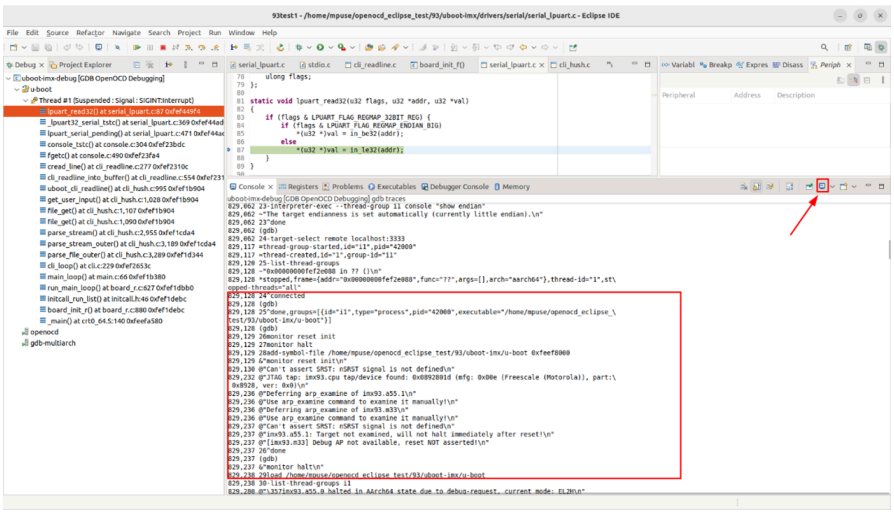

Figure 22. GDB traces

14. Click the **Suspend** button above the IDE to pause and start step-by-step debugging.
    From the left to right, these buttons are **Resume**, **Suspend**, **Terminate**, **Disconnect**, **StepInto**, **StepOver**, **Step Return**.

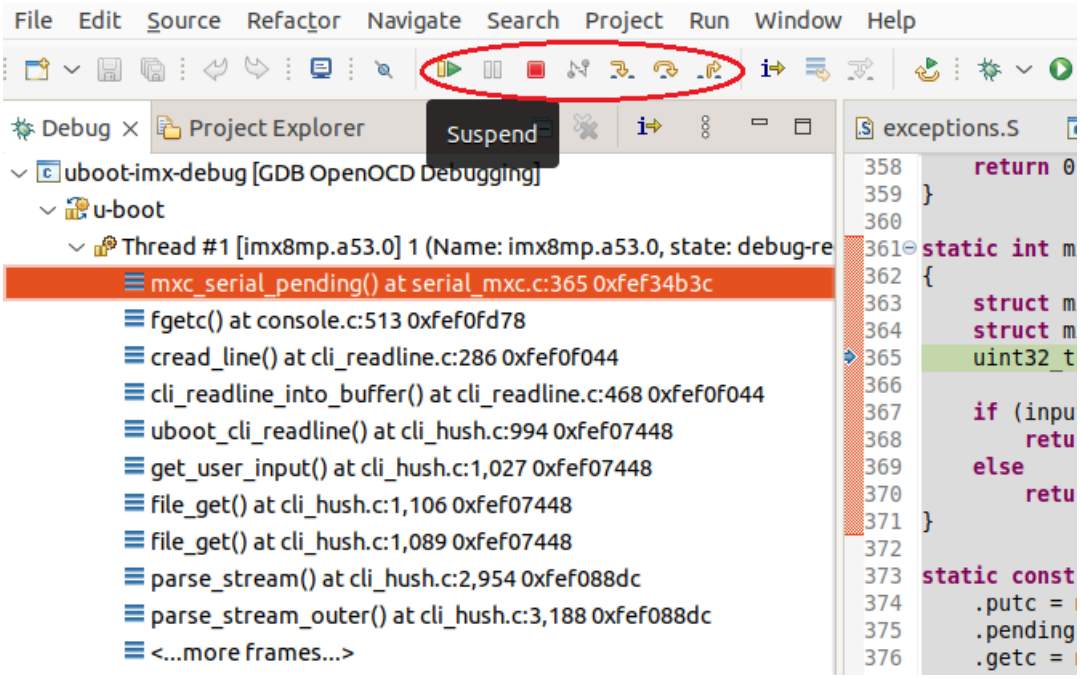

Figure 23. Debug buttons

15. Always make sure that your U-Boot is alive, if it reports an error or the terminal does not response to any input, disable the remote debug function by using bcu, reset the board and re-enable the remote debug function again.

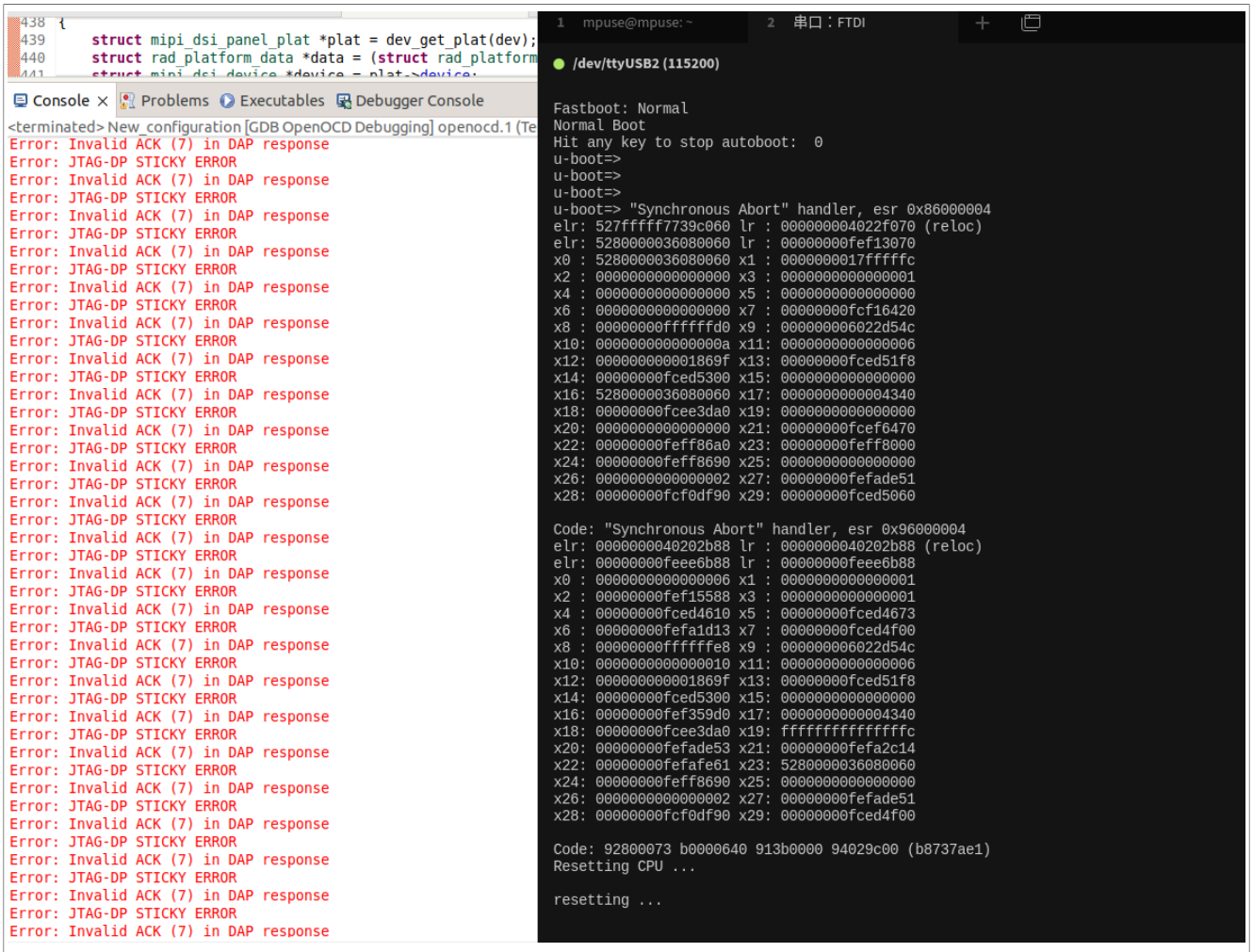

Figure 24. Error reported in Eclipse console and serial console

4.2.4 Debugging U-Boot with on GDB terminal

There are a lot of GDB commands. Use **help** to check these commands. Then these commands classes are listed:

```
Type "apropos -v word" for full documentation of commands related to "word".
Command name abbreviations are allowed if unambiguous.
(gdb) help
List of classes of commands:

aliases -- User-defined aliases of other commands.
breakpoints -- Making program stop at certain points.
data -- Examining data.
files -- Specifying and examining files.
internals -- Maintenance commands.
obscure -- Obscure features.
running -- Running the program.
stack -- Examining the stack.
status -- Status inquiries.
support -- Support facilities.
text-user-interface -- TUI is the GDB text based interface.
tracepoints -- Tracing of program execution without stopping the program.
user-defined -- User-defined commands.
```

Figure 25. GDB help command

Use **help stack** to check the GDB commands in the stack class:

```
(gdb) help stack
Examining the stack.
The stack is made up of stack frames.  Gdb assigns numbers to stack frames
counting from zero for the innermost (currently executing) frame.

At any time gdb identifies one frame as the "selected" frame.
Variable lookups are done with respect to the selected frame.
When the program being debugged stops, gdb selects the innermost frame.
The commands below can be used to select other frames by number or address.

List of commands:

backtrace, where, bt -- Print backtrace of all stack frames, or innermost COUNT frames.
down, dow, do -- Select and print stack frame called by this one.
faas -- Apply a command to all frames (ignoring errors and empty output).
frame, f -- Select and print a stack frame.
frame address -- Select and print a stack frame by stack address.
frame apply -- Apply a command to a number of frames.
frame apply all -- Apply a command to all frames.
frame apply level -- Apply a command to a list of frames.
frame function -- Select and print a stack frame by function name.
frame level -- Select and print a stack frame by level.
frame view -- View a stack frame that might be outside the current backtrace.
return -- Make selected stack frame return to its caller.
select-frame -- Select a stack frame without printing anything.
select-frame address -- Select a stack frame by stack address.
select-frame function -- Select a stack frame by function name.
select-frame level -- Select a stack frame by level.
select-frame view -- Select a stack frame that might be outside the current backtrace.
up -- Select and print stack frame that called this one.
```

Figure 26. GDB help stack command

4.2.4.1 Common commands

Common commands are the following:

4.2.4.2 Stack commands

backtrace(bt) Print backtrace of all stack frames or innermost COUNT frames.

frame n Select and print a stack frame.

up Select and print the stack frame that called this one.

down Select and print the stack frame called by this one.

info stack Backtrace of the stack, or innermost COUNT frames.

