



# Airoha IoT SDK for BT Audio EVK Debug Application Note

Version: 1.1  
Release date: 18 July 2023

Use of this document and any information contained therein is subject to the terms and conditions set forth in Exhibit 1. This document is subject to change without notice.

## Airoha IoT SDK for BT Audio EVK Debug Application Note

## Document Revision History

Revision	Date	Description
1.0	18 May 2023	Initial version
1.1	18 July 2023	Added an introduction for the VS code debugging MCU, etc.

## Table of Contents

<b>Document Revision History.....</b>	<b>2</b>
<b>Table of Contents.....</b>	<b>3</b>
<b>List of Figures.....</b>	<b>3</b>
<b>List of Tables.....</b>	<b>4</b>
<b>1 Introduction .....</b>	<b>5</b>
<b>2 How to debug DSP with JTAG .....</b>	<b>6</b>
2.1 Setting up the JTAG debug environment for DSP .....	6
2.2 Code modification for DSP JTAG debug .....	7
2.3 Starting DSP JTAG debug.....	9
<b>3 How to debug MCU with JTAG.....</b>	<b>12</b>
3.1 Setting up the JTAG debug environment for MCU.....	12
3.2 Code modification for MCU JTAG debug.....	14
3.3 How to change pinmux configuration for JTAG .....	15
3.4 Starting MCU JTAG debug .....	15
3.5 How to configure FreeRTOS for task debug.....	16
3.6 How to debug MCU with VS Code.....	18
<b>4 Airoha software improvement for JTAG debugging .....</b>	<b>21</b>
4.1 How to auto hold DSP when debugging MCU.....	22
4.2 How to auto hold MCU when debugging DSP.....	24
<b>5 How to debug in system initialization stage .....</b>	<b>26</b>
<b>6 Troubleshooting .....</b>	<b>27</b>
<b>Exhibit 1 Terms and Conditions.....</b>	<b>29</b>

## List of Figures

Figure 1 DSP hardware configuration file .....	6
Figure 2 Xplorer QuickStart Wizard button.....	7
Figure 3 EPT tool.....	8
Figure 4 xt-ocd configuration.....	9
Figure 5 Xplorer Debug/Run/Profile configuration.....	10
Figure 6 Xplorer debug window.....	10
Figure 7 Xplorer debug perspective .....	11
Figure 8 Code modifications for FreeRTOS .....	16
Figure 9 Code modification in prvAddNewTaskToReadyList .....	17
Figure 10 OCD response for FreeRTOS.....	17
Figure 11 Tasks view from GDB.....	18
Figure 12 Cortex-Debug for VS Code .....	18
Figure 13 VS Code debug window .....	20
Figure 14 AT command for enable JTAG debugging.....	22
Figure 15 AB1585 DSPCFG_STALL register .....	23
Figure 16 Auto hold DSP GDB log .....	23
Figure 17 AB1585 WDT Register .....	24

## Airoha IoT SDK for BT Audio EVK Debug Application Note

Figure 18 Xplore GDB commands window .....	25
Figure 19 Xplorer air-continue command log .....	25
Figure 20 Xplorer source code mapping example .....	27
Figure 21 Search API in the disassembly window .....	28

## List of Tables

Table 1 DSP tool version list .....	6
Table 2 DSP hardware configuration file list .....	7

# 1 Introduction

This guide provides information about how to use the JTAG to debug MCU or DSP. It provides information about the subsequent items:

1. How to debug DSP with JTAG
  - Setting up the JTAG debug environment for DSP
  - Code modification for DSP JTAG debug
  - Debug DSP with Xplorer
2. How to debug MCU with JTAG
  - Setting up the JTAG debug environment for MCU
  - Code modification for MCU JTAG debug
  - Debugging with EVK from windows command line
3. Airoha software improvement for JTAG debugging
4. How to debug in system initialization stage
5. Troubleshooting

This information can be applied to the part of Airoha chip, including the AB1565, AB1568, AB1571, AB1577, AB1585, and AB1588. However, the software improvement for JTAG debugging ([Chapter 4](#)) is only supported on AB158x and AB157x.

**Note:**

The DSP JTAG debugger supports J-Link and Flyswatter2.

The MCU JTAG debugger supports J-Link, Tiny-2, etc.



## 2 How to debug DSP with JTAG

This chapter provides detailed information about how to debug DSP with JTAG. Including environment setup, code modification and debugging initiation.

### 2.1 Setting up the JTAG debug environment for DSP

This section provides a guide to setting the DSP JTAG environment base on windows, which includes:

- Install Cadence DSP IDE Xplorer
- Install XOCD

The Xplorer and XOCD version table as Table 1 DSP tool version list, you can download it from [here](#) (You must have an account to download Xplorer and XOCD).

**Table 1 DSP tool version list**

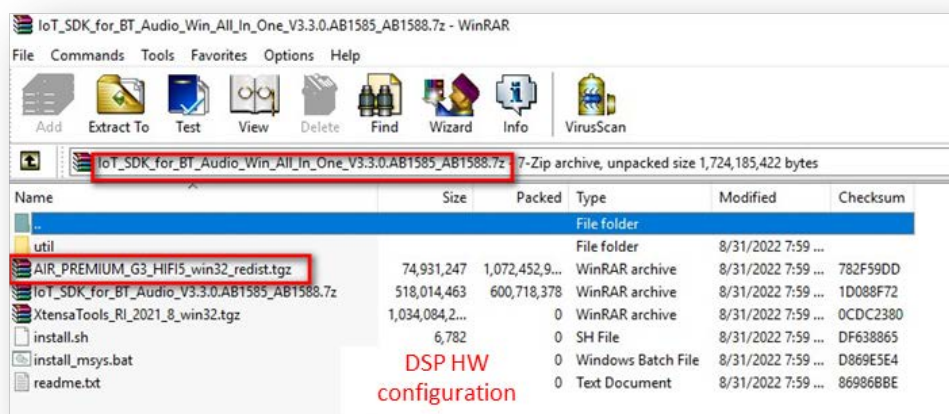
	Version of Xplorer	Version of XOCD
AB156x	9.0.18	14.0.8
AB157x	9.0.18	14.0.8
AB158x	9.0.18	14.0.8

#### 2.1.1 Install DSP IDE Xplorer

After downloading the corresponding version of Xplorer from Xtensa website, and install it with default setting.

Licenses are required to use Xplorer, so make sure that you have appropriate licenses installed before you use Xplorer to debug. To install your licenses, from the **Help** menu, select **Xtensa License Keys** to open the Xplorer License Keys dialog. Then, select **License Options -> Install Software Keys** to open the License setup dialog and input your license key.

After Xplorer and licenses key have installed, the DSP hardware configuration should be imported. The DSP hardware configuration is in the package IoT\_SDK\_for\_BT\_Audio\_Win\_All\_In\_One\_Vx.x.x. Such as Figure 1 DSP hardware configuration file for AB1585, and the corresponding DSP configuration version can refer to Table 2.



**Figure 1 DSP hardware configuration file**

## Airoha IoT SDK for BT Audio EVK Debug Application Note

Table 2 DSP hardware configuration file list

	Version of DSP configuration
AB156x	AB1568_i64B_d32B_512K_win32_redist.tgz
AB157x	AIR_STEREO_HIGH_G3_MINI_A_win32_redist.tgz
AB158x	AIR_PREMIUM_G3_HIFI5_win32_redist.tgz

Add the DSP hardware configuration steps as below:

Click the **Xplorer QuickStart Wizard** button (as Figure 2 Xplorer QuickStart Wizard) in the main toolbar.

