T-Head DebugServer User Guide

Revision 5.14

Security Pubilc

T-Head Semiconductor Co., LTD

Address:	West Building T6, UK Center, EFC, No. 1112 Xiangwang Street, Yuhang District, Hangzhou
Zip code:	31112
Website:	www.T-Head.cn

Contents

List of Figures and Tables

S

1. Introduction

This user guide describes the features of T-Head DebugServer (T-HEAD DebugServer). This software supports Windows and Linux platforms. Therefore, the descriptions are divided by platform.

1.1. Terms

Table 11—1 Terms and definitions

Term	Description
DDC	A direct download channel. Such a channel can significantly accelerate data download.
T-HEAD DebugServer	This software is a proxy debug server program.
ICE	An in-circuit emulator. In this document, CKLink is used as an example of an ICE.

1.2. Feature description

T-HEAD DebugServer receives a primitive debug command sent by T-Head GDB and then sends a command to the hardware debug API (HAD) based on the Joint Test Access Group (JTAG) protocol. Then, T-HEAD DebugServer controls the execution of debug commands, obtains debug data, and returns the debug data to T-Head GDB. T-Head GDB communicates with T-HEAD DebugServer in socket mode. T-Head GDB and T-HEAD DebugServer can run on different hosts. T-HEAD DebugServer communicates with the target machine via ICE based on the JTAG protocol.

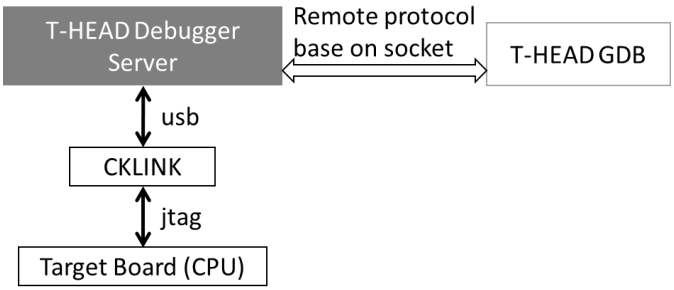


Figure 11-1 T-HEAD DebugServer — Server communication method

2. T-HEAD DebugServer for Windows

T-HEAD DebugServer can run in the graphical user interface (UI edition) and the command line interface (console edition) in Windows host systems.

2.1. Install T-HEAD DebugServer and the ICE driver

The ICE driver is packaged in the installation package of T-HEAD DebugServer. When you install T-HEAD DebugServer, select ICE Driver. The driver file will be copied to the system directory during installation of T-HEAD DebugServer. After you plug in an ICE device, Windows automatically installs the driver for the ICE.

2.1.1.Obtain the installation package

Obtain the installation package from the Open Chip Community (OCC) platform of T-Head at https://occ.T-Head.cn/community/download_detail?id=616215132330000384 or from Customer Service. The installation package includes the compressed T-Head-DebugServer-windows*.zip file for Windows systems and two T-Head-DebugSever-linux-*.sh files for 32-bit and 64-bit Linux systems.

2.1.2.Install T-HEAD DebugServer

1. Decompress the T-Head-DebugServer-windows*.zip file and double-click the Setup.exe file. On the installation page of T-HEAD DebugServer that appears, click Next.