list(l) List a specified function or line.

```
(gdb) bt
#0  0x00000000fef449f4 in lpuart_read32 (flags=<optimized out>, addr=0x4438002c, val=0xfceebcec) at drivers/serial/serial_lpuart.c:87
#1  0x00000000fef44ad0 in _lpuart32_serial_tstc (plat=0xfcef8ce0) at drivers/serial/serial_lpuart.c:369
#2  lpuart_serial_pending (dev=0xfcef8c40, input=true) at drivers/serial/serial_lpuart.c:471
#3  0x00000000fef23bdc in console_tstc (file=file@entry=0) at common/console.c:304
#4  0x00000000fef23fa4 in fgetc (file=0) at common/console.c:490
#5  0x00000000fef2310c in cread_line (timeout=-16806880, len=<synthetic pointer>, buf=0xfeff8404 "", prompt=0xfcf181e0 "\003") at common/cli_readline.c:277
#6  cli_readline_into_buffer (prompt=0xfcf181e0 "\003", buffer=0xfeff8404 "", timeout=-16806880) at common/cli_readline.c:554
#7  0x00000000fef1b904 in uboot_cli_readline (i=0xfceebef0) at common/cli_hush.c:995
#8  get_user_input (i=0xfceebef0) at common/cli_hush.c:1028
#9  file_get (i=<optimized out>) at common/cli_hush.c:1107
#10 file_get (i=0xfceebef0) at common/cli_hush.c:1090
#11 0x00000000fef1cda4 in parse_stream (end_trigger=<optimized out>, input=<optimized out>, ctx=<optimized out>, dest=<optimized out>) at common/cli_hush.c:2955
#12 parse_stream_outer (inp=inp@entry=0xfceebef0, flag=flag@entry=2) at common/cli_hush.c:3189
#13 0x00000000fef1d344 in parse_file_outer () at common/cli_hush.c:3289
#14 0x00000000fef2653c in cli_loop () at common/cli.c:229
#15 0x00000000fef1b380 in main_loop () at common/main.c:66
#16 0x00000000fef1dbb0 in run_main_loop () at common/board_r.c:627
#17 0x00000000fef1debc in initcall_run_list (init_sequence=0xfefc1118) at include/initcall.h:46
#18 board_init_r (new_gd=<optimized out>, dest_addr=<optimized out>) at common/board_r.c:880
#19 0x00000000feefa580 in _main () at arch/arm/lib/crt0_64.S:140
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb) frame 19
#19 0x00000000feefa580 in _main () at arch/arm/lib/crt0_64.S:140
140             bl      c_runtime_cpu_setup             /* still call old routine */
(gdb) l
135     relocation_return:
136
137     /*
138      * Set up final (full) environment
139      */
140             bl      c_runtime_cpu_setup             /* still call old routine */
141     #endif /* !CONFIG_SPL_BUILD */
142     #if !defined(CONFIG_SPL_BUILD) || CONFIG_IS_ENABLED(FRAMEWORK)
143     #if defined(CONFIG_SPL_BUILD)
144             bl      spl_relocate_stack_gd           /* may return NULL */
```

Figure 27. GDB stack example in U-Boot project -1

```
(gdb) down
#18 board_init_r (new_gd=<optimized out>, dest_addr=<optimized out>) at common/board_r.c:880
880             if (initcall_run_list(init_sequence_r))
(gdb) down
#17 0x00000000fef1debc in initcall_run_list (init_sequence=0xfefc1118) at include/initcall.h:46
46                      ret = (*init_fnc_ptr)();
(gdb) down
#16 0x00000000fef1dbb0 in run_main_loop () at common/board_r.c:627
627                     main_loop();
(gdb) down
#15 0x00000000fef1b380 in main_loop () at common/main.c:66
66              cli_loop();
(gdb) l
61              if (cli_process_fdt(&s))
62                      cli_secure_boot_cmd(s);
63
64              autoboot_command(s);
65
66              cli_loop();
67              panic("No CLI available");
68      }
(gdb) up
#16 0x00000000fef1dbb0 in run_main_loop () at common/board_r.c:627
627                     main_loop();
(gdb) up
#17 0x00000000fef1debc in initcall_run_list (init_sequence=0xfefc1118) at include/initcall.h:46
46                      ret = (*init_fnc_ptr)();
(gdb) l
41                                              (char *)*init_fnc_ptr - reloc_ofs,
42                                              (char *)*init_fnc_ptr);
43                      else
44                              debug("initcall: %p\n", (char *)*init_fnc_ptr - reloc_ofs);
45
```

Figure 28. GDB stack example in U-Boot project -2

1. Use **backtrace(bt)** to check the stack as shown in Figure 27
2. Use **frame** 19 to select the #19 frame
3. Use **list** to check the code.
4. Use **down/up** to switch the frame
5. Use **list(l)** to check the code anytime.
   Combining these commands can easily browse stack information and related code.

4.2.4.3 breakpoint

breakpoint(b) Set the breakpoint at a specified location.

delete(d) Delete some breakpoints or autodisplay expressions.

clear Clear breakpoint at specified location.

info b Status of specified breakpoints.

disable/enable Disable/Enable some breakpoints.

```
(gdb) bt
#0  console_tstc (file=file@entry=0) at common/console.c:302
#1  0x00000000fef23fa4 in fgetc (file=0) at common/console.c:490
#2  0x00000000fef2310c in cread_line (timeout=-16806880, len=<synthetic pointer>, buf=0xfeff8404 "sssh\n", prompt=0xfcf181e0 "\003") at common/cli_readline.c:277
#3  cli_readline_into_buffer (prompt=0xfcf181e0 "\003", buffer=0xfeff8404 "sssh\n", timeout=-16806880) at common/cli_readline.c:554
#4  0x00000000fef1b904 in uboot_cli_readline (i=0xfceebef0) at common/cli_hush.c:995
#5  get_user_input (i=0xfceebef0) at common/cli_hush.c:1028
#6  file_get (i=<optimized out>) at common/cli_hush.c:1107
#7  file_get (i=0xfceebef0) at common/cli_hush.c:1090
#8  0x00000000fef1cda4 in parse_stream (end_trigger=<optimized out>, input=<optimized out>, ctx=<optimized out>, dest=<optimized out>) at common/cli_hush.c:2955
#9  parse_stream_outer (inp=inp@entry=0xfceebef0, flag=flag@entry=2) at common/cli_hush.c:3189
#10 0x00000000fef1d344 in parse_file_outer () at common/cli_hush.c:3289
#11 0x00000000fef2653c in cli_loop () at common/cli.c:229
#12 0x00000000fef1b380 in main_loop () at common/main.c:66
#13 0x00000000fef1dbb0 in run_main_loop () at common/board_r.c:627
#14 0x00000000fef1debc in initcall_run_list (init_sequence=0xfefc1118) at include/initcall.h:46
#15 board_init_r (new_gd=<optimized out>, dest_addr=<optimized out>) at common/board_r.c:880
#16 0x00000000feefa580 in _main () at arch/arm/lib/crt0_64.S:140
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb) b fgetc
Breakpoint 1 at 0xfef23f7c: fgetc. (2 locations)
(gdb) b get_user_input
Breakpoint 2 at 0xfef1b8d0: file common/cli_hush.c, line 1044.
(gdb) b console.c:300
Breakpoint 3 at 0xfef23b6c: file common/console.c, line 301.
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   <MULTIPLE>
1.1                         y   0x00000000fef23f7c in fgetc at common/console.c:475
1.2                         y   0x00000000fef23fdc in fgetc at common/console.c:477
2       breakpoint     keep y   0x00000000fef1b8d0 in get_user_input at common/cli_hush.c:1044
3       breakpoint     keep y   0x00000000fef23b6c in console_tstc at common/console.c:301
(gdb) c
Continuing.

Breakpoint 3, console_tstc (file=file@entry=0) at common/console.c:838
838             int prev = ctrlc_disabled;      /* save previous state */
```

Figure 29. GDB breakpoint example in U-Boot project -1

```
(gdb) d 1
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00000000fef1b8d0 in get_user_input at common/cli_hush.c:1044
        breakpoint already hit 1 time
3       breakpoint     keep y   0x00000000fef23b6c in console_tstc at common/console.c:301
        breakpoint already hit 5 times
(gdb) n
301             prev = disable_ctrlc(1);
(gdb) n
840             ctrlc_disabled = disable;
(gdb) n
301             prev = disable_ctrlc(1);
(gdb) n
302             for_each_console_dev(i, file, dev) {
(gdb) n
303                     if (dev->tstc != NULL) {
(gdb) n
304                             ret = dev->tstc(dev);
(gdb) c
Continuing.

Breakpoint 3, console_tstc (file=file@entry=0) at common/console.c:838
838             int prev = ctrlc_disabled;      /* save previous state */
(gdb) n
301             prev = disable_ctrlc(1);
(gdb) n
840             ctrlc_disabled = disable;
(gdb) n
301             prev = disable_ctrlc(1);
(gdb) n
302             for_each_console_dev(i, file, dev) {
```

Figure 30. GDB breakpoint example in U-Boot project -2

1. Use **backtrace(bt)** to check the stack as shown in Figure 29
2. Use **breakpoint(b)** to set the breakpoint
3. Use **b getc** and **b get_user_input** to set the breakpoint on the function
4. Use **b console.c:300** to set the breakpoint on line 300 of the `console.c` file

5. Use **info b** to check the current breakpoint setting
6. Use **delete(d)** to delete the breakpoint
7. Use **c** to continue the process, then the program stops at the next breakpoint.

4.2.4.4 Control and view commands

next(n) Step program, proceeding through subroutine calls.

continue(c) Execute to the next breakpoint or program end.

run(r) Start the debugged program.

whatis Print the data type of the expression EXP.

print(p) Print the value of the expression EXP.

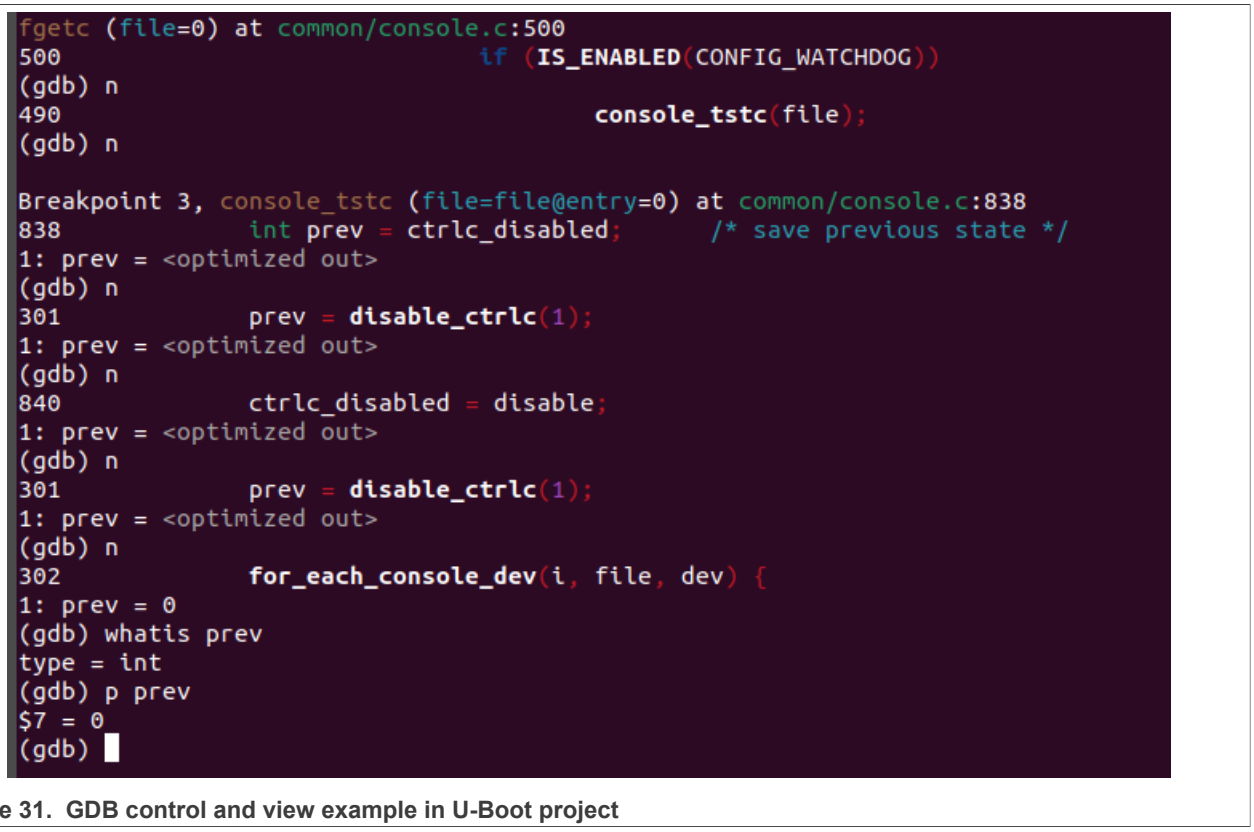

Figure 31. GDB control and view example in U-Boot project

1. Use **whatis** *prev* to print the data type of *prev* as shown in Figure 31
2. Use **print(p)** prev to print the data value of *prev.*