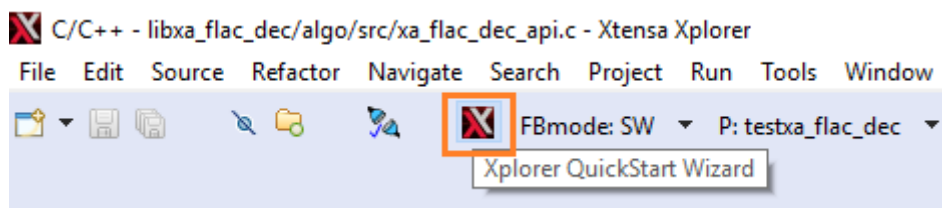


Figure 2 Xplorer QuickStart Wizard button

In the QuickStart Wizard, select **Create a new Xtensa Configuration -> Install a new build from downloaded bundle as the new configuration**, then click **Browse** and choose the corresponding DSP hardware configuration (\*\_win32\_redist.tgz) and then click **Add Build**.

## 2.1.2 Install XOCD

XOCD allows direct control over the connected processor cores through OCD (On-Chip Debugging) and does not require a stub running on the target. Thus, the target core can be stopped at any time to examine the state of the core, memory, and program, and, therefore, provides greater target visibility.

Download the correct XOCD from here, and install it follow the wizard flow.

## 2.2 Code modification for DSP JTAG debug

For JTAG debugging, the default software configuration may cause JTAG debug to fail, such as incorrect GPIO configuration that cause JTAG to be unable to connect, or system entering deep sleep that cause JTAG to disconnect. Therefore, it is necessary to make some code changes.

- Modify GPIO Pinmux configuration for DSP JTAG.
- Disable sleep feature.
- Disable system hang feature.
- Disable mask IRQ time check feature.

### 2.2.1 How to change Pinmux configuration for JTAG

The GPIO corresponding to the project is configured under the path `mcu/project/xxx/xxx_ref_design/inc/boards/xxx/ept_xxx.h`, you can modify it directly, or use the EPT tool to

## Airoha IoT SDK for BT Audio EVK Debug Application Note

modify the EPT configuration and generate new EPT files to replace old files.

As shown in Figure 3 EPT tool, the EPT tool can be download from MOL by searching with the keyword “Easy\_PinMux”. The EPT tool user guide document is located in the tool package folder and can be referred to for instructions on how to use the EPT tool.

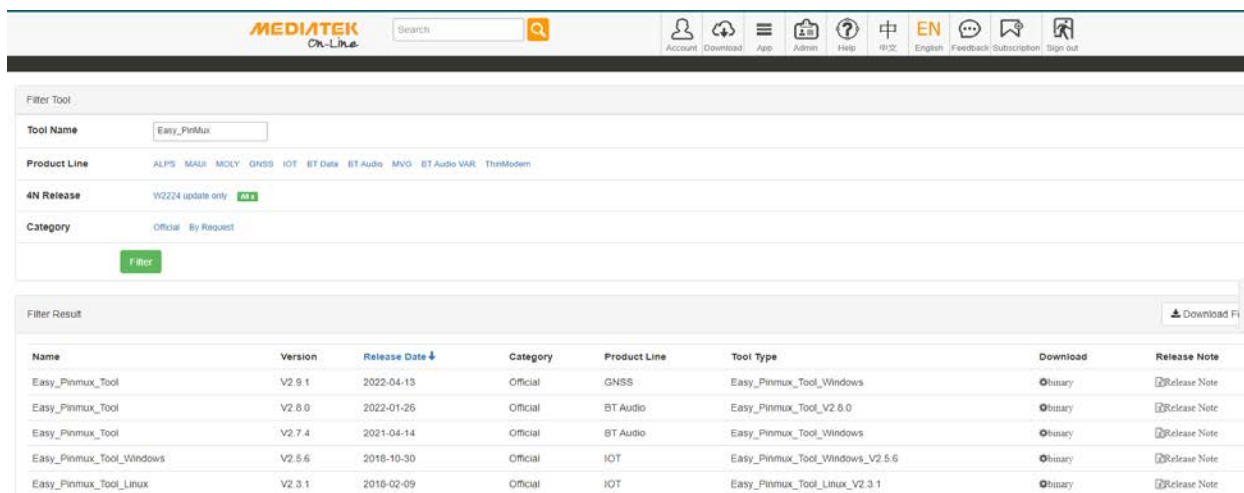


Figure 3 EPT tool

## 2.2.2 How to disable sleep

You can change the config option `USE_TICKLESS_IDLE` definition to 0 to disable the sleep feature. The option is in the file: `dsp/project/xxx/apps/xxx_ref_design/inc/FreeRTOSConfig.h`

```
#define configUSE_TICKLESS_IDLE 0
```

## 2.2.3 How to disable system hang feature

The system hang feature uses WDT to prevent the system from hanging, so stop or single-step during debugging may cause a WDT timeout, you can change option `AIR_SYSTEM_HANG_TRACER_ENABLE` to n to disable this feature. The option in the file: `mcu/config/chip/xxx/chip.mk`

```
AIR_SYSTEM_HANG_TRACER_ENABLE := n
```

## 2.2.4 How to disable mask IRQ time check feature

The `HAL_TIME_CHECK_ENABLED` define in `hal_feature_config.h` should be commented out.

```
//#define HAL_TIME_CHECK_ENABLED
```



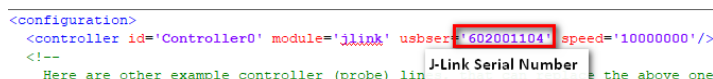
## 2.3 Starting DSP JTAG debug

This section provides a guide to starting DSP JTAG debug.

- 1) Modify xt-ocd configuration file (topology.xml) for the corresponding debugger, The following example shows a minimal configuration for the J-Link probe attached to the USB:

```
<controller id='Ctrl0' module='jlink' type='jtag' usbser='{J-Link Serial Number}' speed='{JTCK in Hz}'/>
```

Such as Figure 4 xt-ocd configuration:



```
<configuration>
<controller id='Controller0' module='jlink' usbser='602001104' speed='10000000' />
<!--
Here are other example controller (probe) link the above one
```

**Figure 4 xt-ocd configuration**

- 2) Open xt-ocd.  
You can open the Xtensa OCD Daemon from the Windows Start menu or start it with the following parameters: -dTD=20 -T 10 in the command-line. Do not open xt-ocd.exe directly.
- 3) Open Xplorer and choose your workspace. You must copy the xxx.out file into the workspace. Open the **Xplorer QuickStart Wizard** and select **Debug/Run/Profile my program**, then you must select the corresponding xxx.out under the workspace. The Debug/Run/Profile dialog should be set as shown below.