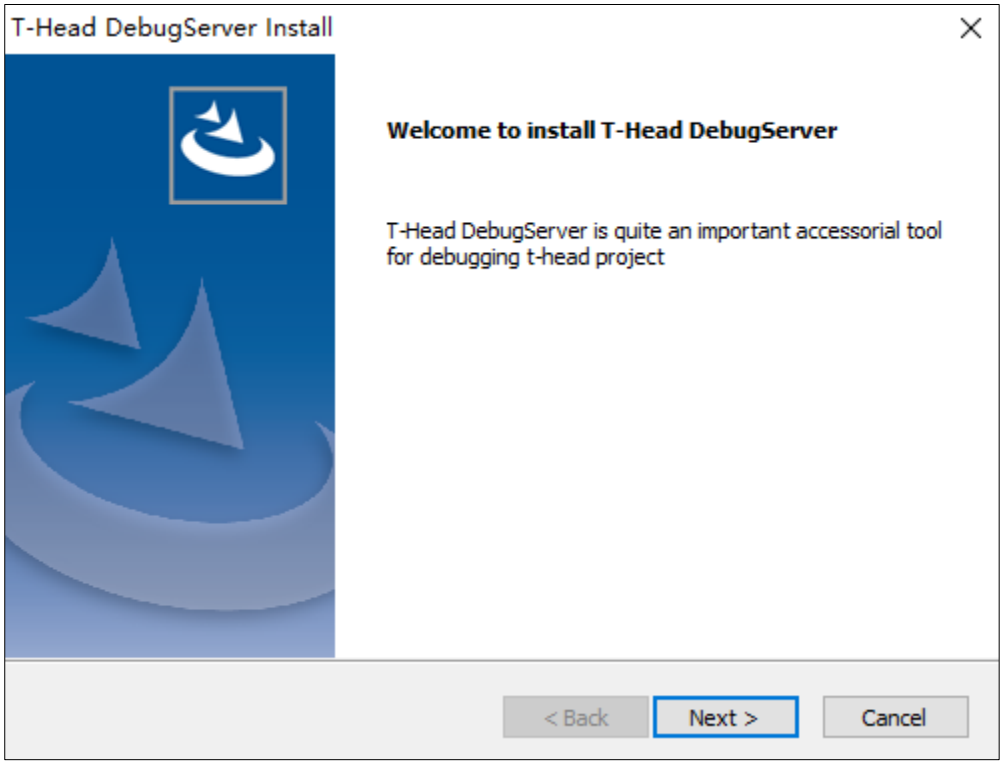


Figure 11-2 Install T-HEAD DebugServer — Step 1

2. Read through the license agreement of THServer and click Yes to continue.

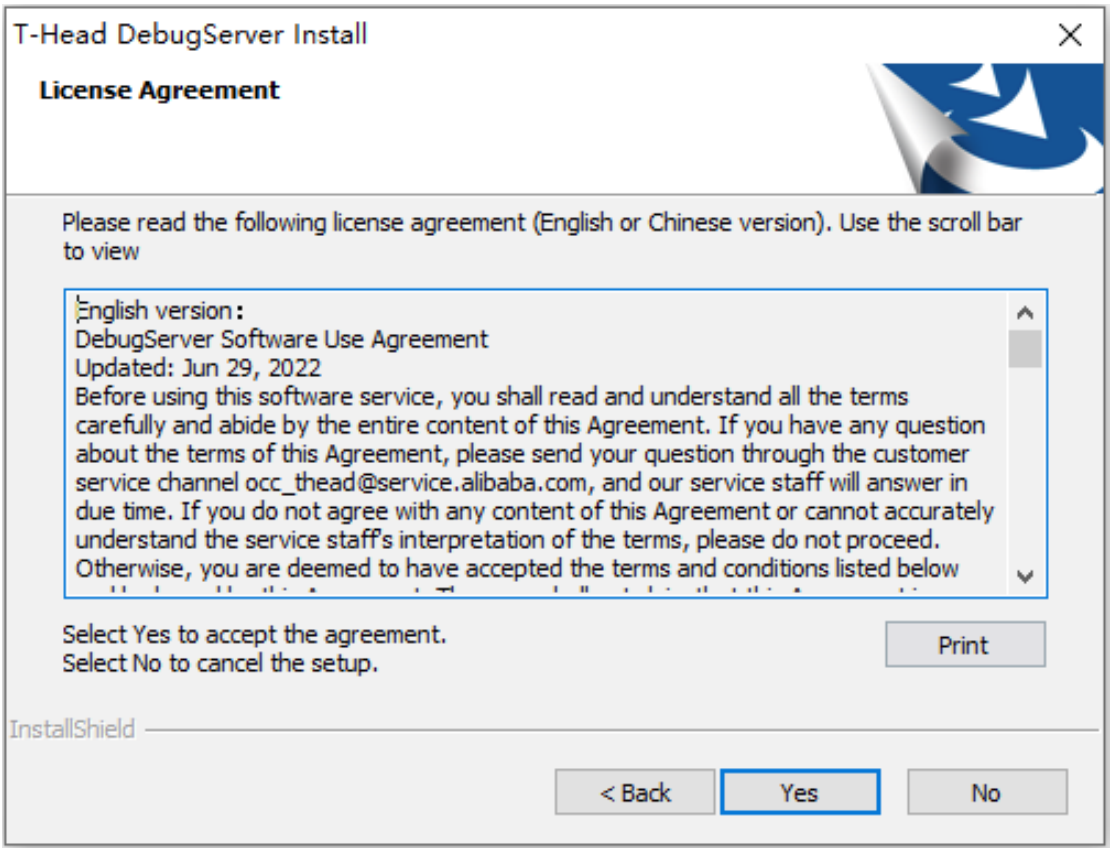


Figure 11-3 Install THServer — Step 2

3. Enter a user name and a company name, select applicable users, and click Next.