4.2.4.5 GDB layout split commands

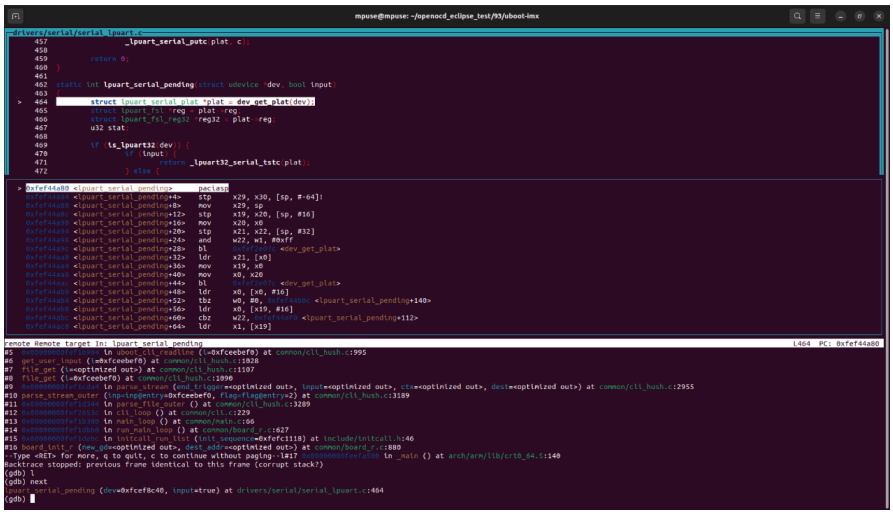

Figure 32. GDB split layout example in U-Boot project

Use the **layout split, bt** combination to see the source code, assembly code, and gdb console in the same window.

5 Native RTOS OpenOCD debugging

This chapter introduces the method of using J-link for native RTOS debugging.

5.1 Native RTOS on Cortex-A Core introduction

Native RTOS on Cortex-A Core refers to the RTOS running on Cortex-A Core without a hypervisor and is kicked to a specified Cortex-A Core by U-Boot commands.

Native RTOS is supported in Real-time Edge Software. The key technology components of Real-time Edge Software include Real-time System, Heterogeneous Multicore Framework, Heterogeneous Multi-SoC Framework, Real-time Networking, and Protocols.

Native RTOS currently supports FreeRTOS and Zephyr on Cortex-A Core, it can use both Cortex-A Core's high performance and RTOS's low latency schedule and interrupt ability.

There are some example applications in Heterogeneous Multicore Framework of Real-time Edge Software. Refer to *Real-time Edge Software User Guide* (document REALTIMEEDGEUG) for more details.

5.2 EVK board software setup

Download a Real-time Edge pre-build SD card binary image from https://www.nxp.com/rtedge, the image for i.MX 93 EVK is nxp-image-real-time-edge-imx93evk.wic after decompression and flash this image to SD by using the following method:

Method 1: download it to the board by using uuu (change the bootmode to serial download mode and use the USB-Type C cable):

```
$ sudo ./uuu -b sd nxp-image-real-time-edge-imx93evk.wic # for 93
```

Method 2: flash it to the SD card directly:

```
$ sudo dd if=nxp-image-real-time-edge-imx93evk.wic of=/dev/sd[x] bs=1M
 status=progress conv=fsync
```

Note: *Check your card reader partition and replace sd[x] with your corresponding partition.*

5.3 Debugging Native RTOS with Eclipse IDE

This example demonstrates how to debug Native RTOS by using OpenOCD together with Eclipse IDE on the i.MX 93 EVK board. In this example, J-Link is used. There is a J-Link link issue on several versions of i.MX 93 EVK boards. A workaround is to remove the Wi-Fi/Bluetooth module from the M.2 connector.

5.3.1 Building Native RTOS with Eclipse IDE

1. Download the source code of Native RTOS in a heterogeneous-multicore project.

```
# mkdir ~/a_core_jlink
# cd ~/a_core_jlink/
# pip install west
# west init -m https://github.com/nxp-real-time-edge-sw/heterogeneous-
multicore.git workspace
# cd workspace & west update
```

2. Download and set up the toolchain:

```
# mkdir ~/toolchains/
# cd ~/toolchains/
# wget https://developer.arm.com/-/media/Files/downloads/gnu-a/10.3-
2021.07/binrel/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf.tar.xz
# tar xf gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf.tar.xz
```

Note: *For detailed information on the environment setting and image building, refer to Section 3.2 "Building Heterogeneous Multicore RTOS Application" of Real-time Edge Software User Guide (document REALTIMEEDGEUG).*

3. Open Eclipse and select the directory of the workspace.

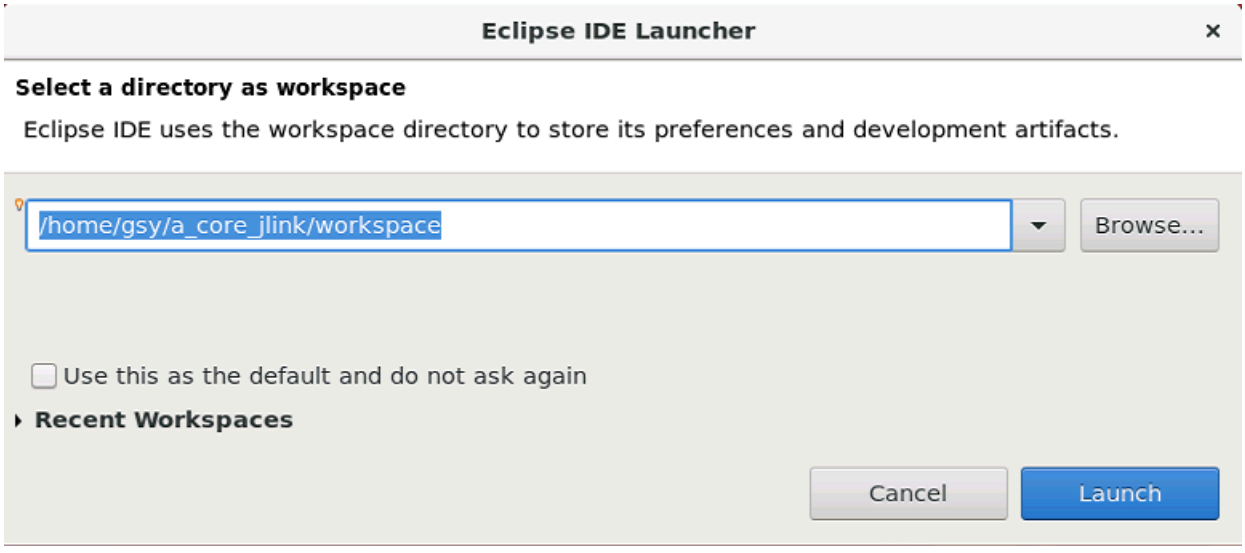

Figure 33. Launch the workspace

4. Import the existing project.

To import the project, click **Project Explorer** => **Import projects** and select **Existing Code as Makefile Project** as shown in Figure 34.

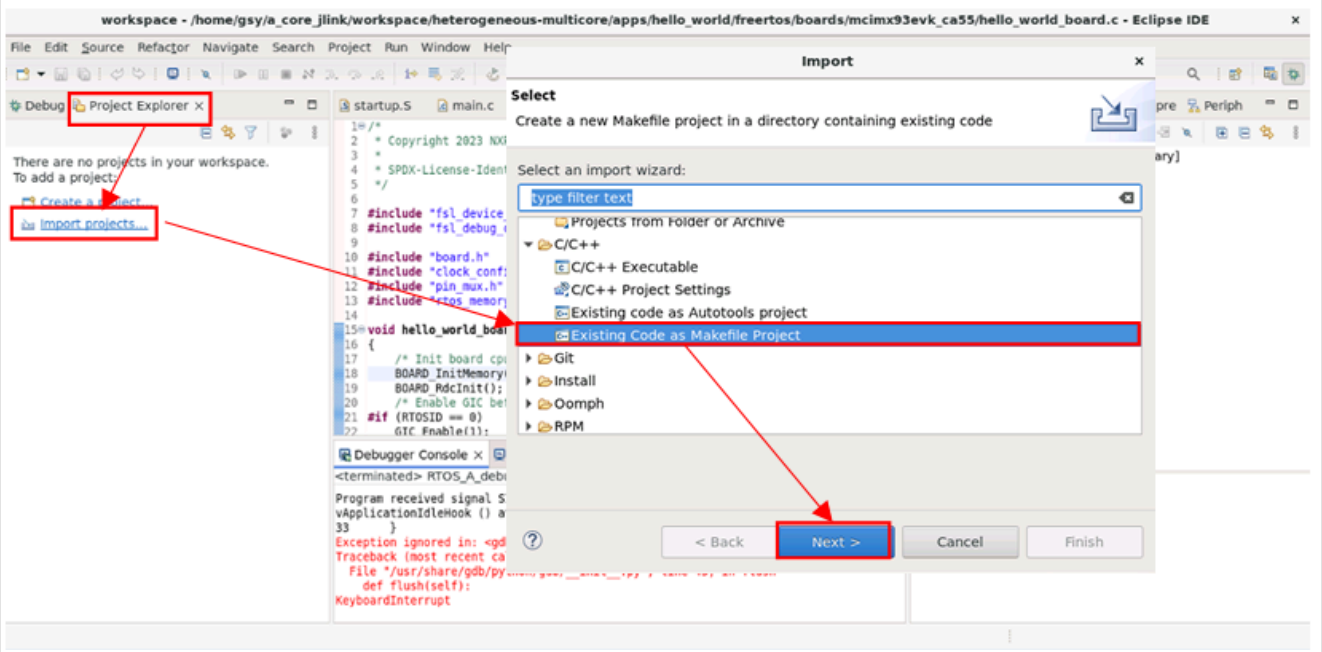

Figure 34. Import existing project

5. Import the existing code.
   To import the existing code, click **Browse** to select the project to build and click **Finish** as shown in Figure 35.
   In this case, the project is apps/hello_world/freertos/boards/mcimx93evk_ca55