## Airoha IoT SDK for BT Audio EVK Debug Application Note

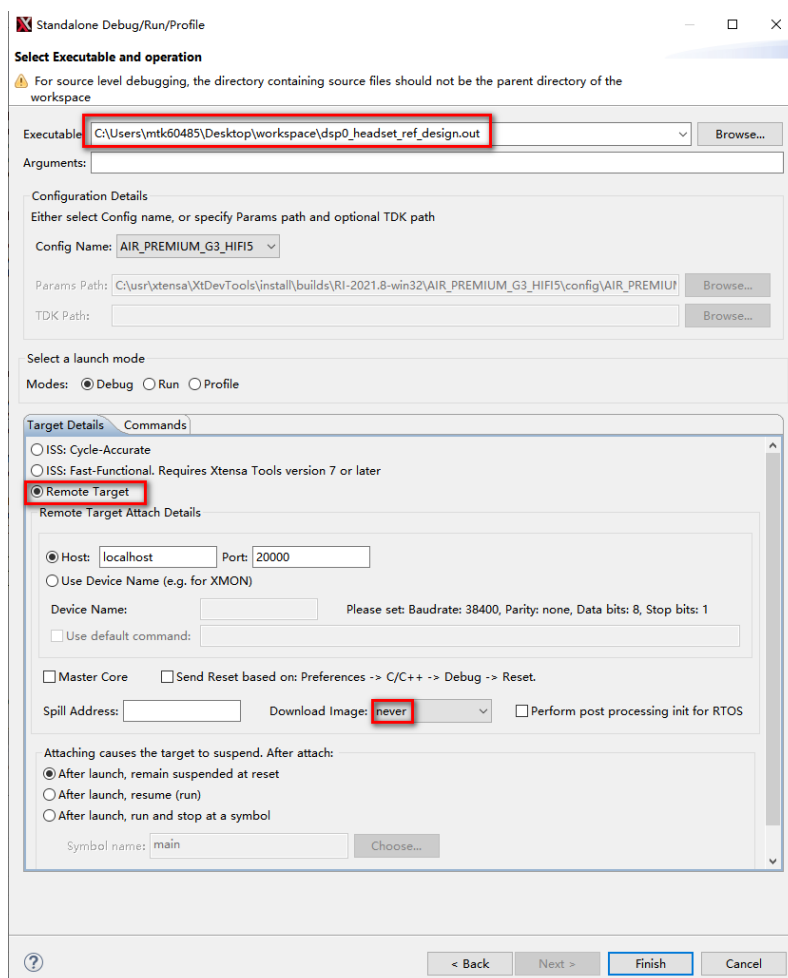


Figure 5 Xplorer Debug/Run/Profile configuration

Then click **Finish** and wait for the target to be attached, the main window of Xplorer should appear as shown Figure 6 Xplorer debug window.

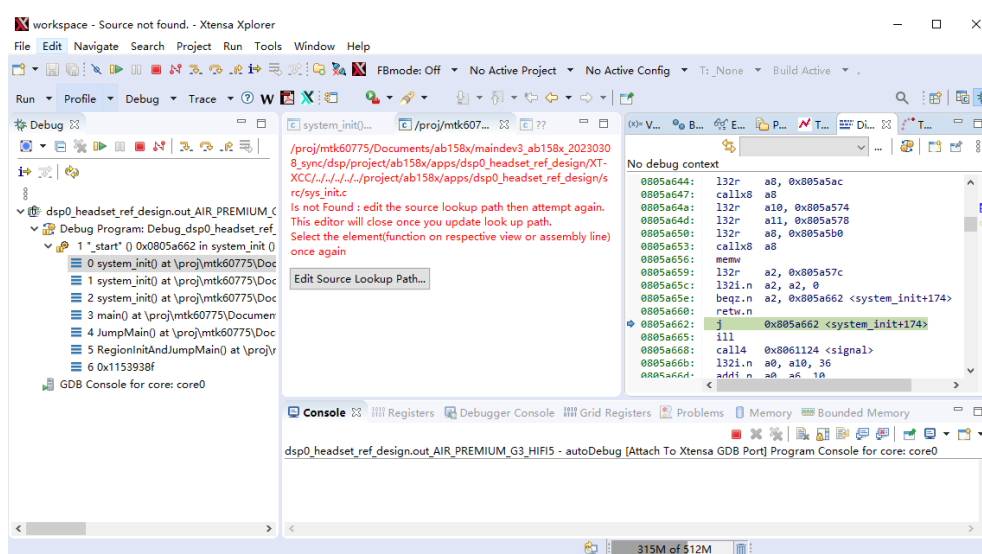
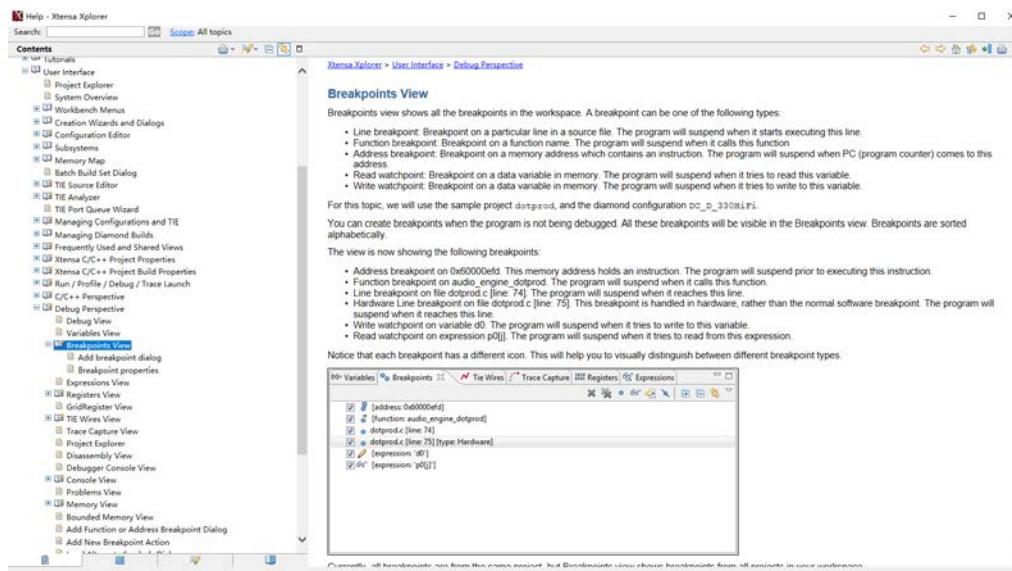


Figure 6 Xplorer debug window

## Airoha IoT SDK for BT Audio EVK Debug Application Note

For detailed instructions about using Xplorer debug, refer to **Help -> Help contents -> Xtensa Xplorer -> User Interface -> Debug Perspective**, such as Variables View, Breakpoints setting, etc.



**Figure 7 Xplorer debug perspective**

### 3 How to debug MCU with JTAG

This chapter provides detailed information about debugging MCU with JTAG.

#### 3.1 Setting up the JTAG debug environment for MCU

This section shows how to set up the JTAG debug environment for windows command line.

You must install the necessary software on Windows OS before starting project debugging,

- 1) Download the software from [here](#) and extract it into the <openocd\_root> folder (openocd-20211118 used by Airoha).
  - a) Download the GCC toolchain for your specific version of Windows from [here](#), and extract it into the <gcc\_root> folder.
- 2) Create an openocd configuration file named air\_chip.cfg and copy the following content into the file.
  - Example for CM33 core + J-LINK configuration:

```
puts "Load air_chip configuration"

source [find interface/jlink.cfg]
#source [find interface/ftdi/jtag-lock-pick_tiny_2.cfg]
transport select swd
source [find target/swj-dp.tcl]

set _CHIPNAME air_chip
set _TARGETNAME $_CHIPNAME.CM33
set _CPUTAPID 0x3ba02477

swj_newdap $_CHIPNAME cpu -irlen 4 -expected-id $_CPUTAPID
dap create $_CHIPNAME.dap -chain-position $_CHIPNAME.cpu
target create $_TARGETNAME cortex_m -dap $_CHIPNAME.dap

adapter_khz 1000
reset_config srst_only

$_TARGETNAME configure -event gdb-attach {
    global _TARGETNAME
    targets $_TARGETNAME
    halt
}

$_TARGETNAME configure -event gdb-detach {
    global _TARGETNAME
    targets $_TARGETNAME
    resume
}

puts "air_chip configuration done"
```

## Airoha IoT SDK for BT Audio EVK Debug Application Note

- example for CM4 core + J-LINK configuration:

```
puts "Load air_chip configuration"

#source [find interface/cmsis-dap.cfg]
source [find interface/jlink.cfg]
#source [find interface/ftdi/jtag-lock-pick_tiny_2.cfg]
transport select swd
#transport select jtag
source [find target/swj-dp.tcl]

set _CHIPNAME air_chip
set _TARGETNAME $_CHIPNAME.CM4
set _CPUTAPID 0x3ba02477

swj_newdap $_CHIPNAME cpu -irlen 4 -expected-id $_CPUTAPID
dap create $_CHIPNAME.dap -chain-position $_CHIPNAME.cpu
target create $_TARGETNAME cortex_m -dap $_CHIPNAME.dap

adapter_khz 1000
reset_config srst_only

$_TARGETNAME configure -event gdb-attach {
    global _TARGETNAME
    targets $_TARGETNAME
    halt
}

$_TARGETNAME configure -event gdb-detach {
    global _TARGETNAME
    targets $_TARGETNAME
}

puts " air_chip configuration done"
```

- 3) Put the board configuration file under <openocd\_root>\bin\.