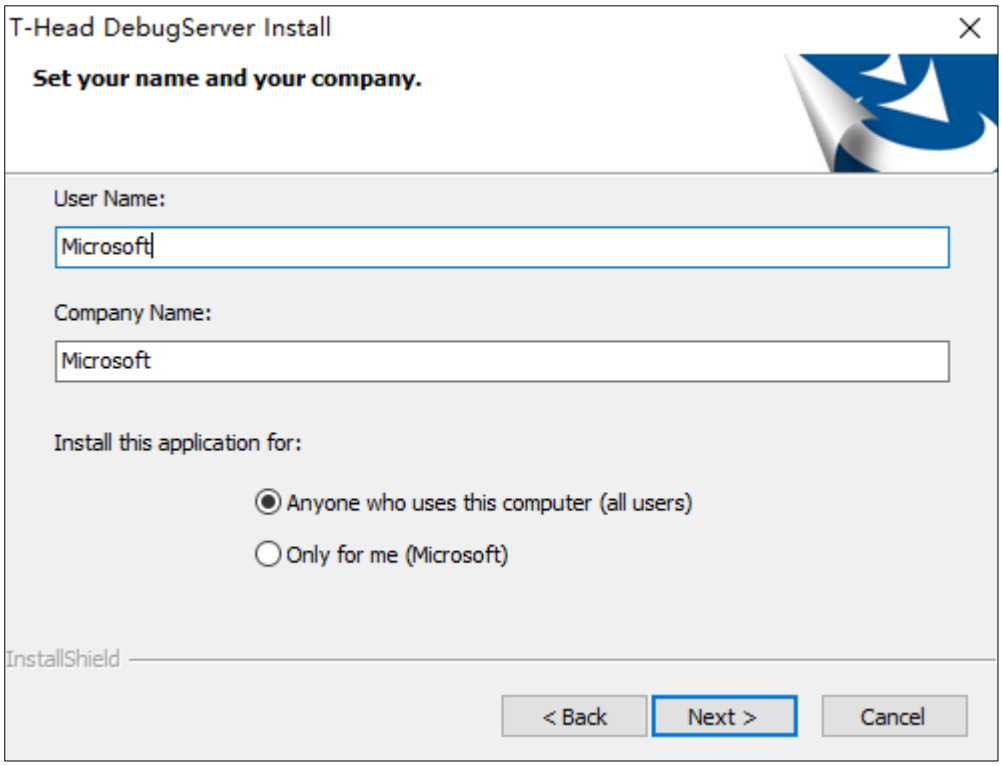


Figure 11-3 Install T-HEAD DebugServer — Step 3

4. Select a directory to install the software.

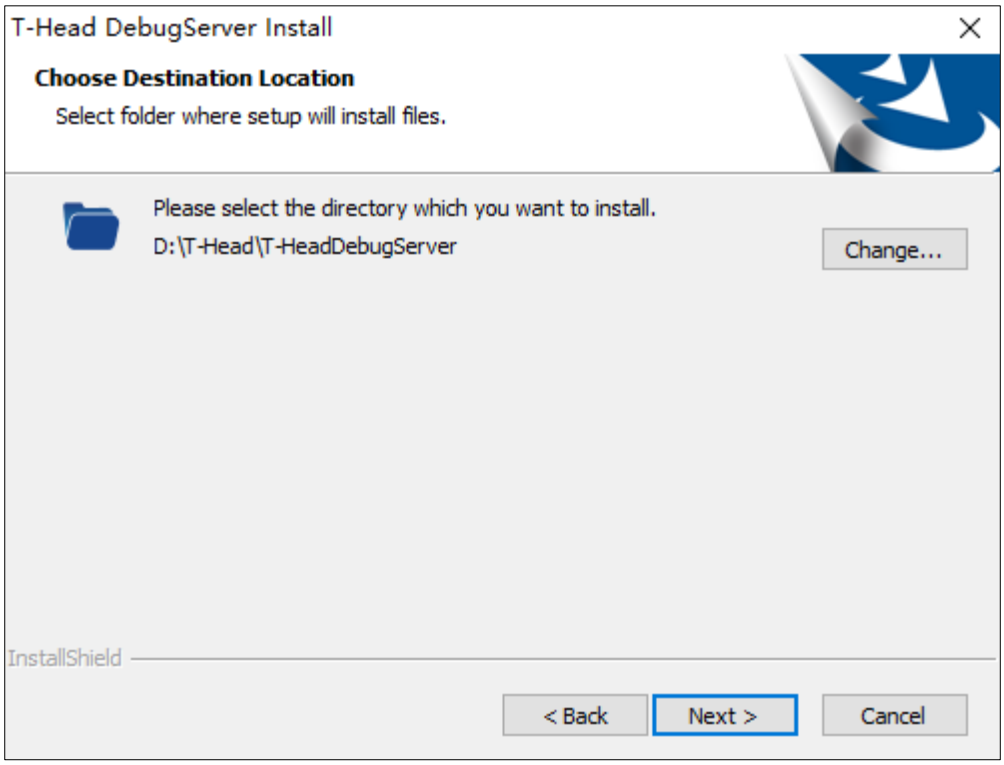


Figure 11-4 Install T-HEAD DebugServer — Step 4

5. Select the components you want to install. Main App is T-HEAD DebugServer, ICE Driver is