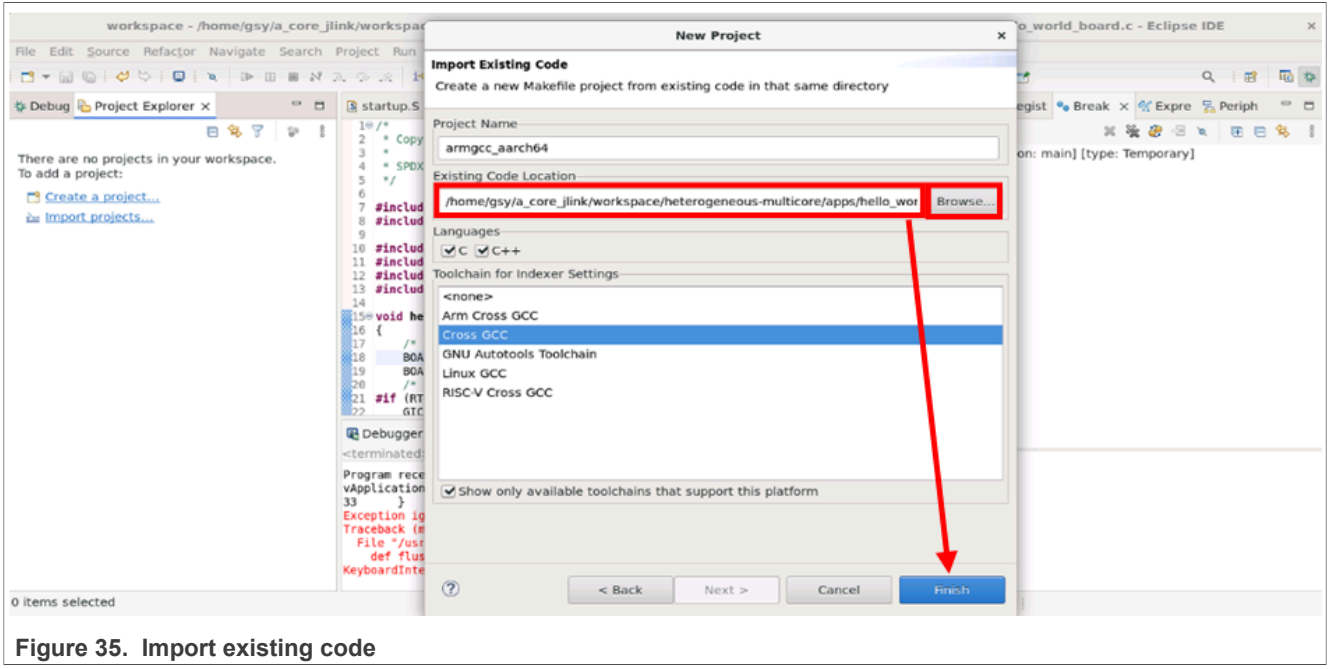

Figure 35. Import existing code

6. Edit the environment setting.
   To edit the environment setting, right-click the imported project and Select **Properties** as shown in Figure 36.

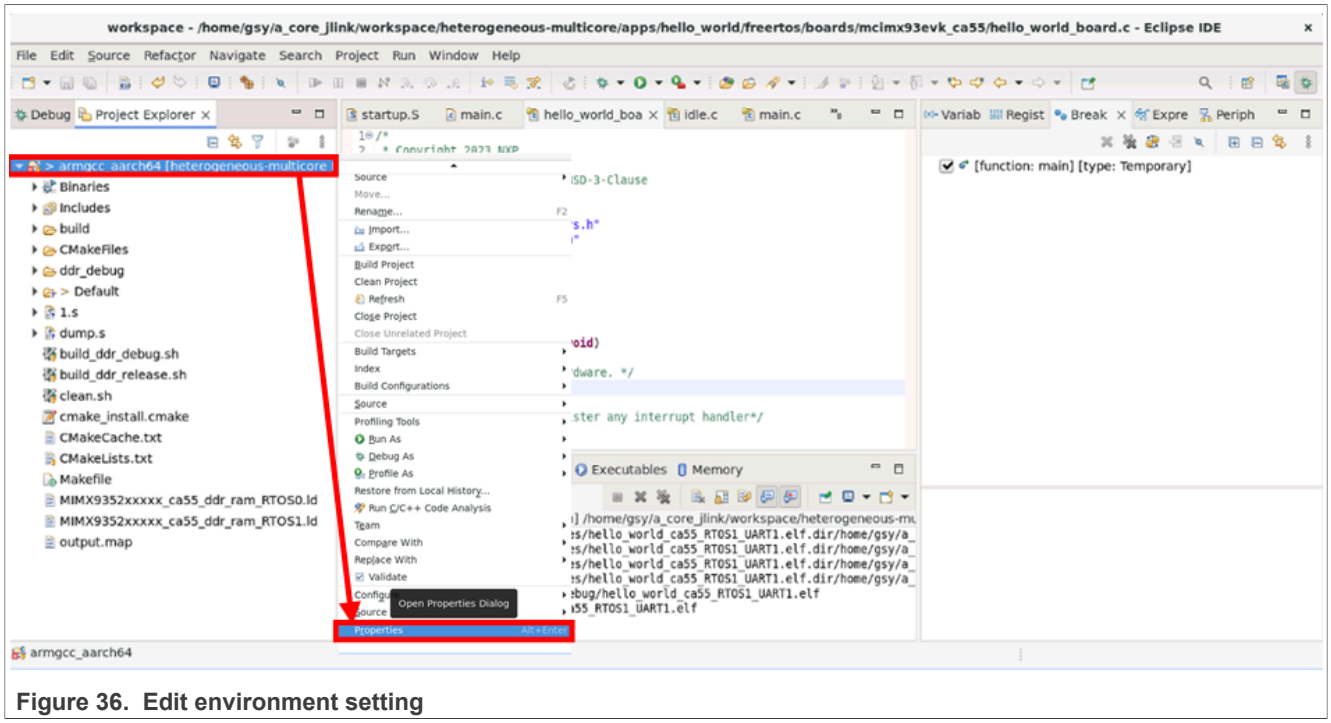

Figure 36. Edit environment setting

7. Add a building environment.
   To do this, follow the steps shown in Figure 37:
   a. Click **Builders** => **New** to add a building configuration.
   b. Click **Main** => Click **Browse** to select the building script.
      Note: `build_ddr_debug.sh` *is used as an example.*

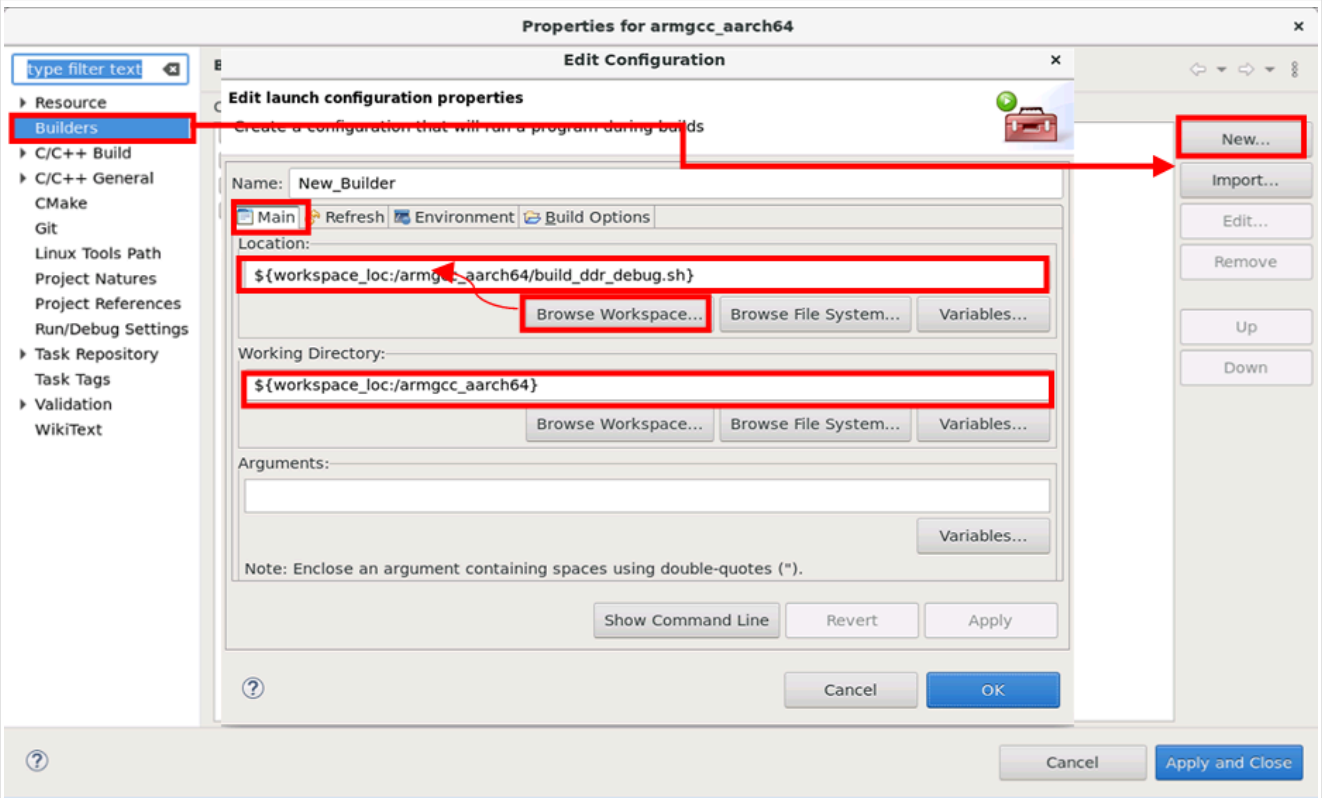

Figure 37. Add building environment

c. Click **Environment** and **Add** to include the GCC toolchain that is to be used as shown in Figure 38.

d. Enter the variable using for the toolchain:
Name: ARMGCC_DIR
Value: `/home/gsy/toolchains/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf`
Note: *Edit the absolute path with one actual using.*

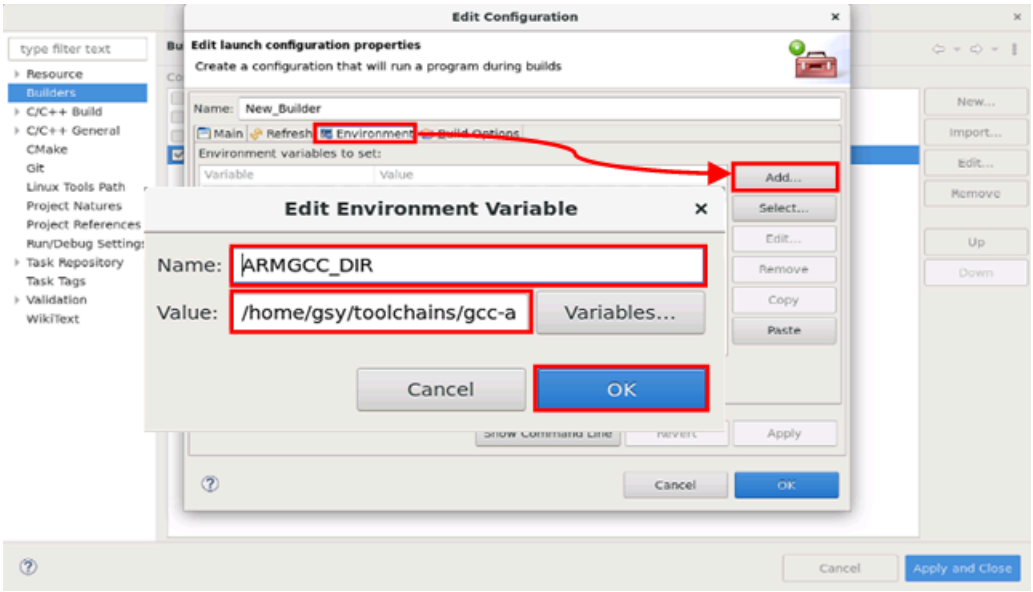

Figure 38. Set the toolchain environment value

e. Click the **Ok** button to save the configuration of the toolchain.

f. Select the building configuration that is added and click **Apply and Close** as shown in Figure 39.