### 3.2 Code modification for MCU JTAG debug

For JTAG debugging, the default software configuration may cause JTAG debugging to fail (e.g., an incorrect GPIO configuration can cause JTAG to be unable to connect; The system entering deep sleep can cause JTAG to disconnect). Therefore, it is necessary to make some changes to the code.

- Modify the GPIO Pinmux configuration for DSP JTAG.
- Disable the sleep feature.

## Airoha IoT SDK for BT Audio EVK Debug Application Note

- Disable the system hang feature.
- Disable the mask IRQ time check feature.

### 3.3 How to change pinmux configuration for JTAG

Refer to the Section 2.2.1 for details about setting the GPIO pinmux to JTAG mode.

#### 3.3.1 How to disable sleep

You can change the config USE\_TICKLESS\_IDLE define to 0 to disable sleep feature, the option in the file: mcu/project/xxx/apps/xxx\_ref\_design/inc/FreeRTOSConfig.h

```
#define configUSE_TICKLESS_IDLE 0
```

#### 3.3.2 How to disable system hang feature

The system hang feature uses WDT to prevent the system from hanging so stopping or single-stepping during debugging may cause a WDT timeout. You can change the option AIR\_SYSTEM\_HANG\_TRACER\_ENABLE to n to disable this feature. The option is in the file: mcu/config/chip/xxx/chip.mk

```
AIR_SYSTEM_HANG_TRACER_ENABLE := n
```

#### 3.3.3 How to disable mask IRQ time check feature

The HAL\_TIME\_CHECK\_ENABLED define in hal\_feature\_config.h should be commented out.

```
//#define HAL_TIME_CHECK_ENABLED
```

### 3.4 Starting MCU JTAG debug

After setting up the MCU JTAG debug environment, you can refer to the following steps to start debugging:

- 1) Copy the project .elf file from project Build folder to GCC tool path, such as <gcc\_root> (For example, copy <sdk\_root>\out\ ab1585\_evk\ headset\_ref\_design\debug\ headset\_ref\_design.elf to <gcc\_root>.)
- 2) Open the command window for openOCD.
- 3) Change the directory in the command window to the openOCD tool folder, such as <openocd\_root>\bin.
- 4) Use a debugger (such as J-Link) to connect to MCU JTAG slot and power on board.
- 5) Run the command to start the openOCD.

```
openocd.exe -f air_chip.cfg
```

- 6) Open another one command window for GDB.

## Airoha IoT SDK for BT Audio EVK Debug Application Note

- 7) Change the directory in the command window to tool folder, such as <gcc\_root>\bin.
- 8) Run the command to start the GDB.

```
arm-none-eabi-gdb.exe -f xxx.elf
(gdb) target remote localhost:3333
(gdb) info registers
```

### 3.5 How to configure FreeRTOS for task debug

By default, task-related information is not visible when performing command-line debugging. However, it is possible to implement this feature by making some modifications to the code. The detailed configuration method is as follows:

- 1) Add configure command (\$\_TARGETNAME configure -rtos auto) in air\_chip.cfg.

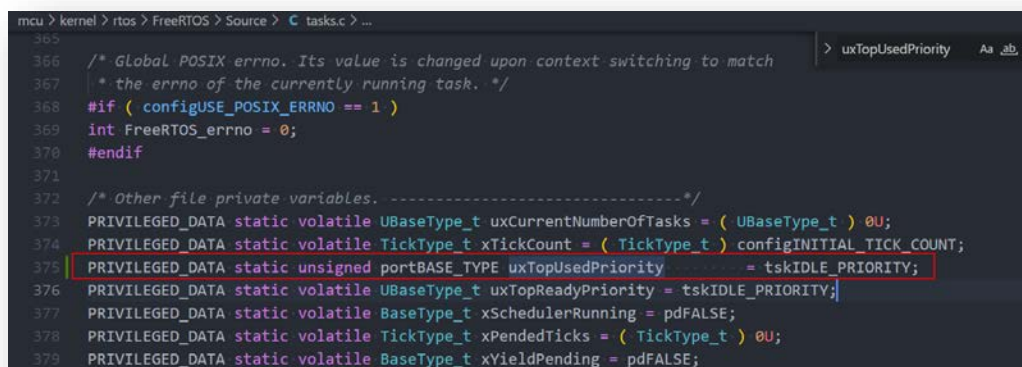
```
...
adapter_khz 1000
reset_config srst_only

$_TARGETNAME configure -rtos auto

$_TARGETNAME configure -event gdb-attach {
    global _TARGETNAME
    targets $_TARGETNAME
    halt
}
```

- 2) Code modifications:

Add code in mcu/kernel/rtos/FreeRTOS/Source/tasks.c as Figure 8 and Figure 9.



```
mcu > kernel > rtos > FreeRTOS > Source > C tasks.c > ...
365
366 /* Global POSIX errno. Its value is changed upon context switching to match
367  * the errno of the currently running task. */
368 #if ( configUSE_POSIX_ERRNO == 1 )
369 int FreeRTOS_errno = 0;
370 #endif
371
372 /* Other file private variables. -----*/
373 PRIVILEGED_DATA static volatile UBaseType_t uxCurrentNumberOfTasks = ( UBaseType_t ) 0U;
374 PRIVILEGED_DATA static volatile TickType_t xTickCount = ( TickType_t ) configINITIAL_TICK_COUNT;
375 PRIVILEGED_DATA static unsigned portBASE_TYPE uxTopUsedPriority = tskIDLE_PRIORITY;
376 PRIVILEGED_DATA static volatile UBaseType_t uxTopReadyPriority = tskIDLE_PRIORITY;
377 PRIVILEGED_DATA static volatile BaseType_t xSchedulerRunning = pdFALSE;
378 PRIVILEGED_DATA static volatile TickType_t xPendedTicks = ( TickType_t ) 0U;
379 PRIVILEGED_DATA static volatile BaseType_t xYieldPending = pdFALSE;
```

Figure 8 Code modifications for FreeRTOS



## Airoha IoT SDK for BT Audio EVK Debug Application Note

```

mcu > kernel > rtos > FreeRTOS > Source > C tasks.c > ...
1098         } else {
1099             mtCOVERAGE_TEST_MARKER();
1100         }
1101     }
1102
1103     uxTaskNumber++;
1104     #if ( configUSE_TRACE_FACILITY == 1 )
1105     if( pxNewTCB->uxPriority > uxTopUsedPriority )
1106     {
1107         uxTopUsedPriority = pxNewTCB->uxPriority;
1108     }
1109     #endif /* configUSE_TRACE_FACILITY */
1110
1111     #if ( configUSE_TRACE_FACILITY == 1 )
1112     {
1113         /* Add a counter into the TCB for tracing only. */
1114         pxNewTCB->uxTCBNumber = uxTaskNumber;
1115     }
1116     #endif /* configUSE_TRACE_FACILITY */
1117     traceTASK_CREATE( pxNewTCB );