the ICE device driver, and Tutorial is the user manual. We recommend that you select all and click Next.

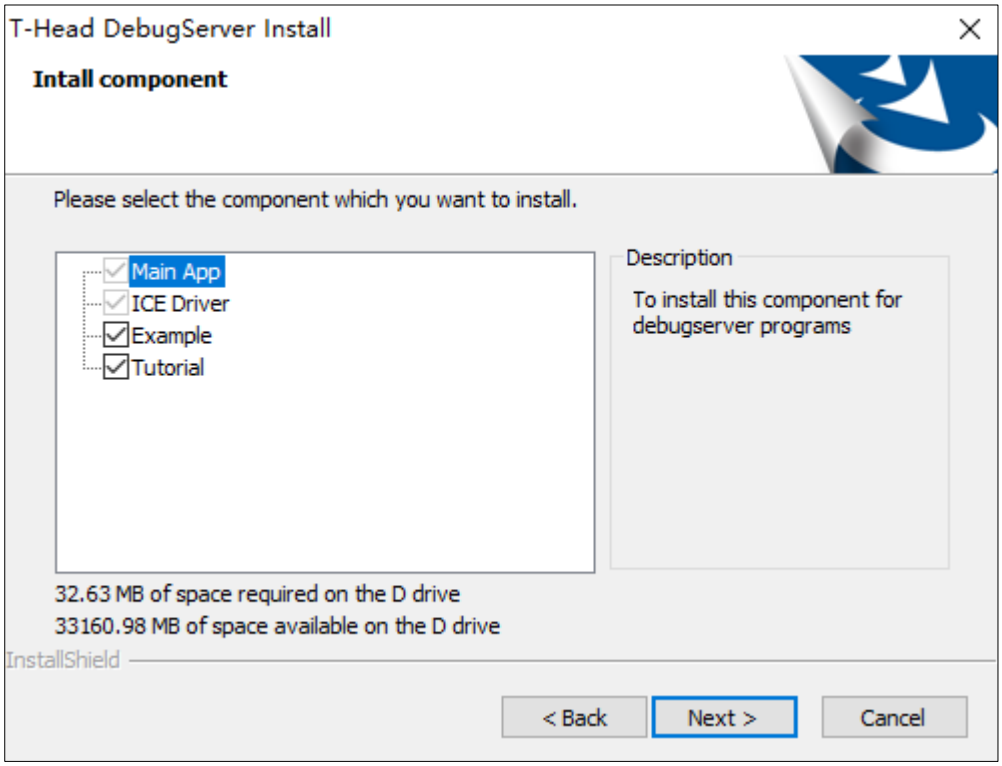


Figure 11-5 Install T-HEAD DebugServer — Step 5

6. Confirm the user information and installation directory on the page that appears and click Next to start the installation.