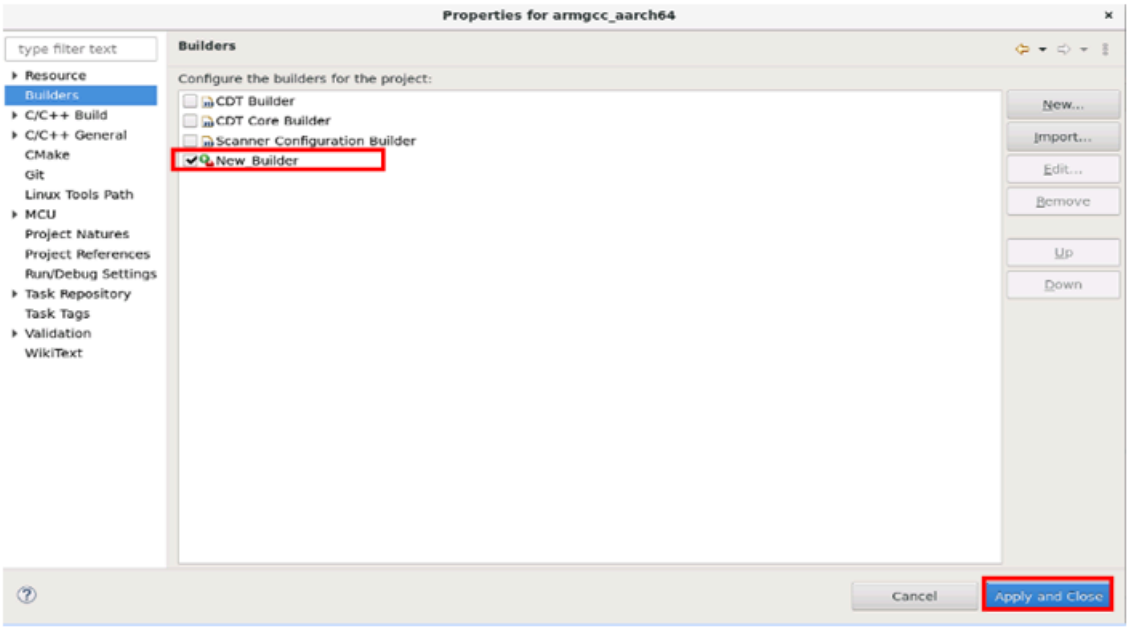

Figure 39. Builder tab

8. Build the Native RTOS image.
   To do this, right-click the imported project and select **Build Project** to start building as shown in Figure 40.

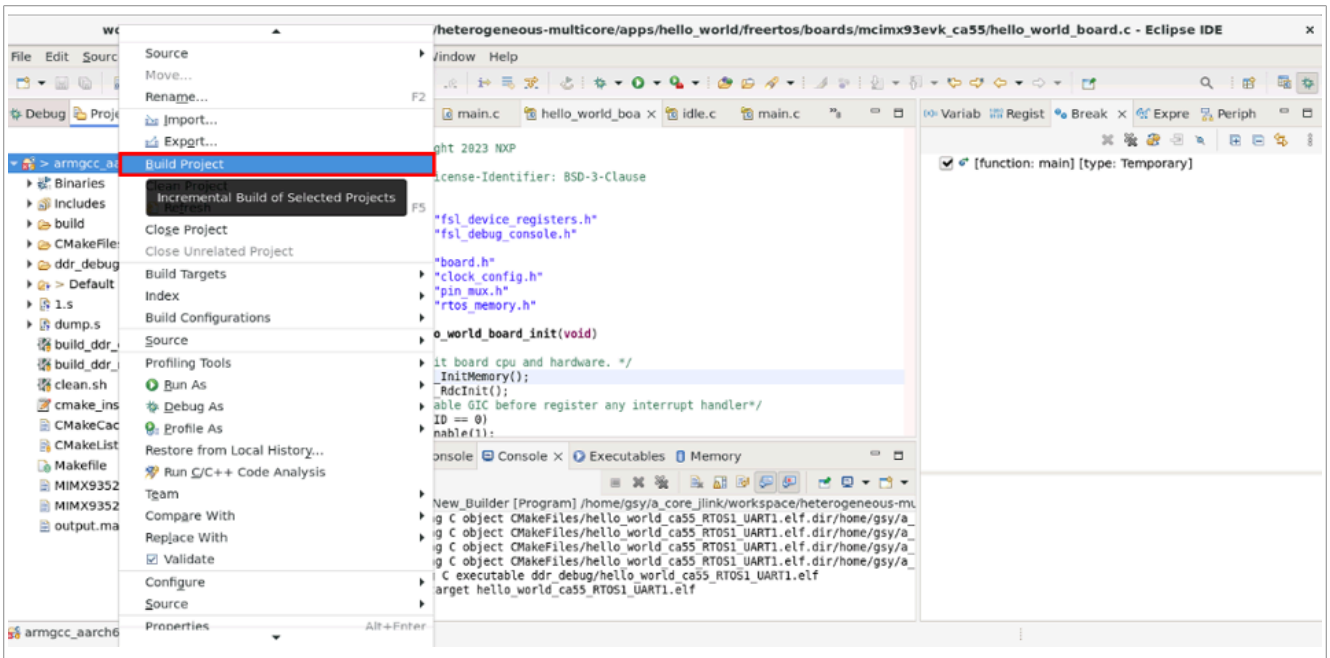

Figure 40. Build project

5.3.2 Eclipse OpenOCD configuration for Native RTOS debugging

1. Open Eclipse and select a directory as a workspace.

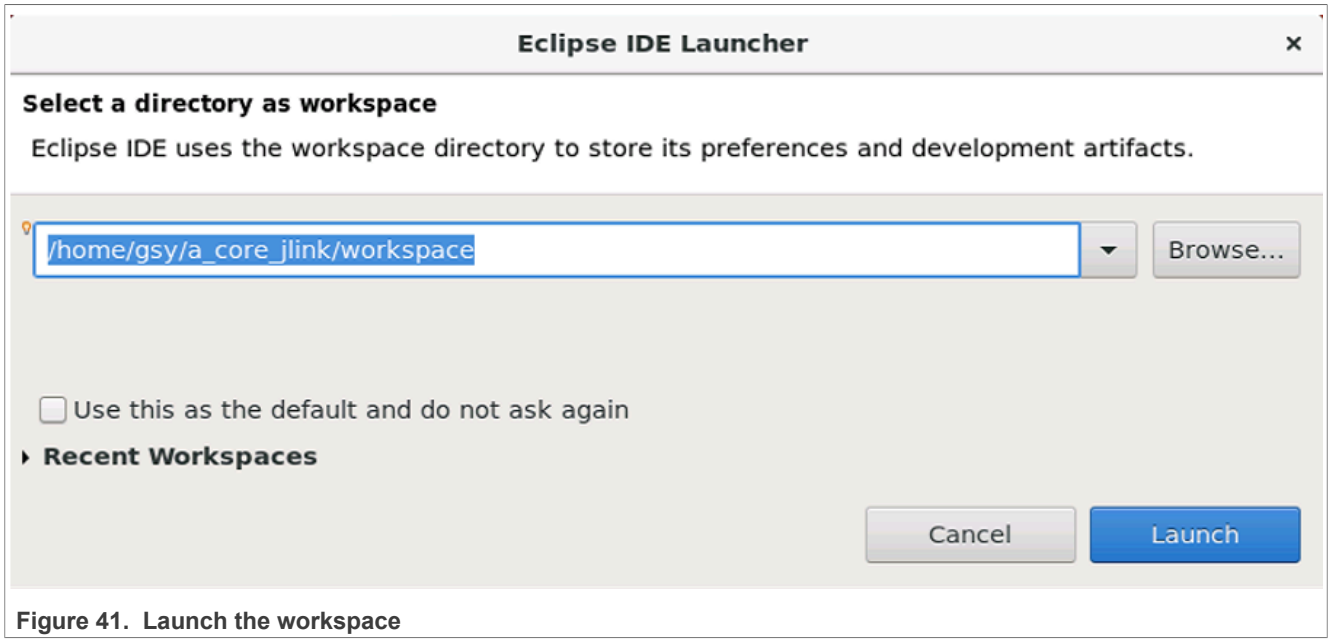

Figure 41. Launch the workspace

2. Add a Debug Configuration.
   To do this, Click **Run** and then click **Debug Configurations** as shown in Figure 42

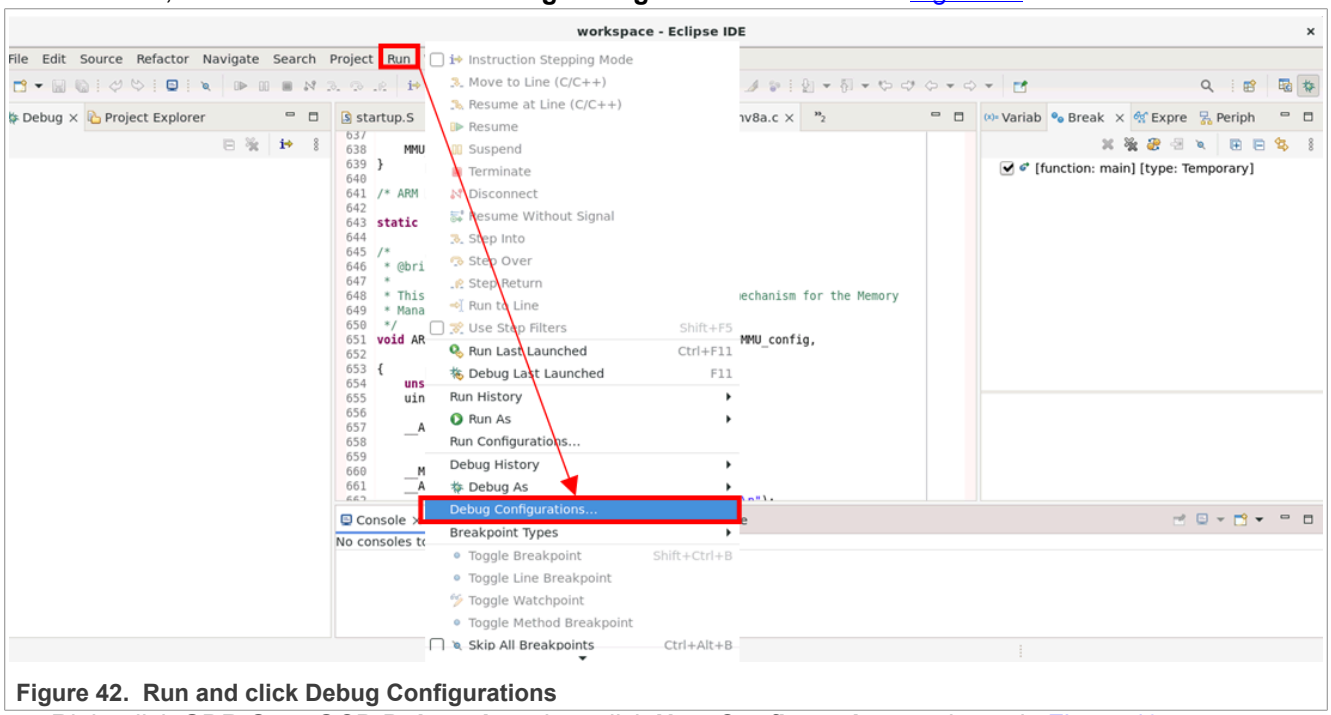

Figure 42. Run and click Debug Configurations

Right-click **GDB OpenOCD Debugging**, then click **New Configuration** as shown in Figure 43