```

Figure 9 Code modification in prvAddNewTaskToReadyList

Then you can get OpenOCD response as Figure 10 and you can use the command 'i thread' to list all tasks, as shown in the Figure 11.

```

C:\Windows\System32\cmd.exe - openocd.exe -f air_chip.cfg
Licensed under GNU GPL v2
libusb1 09e75e98b4d9ea7909e8837b7a3f00dda4589dc3
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Load air_chip configuration
DEPRECATED! use 'adapter speed' not 'adapter_khz'
air_chip configuration done
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : J-Link V12 compiled Mar 28 2023 17:00:20
Info : Hardware version: 12.00
Info : VTarget = 1.793 V
Info : clock speed 1000 kHz
Info : SWD DPIDR 0x0be12477
Info : AB158X.CM33: Cortex-M33 r0p4 processor detected
Info : AB158X.CM33: target has 8 breakpoints, 4 watchpoints
Info : starting gdb server for AB158X.CM33 on 3333
Info : Listening on port 3333 for gdb connections
Info : accepting 'gdb' connection on tcp/3333
target halted due to debug-request, current mode: Thread
xPSR: 0x61000000 pc: 0x081c093a psp: 0x0421f448
Warn : Prefer GDB command "target extended-remote :3333" instead of "target remote :3333"
Info : Auto-detected RTOS: FreeRTOS

```

Figure 10 OCD response for FreeRTOS

## Airoha IoT SDK for BT Audio EVK Debug Application Note

```
(gdb) i thread
Id      Target Id      Frame
* 13    Thread 69329984 (Name: IDLE, State: Running) prvIdleTask (pvParameters=<optimized out>)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/tasks.c:3177
12    Thread 69308912 (Name: UI_re) vClearInterruptMask (ulMask=269488144)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
11    Thread 69312104 (Name: bt_ta) vClearInterruptMask (ulMask=269488144)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
10    Thread 69315296 (Name: ATCI) vClearInterruptMask (ulMask=269488144)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
9     Thread 69317464 (Name: race) vClearInterruptMask (ulMask=269488144)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
8     Thread 69321168 (Name: AM_Ta) 0x00000000 in ?? ()
7     Thread 69323024 (Name: ui_sh) 0x00000000 in ?? ()
6     Thread 69335344 (Name: SYSDE) vClearInterruptMask (ulMask=269488144)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
5     Thread 69323184 (Name: vTest) vClearInterruptMask (ulMask=269488144)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
4     Thread 69333176 (Name: Tmr S) vClearInterruptMask (ulMask=80000)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
3     Thread 69285008 (Name: Linea) vClearInterruptMask (ulMask=269488144)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
2     Thread 69283352 (Name: charg) vClearInterruptMask (ulMask=269488144)
    at ../../../../../../kernel/rtos/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c:197
```

Figure 11 Tasks view from GDB

### 3.6 How to debug MCU with VS Code

For VS Code (Visual Studio Code) debug JTAG, you can download it from [here](#), and the plug-in Cortex-Debug Extension(as Figure 12) should be installed before debugging (The latest Cortex-Debug version requires GDB version greater than 8, otherwise an error will be reported, you can install an old version of the Cortex-Debug version to solve this problem).

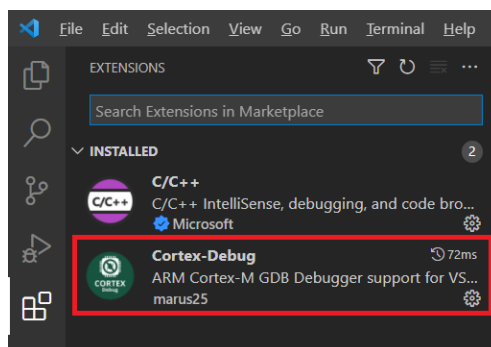


Figure 12 Cortex-Debug for VS Code

After installing VS code, the configuration settings.json and launch.json files must be configured. Please refer to the following steps for details:

- 1) Click **File** and select **Open Folder** to choose the corresponding source code path that will be used for debugging.
- 2) Click **Settings** and search for the keyword "cortex-" in the settings window, then click **Edit in settings.json** and configure the values for `cortex-debug.armToolchainPath` and `cortex-debug.JLinkGDBServerPath` as shown below:

## Airoha IoT SDK for BT Audio EVK Debug Application Note

```
{
  "C_Cpp.dimInactiveRegions": false
  "cortex-debug.armToolchainPath": "xxx\\gcc\\bin"
  "cortex-debug.JLinkGDBServerPath": "C:\\Program Files\\SEGGER\\JLink\\JLinkGDBServer.exe",
}
```