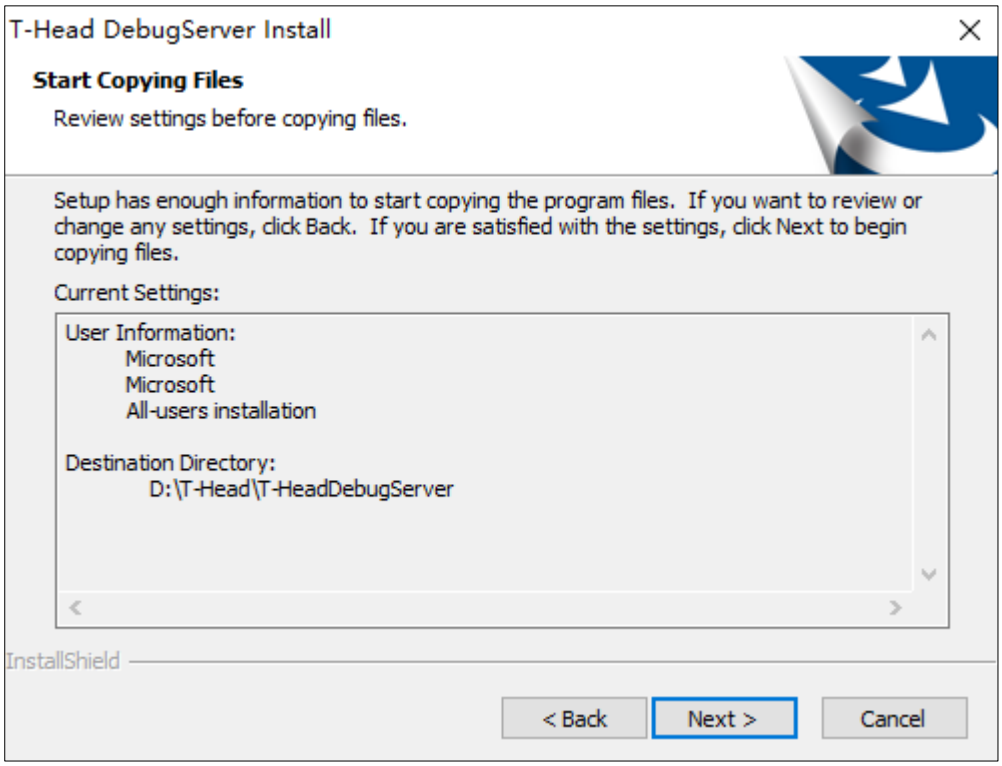


Figure 11-6 Install T-HEAD DebugServer — Step 6

7. The system starts installing the selected components.

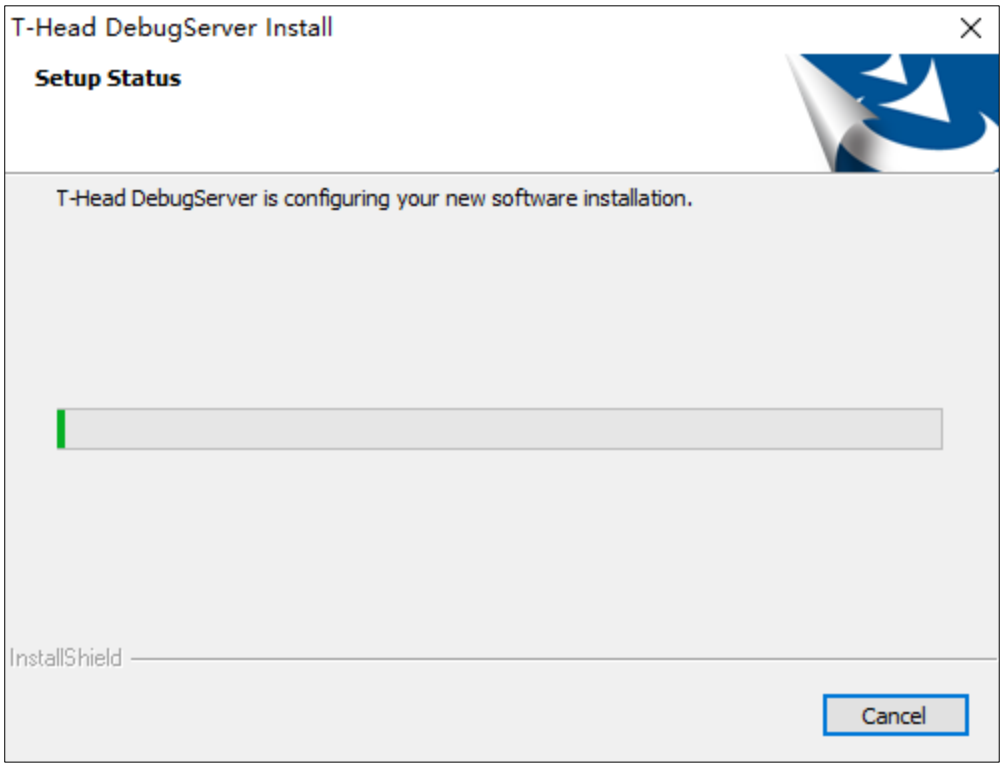


Figure 11-7 Install T-HEAD DebugServer — Step 7

7. If an ICE driver is already installed on your PC, a message is displayed to ask whether you want to update the driver. We recommend that you select Yes, so that your ICE driver will be updated to the latest.