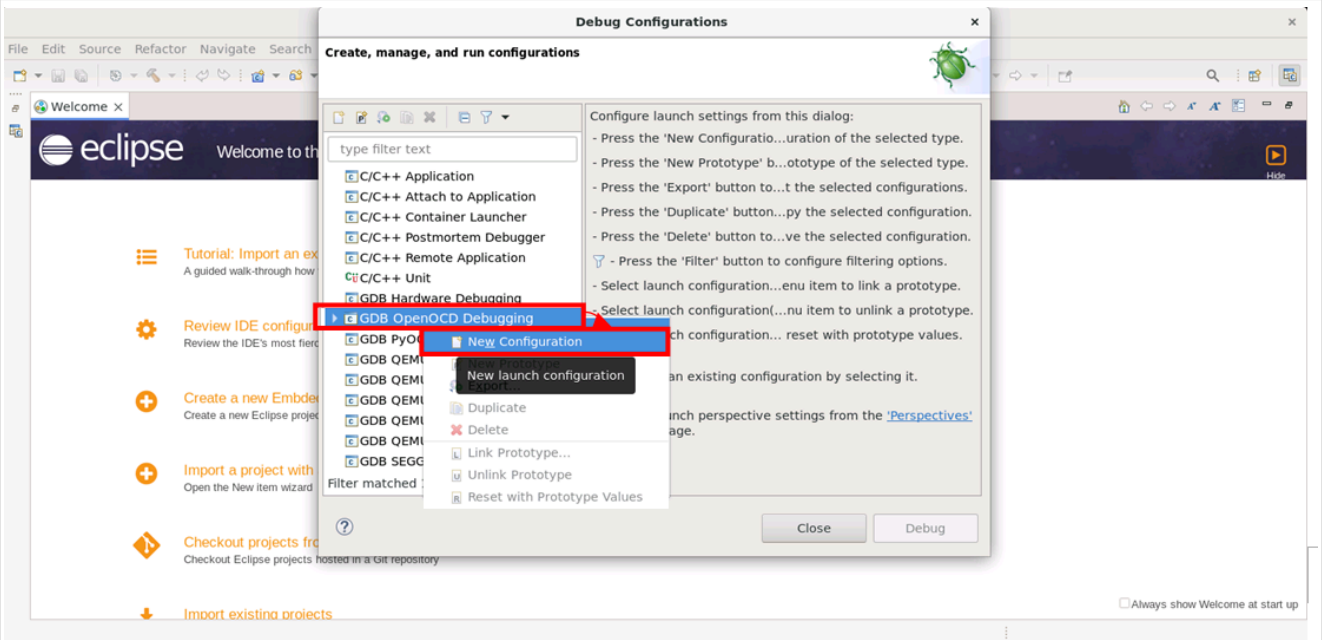

Figure 43. Create the debug configuration

3. Set **Main** of Debug Configuration.
   To do this, enter the **Name** and **Project**. To select the directory of RTOS that runs on the A core, click **Browse**.
   Note: *`hello_world_ca55_RTOS0_UART2.elf` is used as an example here. Other demo apps or customized apps can also be used.*

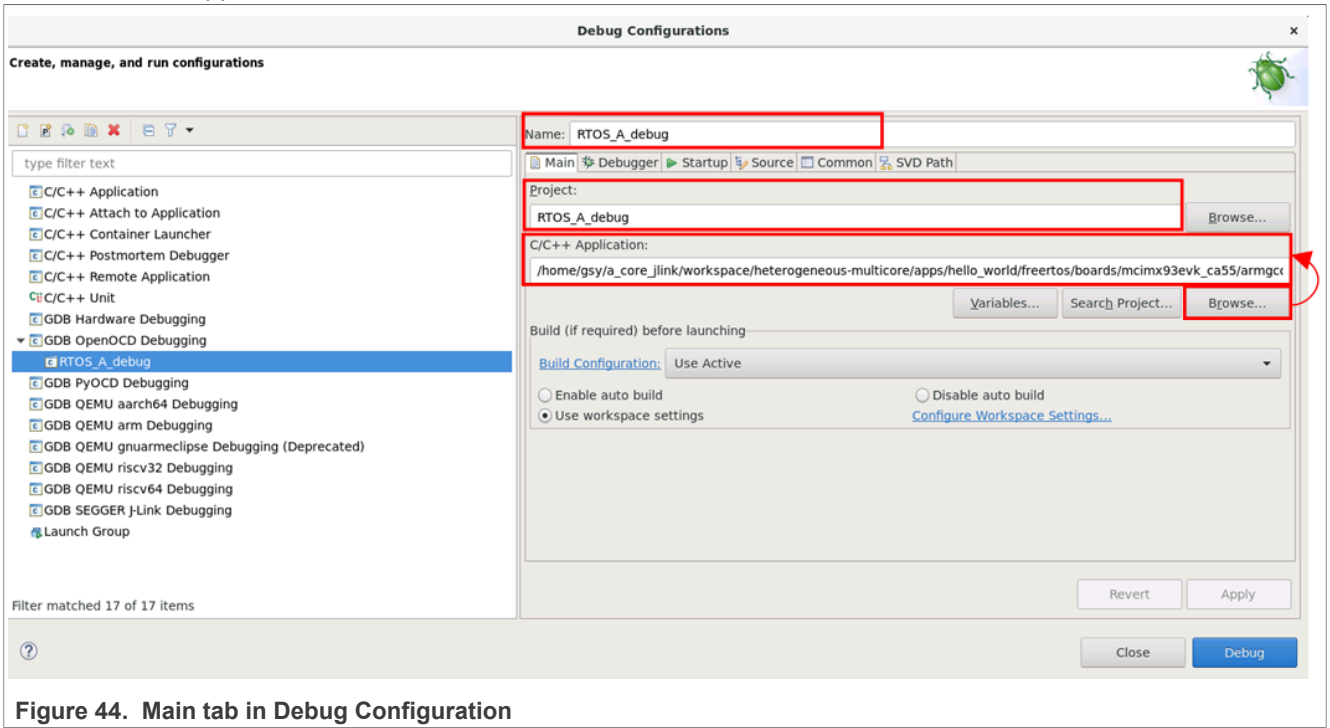

Figure 44. Main tab in Debug Configuration

4. Set OpenOCD in **Debugger**.

To do this, click **Browse** to select the directory of OpenOCD and enter **Config options** to select 'Jlink' and a specific board that is to be used:

```
-s <Path of tcl folder> -f <Path of jlink.cfg> -f <Path of board script> -c
 "gdb_breakpoint_override hard"
```

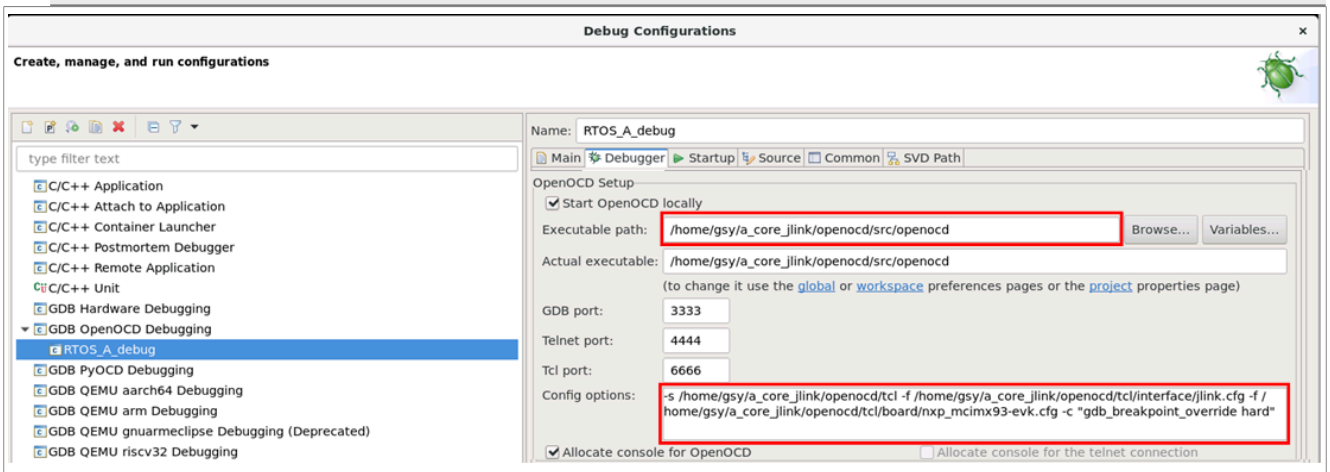

Figure 45. Debugger tab in Debug Configuration - 1

Note: *The debugging script with initial reset (nxp_mcimx93-evk-reset.cfg) is used here. If the initial reset is not wanted, the nxp_mcimx93-evk.cfg script could be used.*