- 3) Open launch.json and configure it as below, the cwd and executable in the launch.json should be configured to the corresponding source code path.
- cwd:** Point out the current working directory, must be the main.c path, otherwise the breakpoint cannot work.
- executable:** Configure the xxx.elf path.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "jlink_debug",
      "type": "cortex-debug",
      "request": "attach",
      "cwd": "xxx\\mcu\\project\\ab158x\\apps\\headset_ref_design\\src",
      "externalConsole": true,
      "showDevDebugOutput": false,
      "servertype": "jlink",
      "interface": "swd",
      "executable": "<sdk_root>/out/ab1585_evk/headset_ref_design/debug/headset_ref_design.elf",
      "device": "Cortex-M33",
      "rtos": "FreeRTOS",
      "runToMain": true
    }
  ]
}
```

- 4) Click **Start Debugging** to start debugging. The debug window appears as shown in Figure 13:

## Airoha IoT SDK for BT Audio EVK Debug Application Note

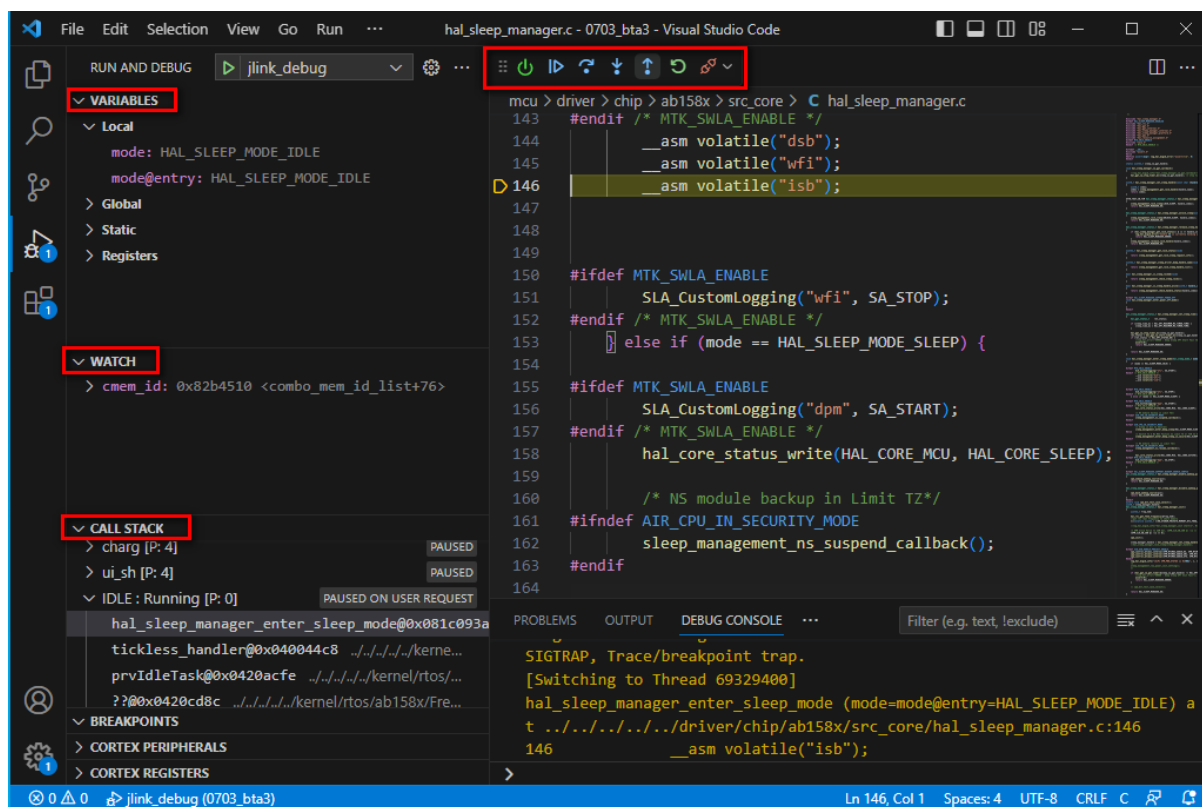


Figure 13 VS Code debug window

## 4 Airoha software improvement for JTAG debugging

This section introduces the software improvement for Airoha chip JTAG debugging. Due to the multi-core architecture of the Airoha chip, so when you use JTAG debugging, unexpected problems may occur during JTAG debugging, such as an exception occurring in another core when one core stops. To address these issues, our software has made some attempts to minimize this impact. We have added a feature to ensure that in certain basic scenarios, such as A2DP and ESCO, JTAG debugging will not cause system exceptions. This feature will be supported from SDK3.7.0.

You can send the AT command “AT+DEBUG=enable” to enable JTAG debug feature (Refer to Figure 14 AT command for enable JTAG debugging). This command includes disabling mask IRQ time check feature, disabling sleep feature, and disabling WDT timer (System hang feature). Additionally, this command disables the system log but the exception dump will still to take effect. Because the system supports multiple cores to print logs with the same hardware port (such as UART), the system log uses a hardware semaphore to ensure synchronized access. However, this may cause another core to fail to take the hardware semaphore and be pending.

For multi-core JTAG debugging, there are two ways to divide it:

- **Auto hold mode:** This means that when one core is stopped by JTAG, then the other cores will be stop immediately, for this mode, refer to sections 4.1 and 4.2 for detailed instructions.
- **Decoupled mode:** This means that when one core stopped by JTAG, then the other cores can continue running freely.

For decoupled mode, you only need to send ‘AT+DEBUG=enable’ to enable the debug mode and then start the JTAG debugging. Currently, this mode supported in A2DP, ESCO scenarios.



Note: This feature is supported by: AB157x, AB158x. Because the AT command is used for entering JTAG debug mode, this feature is NOT supported to debug in the system initialization stage. If you want to debug in system initialize stage, refer to [Chapter 5](#).

## Airoha IoT SDK for BT Audio EVK Debug Application Note

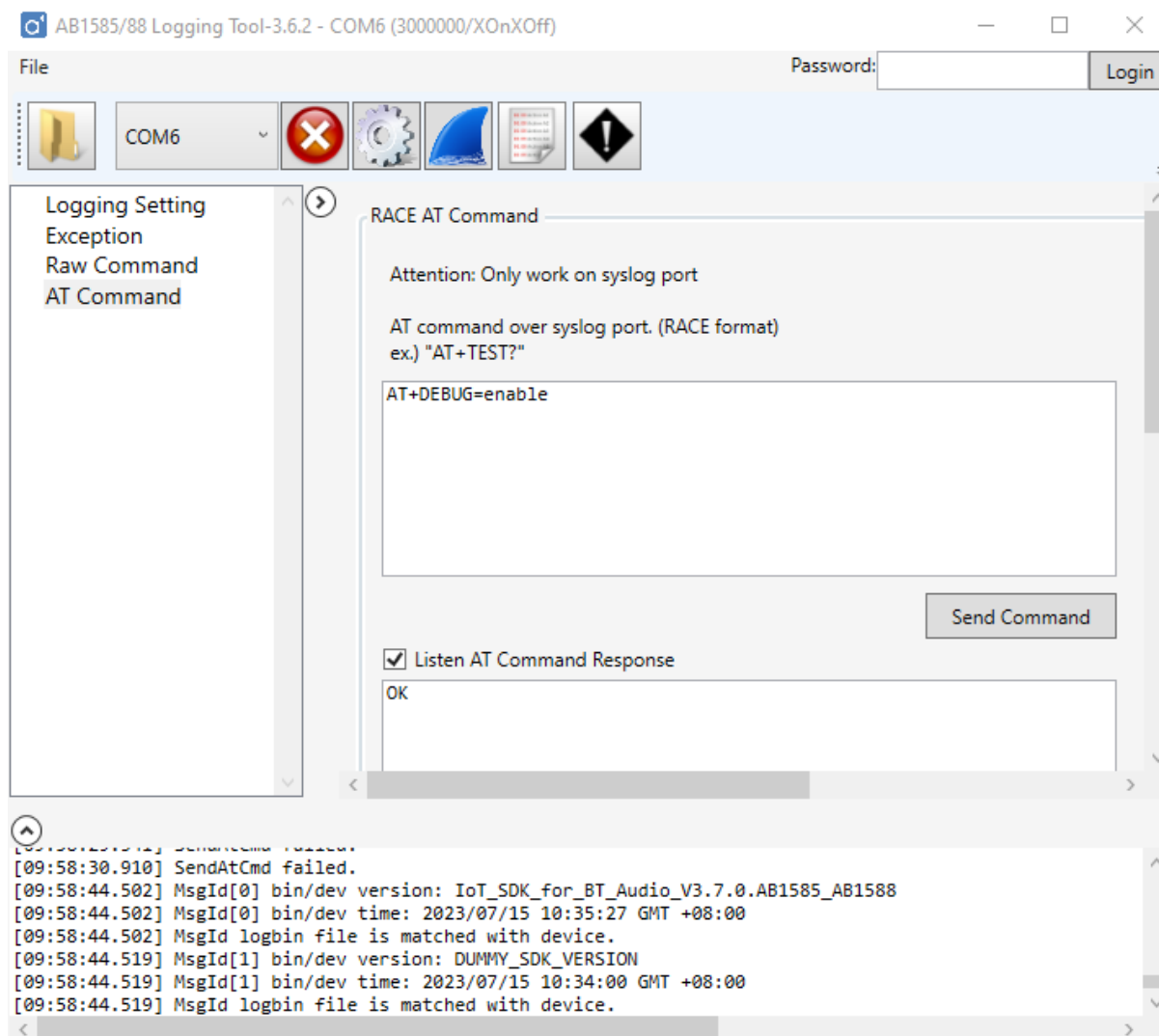


Figure 14 AT command for enable JTAG debugging

#### 4.1 How to auto hold DSP when debugging MCU

- 1) Add GDB command 'air-continue', located under <gcc\_root> \bin, to replace the GDB continue command.

For example, create the file 'air\_continue.txt' and define the command 'air-continue' within it.