Click **Browse** to select the directory of gdb as shown in Figure 46

Enter **Port number** with **3334** as '3334' is assigned to imx93.a55.1

Info : starting gdb server for imx93.a55.1 on 3334

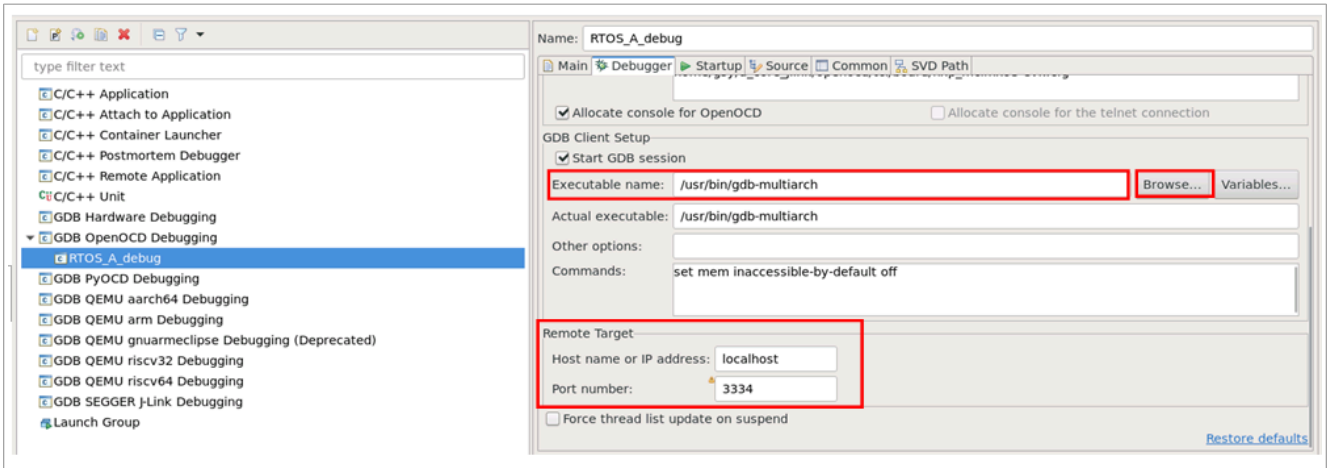

Figure 46. Debugger tab in Debug Configuration - 2

5. Unflag **Initial Reset** and **Pre-run/Restart reset** in **Startup** to avoid board resetting as shown in Figure 47

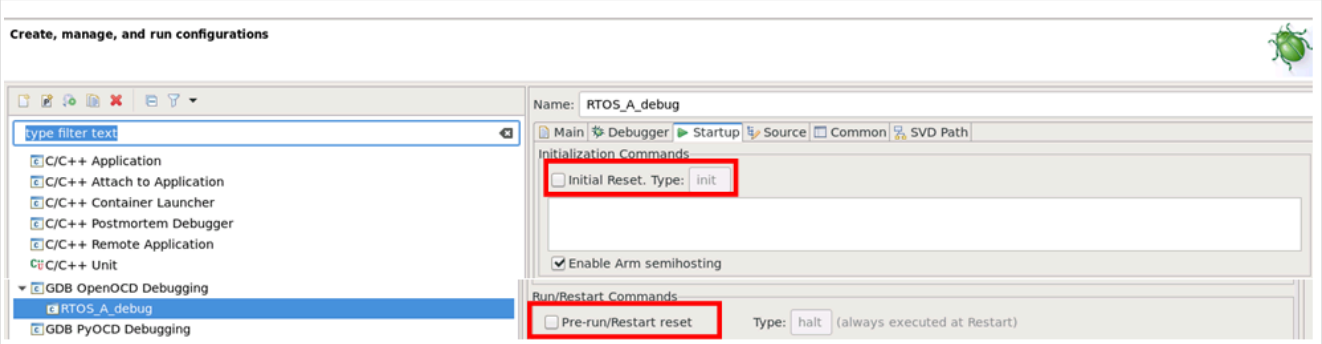

Figure 47. Startup tab in Debug Configuration

5.3.3 Eclipse debugging steps

1. Flash v2.8 Real-Time Image into eMMC or the SD card:

```
.\uuu.exe -b sd_all nxp-image-real-time-edge-imx93evk.wic
```

2. Boot the board and stop booting in U-Boot.
3. Enter the following command to boot the A core.

```
$ uboot => setenv boot_a1 "mw 0xA0000000 14000000; dcache flush; icache
 flush; cpu 1 release 0xA0000000"
$ uboot => setenv bootcmd "run boot_a1"
$ uboot => saveenv
```

4. If the configuration is set successfully in previous sections, power on the board.
5. Click **debug** to start debug.

Figure 48. Debug button

Note: *In the Console window, a successfully connected log could be observed.*

6. The program is set to stop at the main function automatically. Then, the address of the pc pointer that is pointing can be found.

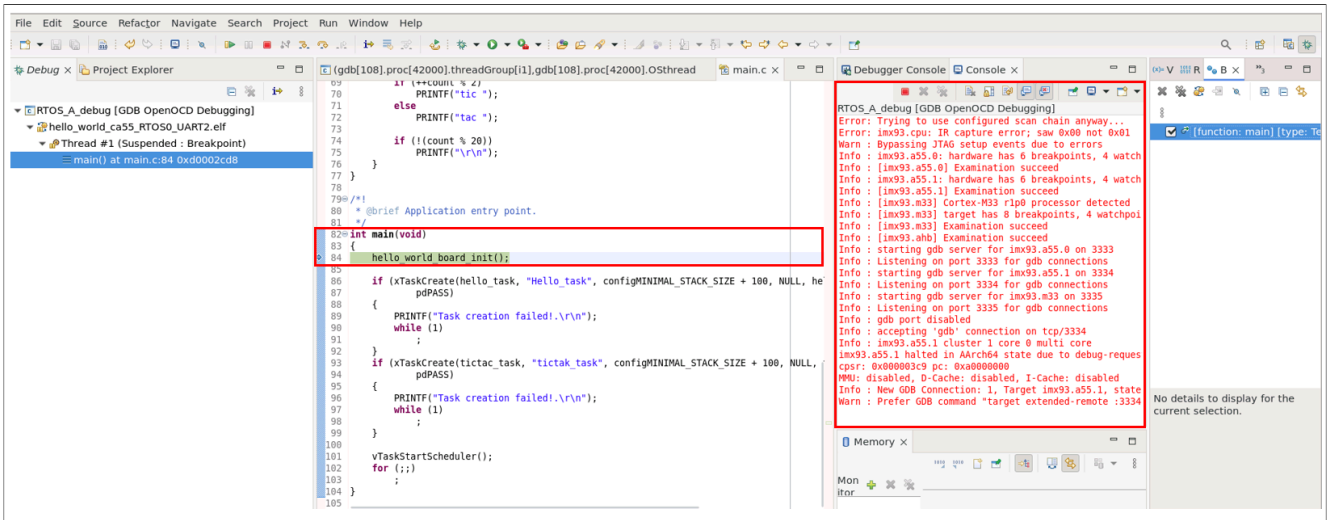

Figure 49. Stop at main function

7. Click **Step Into**, to go into the pointing function and execute the next command.

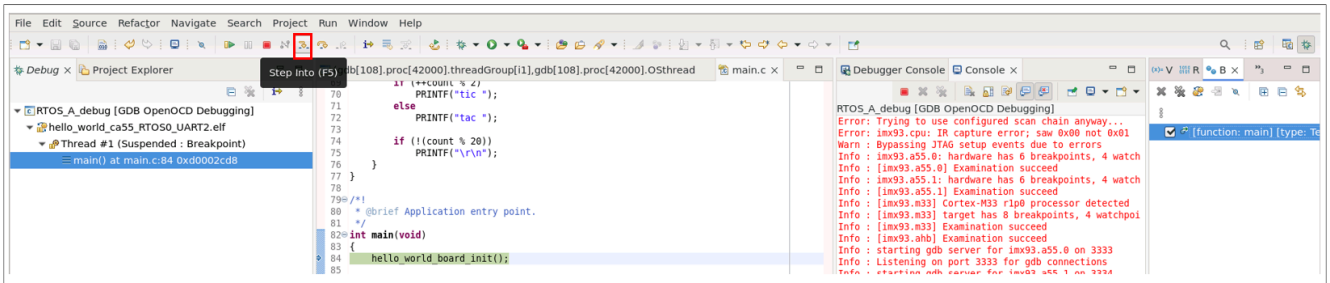

Figure 50. Step into debugging

8. **Step Over** could also be used to step over the next method call (without entering it).

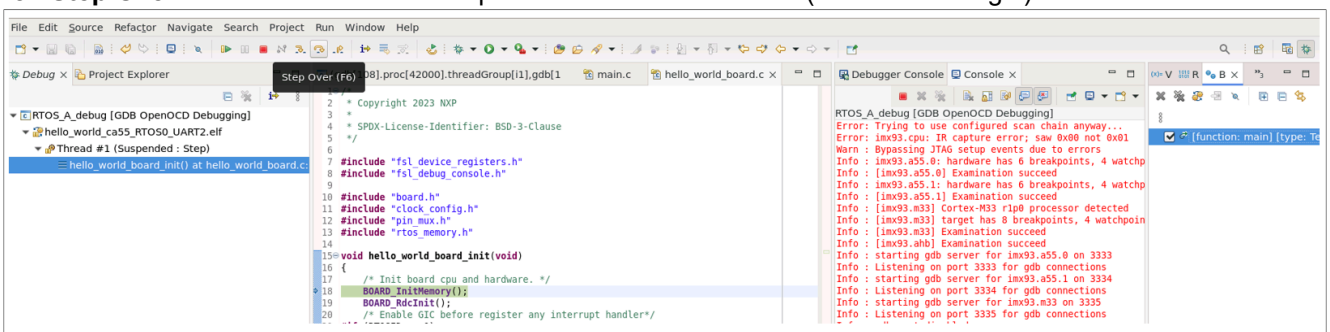

Figure 51. Step over debugging

9. After adding the break point, click **Resume** to run the program until the break-point is reached.
   To add the break point, double-click the command **Add break point**

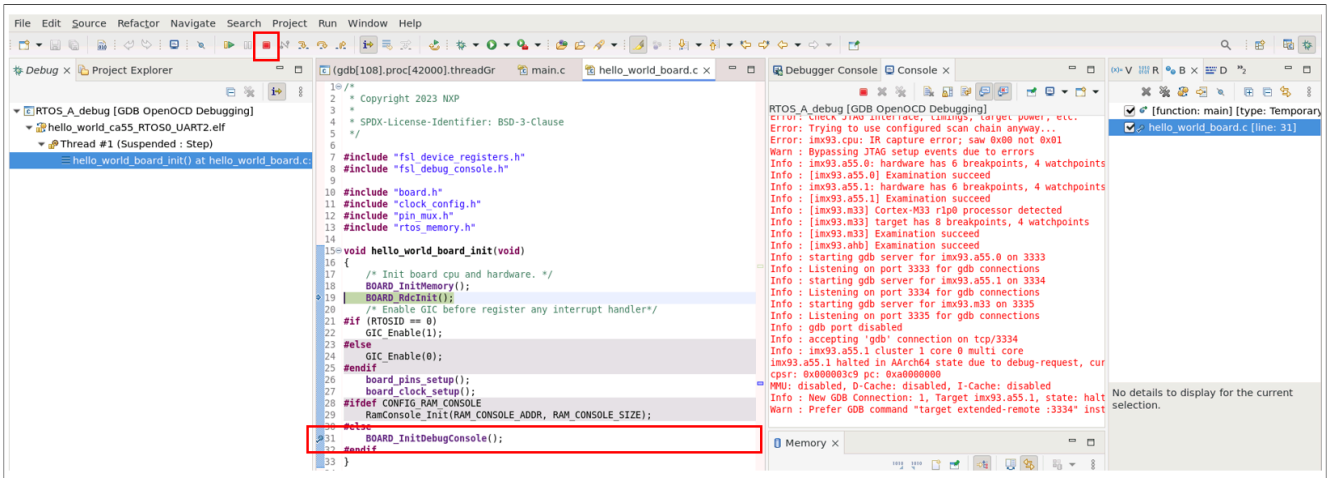

Figure 52. Instruction stepping mode

10. **Instruction stepping mode** can be selected to check the assembly.

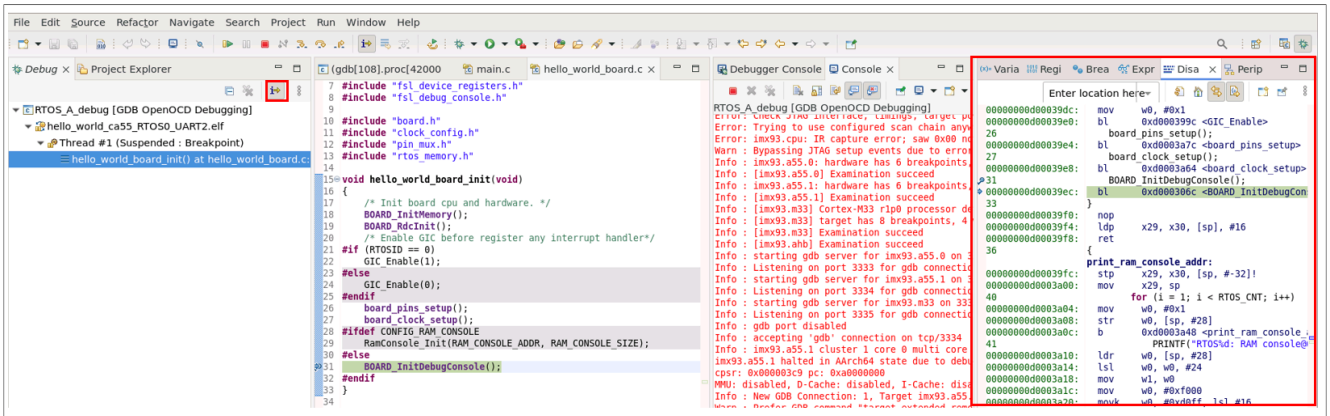

Figure 53. Instruction stepping mode

11. If editing and image rebuilding are required, click **Terminate** to terminate the debugging. After that, code editing and rebuilding could be done. For more information of image building, refer to Section 5.3.1.