```
define air-continue
  set *0x422C0018=0x0
  printf "continue run\n"
  continue
  set *0x422C0018=0x1
  printf "Stop DSP \n"
```

In this command, if MCU core is stopped by JTAG (breakpoint or target attached), executing 'set \*0x422C0018=0x1' will stop the DSP core (with bit 0 set to 1 in the register DSPCFG\_STALL). Executing the 'air\_continue' command will execute '\*0x422C0018=0x0', which releases the DSP to continue running freely

## Airoha IoT SDK for BT Audio EVK Debug Application Note

(with bit 0 set to 0 in the register DSPCFG\_STALL).

The address 0x422C0018 in air\_continue.txt is just an example for AB1585 (as shown in Figure 15 AB1585 DSPCFG\_STALL register). For other chips, refer to the corresponding ABxxxx\_Reference\_Manual.pdf.

422C0018	DSPCFG_STALL	32	DSP stall
----------	--------------	----	-----------

Figure 15 AB1585 DSPCFG\_STALL register

- 2) Add the GDB init command under <gcc\_root> \bin, such as air\_gdbinit.txt as below:

```
target remote:3333
source -v xxx\gcc\bin\air_continue.txt
set *0x422C0018=0x1
printf "Stop DSP \n"
```

When the core is attached with command 'arm-none-eabi-gdb.exe -f xxx.elf -x air\_gdbinit.txt', executing 'set \*0x422C0018=0x1' will stop the DSP. To apply the 'air-continue' command, use 'source -v <gcc\_root> \bin\air\_continue.txt'.

- 3) Power on the board and send the AT command "AT+DEBUG=enable" to enable JTAG debugging mode. If it successfully enters JTAG debug mode, "OK" appears in the AT command response dialog.

- 4) Refer to the Section 3.4 to start GDB debugging with the next cmd.

```
arm-none-eabi-gdb.exe -f xxx.elf -x air_gdbinit.txt
```

The GDB log is shown in Figure 16:

```
选择C:\Windows\System32\cmd.exe - .\bin\arm-none-eabi-gdb.exe -f headset_ref_design.elf -x .\bin\air_gdbinit.txt
Inferior 1 [Remote target] will be detached.
Quit anyway? (y or n) y
Detaching from program: E:\2833\customer_trianing\J-Link_test\gcc\headset_ref_design.elf, Remote target
Ending remote debugging.
E:\2833\customer_trianing\J-Link_test\gcc\bin\arm-none-eabi-gdb.exe -f headset_ref_design.elf -x .\bin\air_gdbinit.txt
GNU gdb (GNU Tools for ARM Embedded Processors) 7.10.1.20151217-cvs
Copyright (C) 2015 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 "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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 headset_ref_design.elf...done.
0x04000dec in gpt_start_free_run_timer (gpt=0x40255f0 <temp_reg>, clock_source=<optimized out>, divide=1)
at ../../../../../../driver/chip/ab158x/src/hal_gpt_internal.c:108
108 ../../../../../../driver/chip/ab158x/src/hal_gpt_internal.c: No such file or directory.
+define air-continue
Stop DSP
(gdb)
```

Figure 16 Auto hold DSP GDB log

## Airoha IoT SDK for BT Audio EVK Debug Application Note

## 4.2 How to auto hold MCU when debugging DSP

You use NMI to stop the MCU core because there is no register. We have implemented this feature in the NMI handler based on SDK3.7.0. Complete the following procedure to auto stop the MCU core.

- 1) Add the command in Xplorer command window as follows (Example for AB1585); The command window appears as shown Figure 18 Xplore GDB commands window.

```
set *0x420B00D4=0x12345678
set *0x420B0040=0x456789ab
printf"hang"
define air-continue
x 0x420B0020
set *0x420B00D4=0x0
printf"continue"
continue
set *0x420B00D4=0x12345678
set *0x420B0040=0x456789ab
printf"hang"
```

The addresses 0x420B00D4, 0x420B0020, and 0x420B0040 are described in Figure 17 AB1585 WDT Register. For other chips, refer to the corresponding ABxxxx\_Reference\_Manual.pdf and configured to the correct address.

420B00D4	<u>RETN_DAT5</u>	32	Retention Data 5 Register
420B0020	<u>WDT_INT</u>	32	Watchdog Timer Interrupt Register
420B0040	<u>WDT1_SW_RST</u>	32	Watchdog Timer 1 Software Reset Register

**Figure 17 AB1585 WDT Register**



## Airoha IoT SDK for BT Audio EVK Debug Application Note

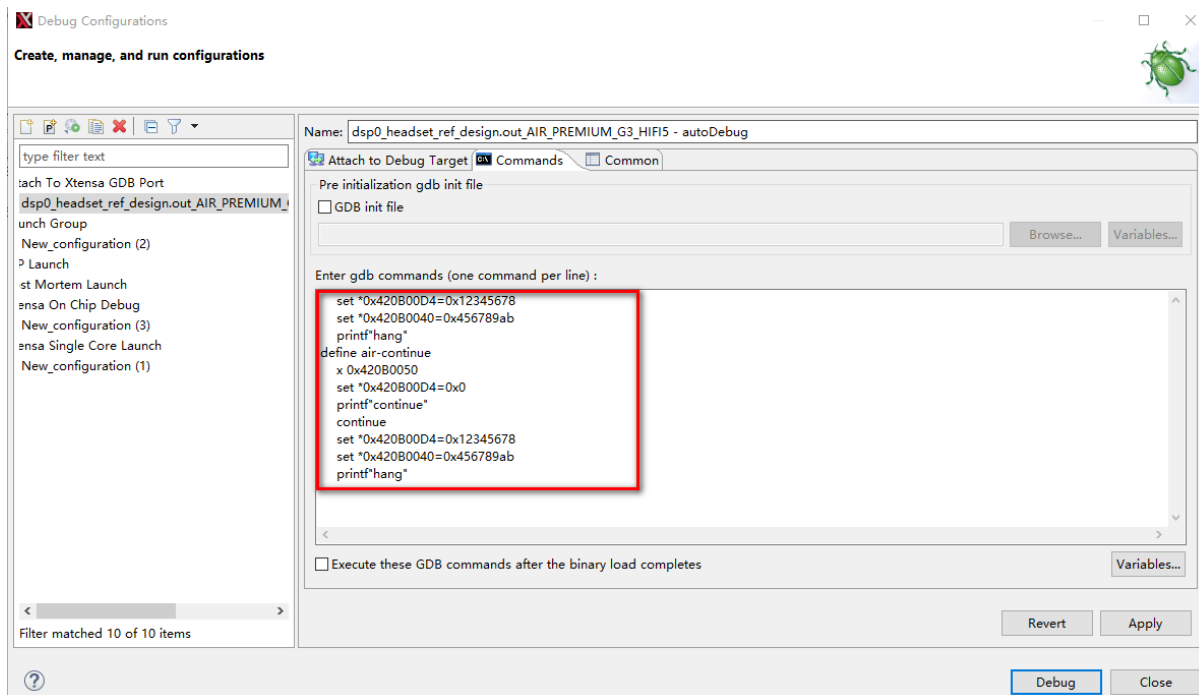


Figure 18 Xplorer GDB commands window

Note: When the DSP is stopped by JTAG after adding the above command, you cannot use the **Resume** button to continue DSP execution. Instead, send the 'air-continue' command in the Debugger Console dialog to continue DSP execution.

The air-continue log in Xplorer Debugger Console dialog is shown in Figure 19.

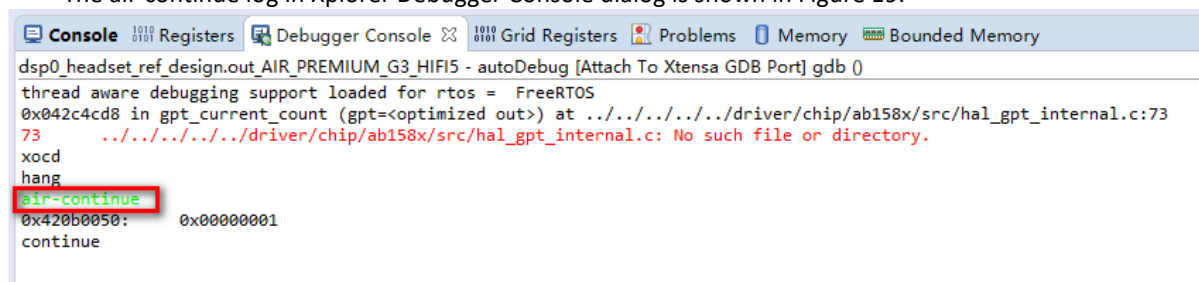


Figure 19 Xplorer air-continue command log

## 5 How to debug in system initialization stage

To enable debugging during system initialization, you must modify the code according to [Chapter 2](#) and [Chapter 3](#), and start JTAG debug accordingly. The GPIO pinmux configured in the function `bsp_ept_gpio_setting_init()`, which is called by `system_init()`. If you want to debug before this function is called, you should use `hal_pinmux_set_function()` to configure GPIO for JTAG before you start the JTAG debug. Additionally, since there will be some multi-core communication during the system initialization stage, you must change set JTAG debug flag to 0xFF to ignore any assert that may occur due to this communication (The JTAG debug flag not supported on AB156x). The JTAG debug flag should be set as follows:

```
*(volatile uint32_t *) HW_SYSRAM_PRIVATE_MEMORY_DSP_ICE_DEBUG_START = 0xff;
```

Additionally, since the system will run through the system initialization process soon after booting up, it is possible that the core PC has already run past the point where you want to stop when you attach to JTAG. To prevent this, you must add a while loop to wait for the JTAG attachment. When the JTAG is attached, you can change the while loop flag to let the system run. The following sample code is for your reference.

```
#include "hal_resource_assignment.h"
volatile uint32_t JTAG_wait_flag = 0;
int main(void)
{
    hal_pinmux_set_function(HAL_GPIO_000, 000); /*Set GPIO pinmux to JTAG mode.*/

    #ifdef AIR_ICE_DEBUG_ENABLE
        /* Set JTAG debug flag. */
        *(volatile uint32_t *) HW_SYSRAM_PRIVATE_MEMORY_DSP_ICE_DEBUG_START = 0xff;
    #endif

    while (1) {
        if (JTAG_wait_flag != 0) {
            break;
        }
    }
    ...
}
```

## 6 Troubleshooting

- GPIO Pinmux has not been configured to JTAG mode.
- Hardware jumper settings are incorrect.

For AB158x EVK, you need confirm the Jumper setting as below:

AP JTAG (GPIO0, GPIO1): Jump J301, J306 pin1 to pin2.

DSP JTAG (GPIO2, GPIO3, GPIO4, GPIO5, GPIO6): Jump J309, J312, J317, J320, J322 pin1 to pin2.

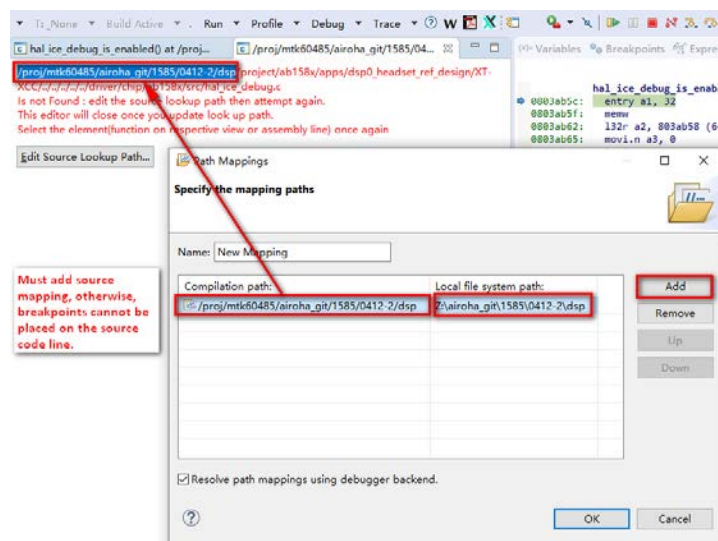
For AB157x EVK, you need confirm the Jumper and resistor setting as below:

AP JTAG (GPIO3, GPIO6): R721 & R717 mount 0ohm resistor, and remove R719, R720, R714, R736.

DSP JTAG (GPIO): R713, R709, R731, R725 mount 0ohm resistor, and jump J708, J706, J6615 pin1 to pin2, and remove R711, R712, R716, R706, R707, R727, R728, R723.

- WDT has not been disabled.
- The system hang feature is not disabled.
- Line breakpoint setting

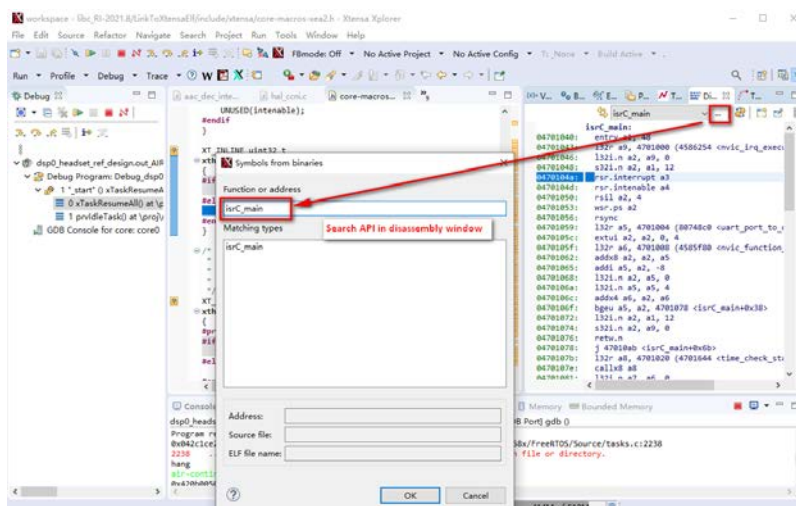
The line breakpoint may not work if you do not map the correct source code path. To add the source code mapping, select **Window → Preferences → “Source Lookup Path” → Add Path Mapping** to add source code mapping. Refer to Figure 20 Xplorer source code mapping example.



**Figure 20 Xplorer source code mapping example**

After configuring the source code mapping, if you open a source code file directly and try to set a line breakpoint, it may not work. To set a line breakpoint, search for an API in the disassembly window (as Figure 21 Search API in the disassembly window) and double-click disassembly code, this will automatically display the source code and then the line breakpoint can be successfully set. Otherwise, you can set the line breakpoint in Debugger Console window with GDB command.

## Airoha IoT SDK for BT Audio EVK Debug Application Note



**Figure 21 Search API in the disassembly window**

For some files, you may not still be able to set the breakpoint. In this case, you can refer to the following methods to add line breakpoint:

- Hardware breakpoint used for flash code

The DSP side only supports two hardware breakpoints and only a hardware breakpoint can be set on flash (flash address starting with 0x08xx\_xxxx). Otherwise, the breakpoint on flash will not work.

- Use GDB command to set line breakpoint.

## Airoha IoT SDK for BT Audio EVK Debug Application Note

## Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this “Document”) is subject to your (including the corporation or other legal entity you represent, collectively “You”) acceptance of the terms and conditions set forth below (“T&C”). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don’t agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to Airoha Technology Corp. and/or its affiliates (collectively “Airoha”) or its licensors and is provided solely for Your internal use with Airoha’s chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against Airoha or any of Airoha’s suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify Airoha for any loss or damages suffered by Airoha for Your unauthorized use or disclosure of this Document, in whole or in part.

Airoha and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. Airoha does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY AIROHA IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. AIROHA SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. AIROHA SHALL NOT BE RESPONSIBLE FOR ANY AIROHA DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, Airoha makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Airoha assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating Airoha’s product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.