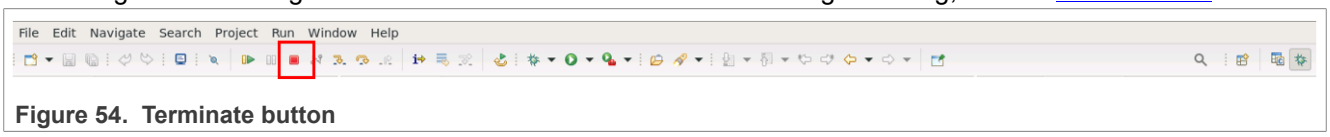

Figure 54. Terminate button

12. After the image is successfully re-build, click debug to start the debugging again.

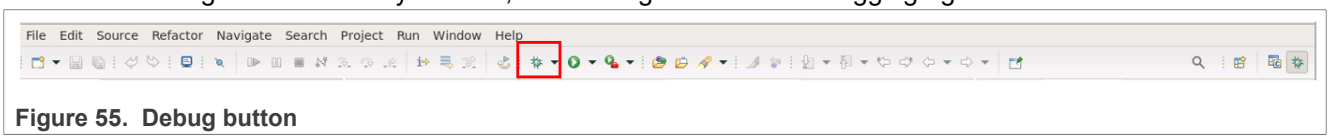

Figure 55. Debug button

5.4 Debugging Native RTOS by using OpenOCD with GDB

This example shows how to debug Native RTOS by using OpenOCD with GDB client command line on i.MX 8M Plus LPDDR4 EVK or i.MX 93 EVK. In this example, the J-Link probe is used to connect to the local JTAG port.

5.4.1 Setup and build RTOS

To keep debug information in RTOS images, change the gcc compile optimization level. “-O0” or “-Og” must be used.

For FreeRTOS applications in heterogeneous-multicore (https://github.com/nxp-real-time-edge-sw/heterogeneous-multicore), build debug images using `build_ddr_debug.sh`. The bin and elf images can be found in the `ddr_debug` directory.

For Zephyr, enable `CONFIG_DEBUG_OPTIMIZATIONS=y` to select the optimization option in applications `prj.conf`.

To check the code address and instruction, the image can be disassembled to get the dump information, for example:

```
~$ aarch64-none-linux-gnu-objdump -alD zephyr.elf > dump.s
```

Then open `dump.s` to check the code address and assemble instructions.

5.4.2 GDB Debugging steps

The heterogeneous-multicore (https://github.com/nxp-real-time-edge-sw/heterogeneous-multicore) hello_world application on the i.MX 8M Plus LPDDR4 EVK is used as an example.

1. Follow the steps in Section 5.4.1 to build the application debug image: `ddr_debug/hello_world_ca53_RTOS0_UART4.bin`
2. Connect the J-Link debugger to the EVK board and Linux host, power on the board and stop it at the U-Boot command line.
3. To boot the RTOS image, put the following in the U-Boot command line:

```
u-boot=> mw 0xA0000000 14000000; dcache flush; icache flush; cpu 1 release
0xA0000000
```

4. Start OpenOCD from Linux Host.

```
~$ sudo openocd -f interface/jlink.cfg -f board/nxp_imx8mp-evk.cfg -c
 "gdb_breakpoint_override hard"
```

Note: *After MMU is enabled on the Cortex-A Core, gdb soft breakpoint cannot be used, use hard breakpoints by running openocd with parameter* `-c "gdb_breakpoint_override hard.`
The following log can be found in Linux Host:

```
$ sudo openocd -f interface/jlink.cfg -f board/nxp_imx8mp-evk.cfg -c
"gdb_breakpoint_override hard"
Open On-Chip Debugger 0.12.0+dev-00559-g04154af5d (2024-04-29-12:00)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
force hard breakpoints
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : J-Link Ultra V6 compiled Apr 15 2024 17:37:59
Info : Hardware version: 6.00
Info : VTarget = 1.797 V
Info : clock speed 1000 kHz
Info : JTAG tap: imx8mp.cpu tap/device found: 0x5ba00477 (mfg: 0x23b (ARM
 Ltd), part: 0xba00, ver: 0x5)
Info : imx8mp.a53.0: hardware has 6 breakpoints, 4 watchpoints
Info : [imx8mp.a53.0] Examination succeed
Error: JTAG-DP STICKY ERROR
Error: [imx8mp.a53.1] Examination failed
Warn : target imx8mp.a53.1 examination failed
Error: JTAG-DP STICKY ERROR
Error: [imx8mp.a53.2] Examination failed
Warn : target imx8mp.a53.2 examination failed
Info : imx8mp.a53.3: hardware has 6 breakpoints, 4 watchpoints
Info : imx8mp.a53.3 cluster 0 core 3 multi core
Info : [imx8mp.a53.3] Examination succeed
Info : [imx8mp.m7] Cortex-M7 r1p2 processor detected
Info : [imx8mp.m7] target has 8 breakpoints, 4 watchpoints
Info : [imx8mp.m7] Examination succeed
Info : [imx8mp.ahb] Examination succeed
Info : starting gdb server for imx8mp.a53.0 on 3333
Info : Listening on port 3333 for gdb connections
Info : starting gdb server for imx8mp.a53.1 on 3334
Info : Listening on port 3334 for gdb connections
Info : starting gdb server for imx8mp.a53.2 on 3335
Info : Listening on port 3335 for gdb connections
Info : starting gdb server for imx8mp.a53.3 on 3336
Info : Listening on port 3336 for gdb connections
Info : starting gdb server for imx8mp.m7 on 3337
Info : Listening on port 3337 for gdb connections
Info : gdb port disabled
```

From the log, find the GDB server port for each CPU Core: four A53 Cores use 3333 3334 3335 and 3336 ports, and the M4 Core uses 3337.

5. Open the GDB client.
   Open the GDB client in another terminal window on Linux Host:

```
~$ gdb-multiarch -f hello_world_ca53_RTOS0_UART4.elf
```

   Use “-f” to specify the RTOS elf file, if the file is not in the current directory, add the location directory before the filename.
   Connect to the remote GDB server:

```
~$ gdb-multiarch -f hello_world_ca53_RTOS0_UART4.elf
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world_ca53_RTOS0_UART4.elf...
(gdb) target remote :3334
Remote debugging using :3334
Reset_Handler () at /work/work/realtimeedge/hypervisor-less-virtio/
workspace2/heterogeneous-multicore/os/freertos/Core_AArch64/core/armv8a/
startup.S:56
/work/work/realtimeedge/hypervisor-less-virtio/workspace2/
heterogeneous-multicore/os/freertos/Core_AArch64/core/armv8a/
startup.S:56:1001:beg:0xc0000004
(gdb) load
Loading section .interrupts, size 0x17d4 lma 0xc0000000
Loading section .text, size 0x13034 lma 0xc0002000
Loading section .init_array, size 0x40 lma 0xc0015040
Loading section .fini_array, size 0xf80 lma 0xc0015080
Loading section .data, size 0x1f0 lma 0xc0016000
Loading section .got, size 0x40 lma 0xc00161f0
Loading section .got.plt, size 0x18 lma 0xc0016230
Start address 0x00000000c0000000, load size 88592
Transfer rate: 63 KB/sec, 8053 bytes/write.
(gdb)
```

   In this demo, RTOS is running on Core1 of the Cortex-A53 Core on i.MX 8M Plus, connect to port 3334.

6. Set Breakpoint and Debug
   For example, set the breakpoint at the function `hello_task()`, change the “pc” register to 0xc0000008 to jump out of the “wfe” dead loop, from the below log find the CPU core stops at this breakpoint, and use the “next” command to run a single step:

```
(gdb) b hello_task
Breakpoint 1 at 0xc0002be0: file /work/work/realtimeedge/hypervisor-less-
virtio/workspace2/heterogeneous-multicore/apps/hello_world/freertos/main.c,
 line 49.
(gdb) c
Continuing.
```

```
Breakpoint 1, hello_task (pvParameters=0x0) at /work/work/realtimeedge/
hypervisor-less-
virtio/workspace2/heterogeneous-multicore/apps/hello_world/freertos/main.c:49
/work/work/realtimeedge/hypervisor-less-virtio/workspace2/heterogeneous-
multicore/apps/hello_world/freertos/main.c:49:1187:beg:0xc0002be0
(gdb) next
/work/work/realtimeedge/hypervisor-less-virtio/workspace2/heterogeneous-
multicore/apps/hello_world/freertos/main.c:52:1217:beg:0xc0002bec
```

Continue to use gdb commands to debug the RTOS application.

6 Reference materials

1. *Board Remote Control Utilities(BCU) Release Notes* (document BCU)
2. *OpenOCD User's Guide* (document Openocd)
3. *Real-time Edge Software User Guide* (document REALTIMEEDGEUG)
4. *i.MX Linux User's Guide* (document IMXLUG_6.6.23_2.0.0)
5. *i.MX Yocto Project User's Guide* (document IMXLXYOCTOUG_6.6.23_2.0.0)

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 4. Revision history

Document ID	Release date	Description
AN14367 v.1.0	01 October 2024	Initial version

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

1 Introduction 2
2 Definitions, acronyms, and abbreviations 2
3 Hardware and software setup 2
3.1 Hardware materials 2
3.2 Hardware connection 3
3.3 Software setup 3
3.3.1 Software setup on Ubuntu PC 4
4 U-Boot OpenOCD debugging 5
4.1 OpenOCD Debugging with JTAG Introduction 5
4.2 Debugging U-Boot with Eclipse IDE 6
4.2.1 Software requirements 6
4.2.2 Eclipse OpenOCD Configuration for U-Boot debugging 7
4.2.3 Eclipse debugging steps 16
4.2.4 Debugging U-Boot with on GDB terminal 25
4.2.4.1 Common commands 26
4.2.4.2 Stack commands 27
4.2.4.3 breakpoint 28
4.2.4.4 Control and view commands 30
4.2.4.5 GDB layout split commands 31
5 Native RTOS OpenOCD debugging 31
5.1 Native RTOS on Cortex-A Core introduction 31
5.2 EVK board software setup 31
5.3 Debugging Native RTOS with Eclipse IDE 32
5.3.1 Building Native RTOS with Eclipse IDE 32
5.3.2 Eclipse OpenOCD configuration for Native RTOS debugging 36
5.3.3 Eclipse debugging steps 40
5.4 Debugging Native RTOS by using OpenOCD with GDB 42
5.4.1 Setup and build RTOS 42
5.4.2 GDB Debugging steps 42
6 Reference materials 45
7 Note about the source code in the document 45
8 Revision history 45
Legal information 46