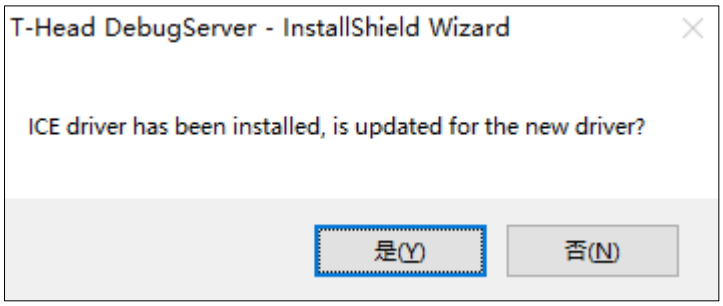


Figure 11-8 Dialog box for specifying whether to update the driver

9. Wait until the installation of T-HEAD DebugServer is complete and then click Finish.

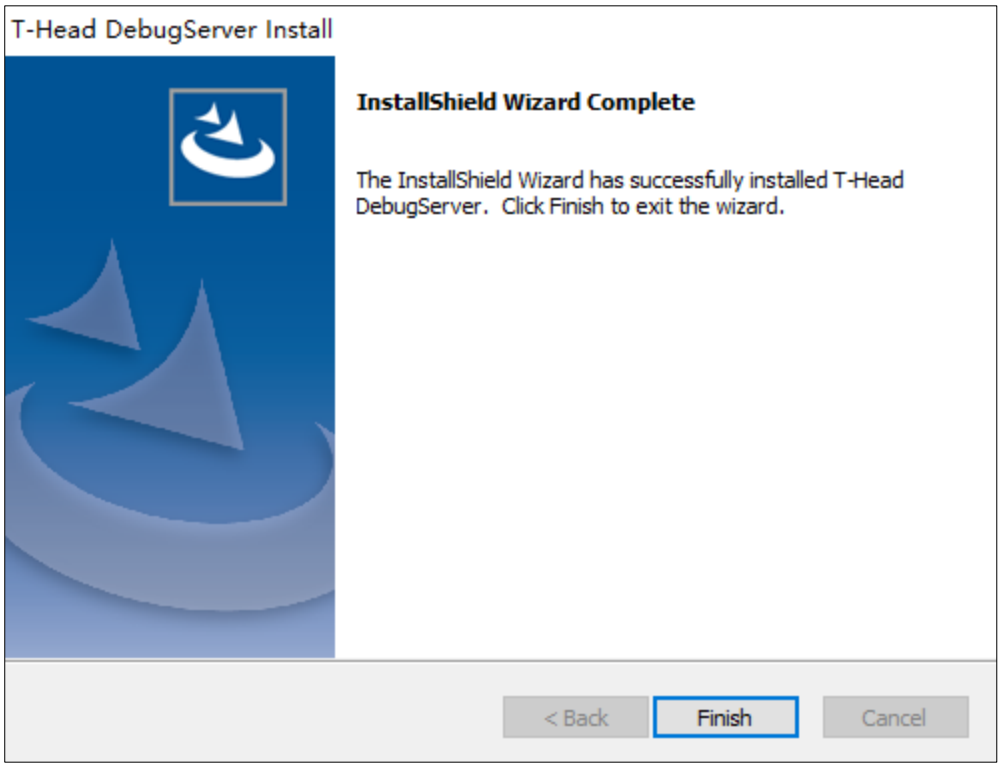
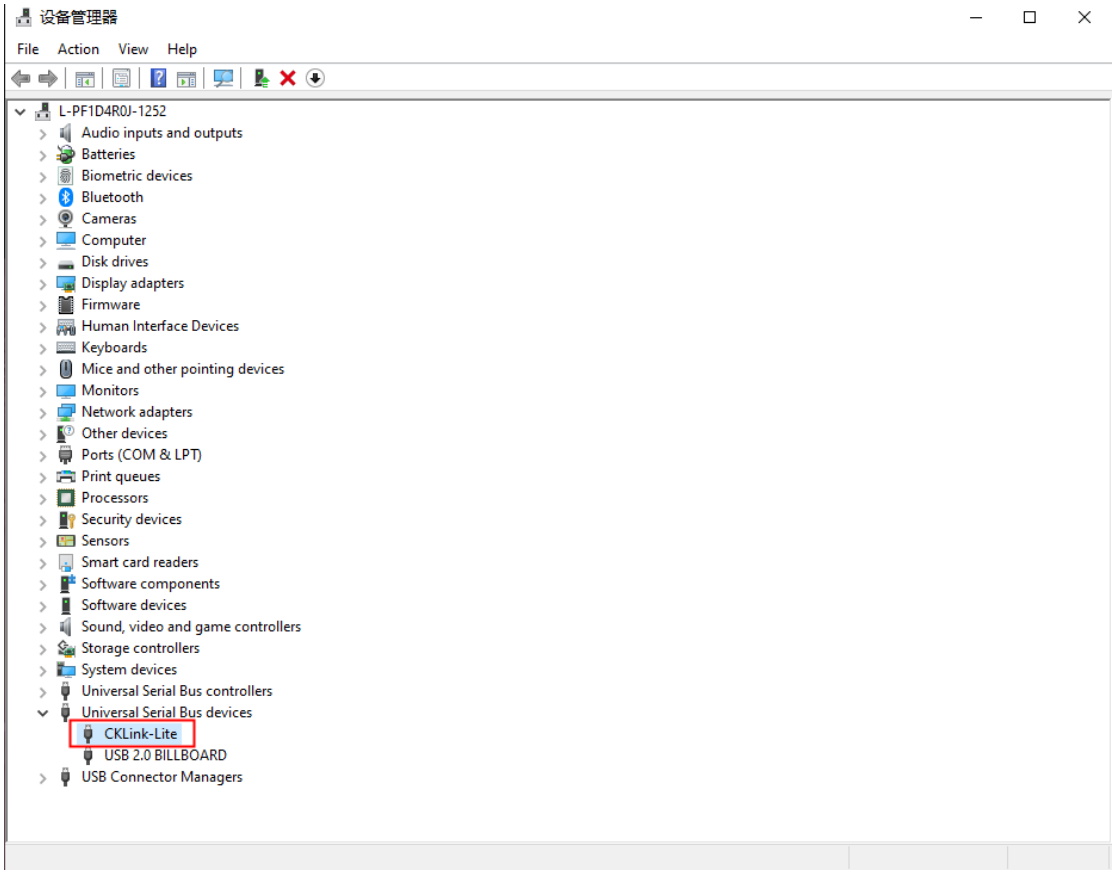


Figure 11-9 Install T-HEAD DebugServer — Step 9

2.1.3.Install the ICE driver

1. If you selected ICE Driver in Section 2.1.2 "Install T-HEAD DebugServer," the driver file will be copied to the system directory during installation of T-HEAD DebugServer.

2. Reconnect your ICE to the PC. You can view that the device name similar to CKLink-Lite or CKLink Pro shown in the following image in the Device Manager window on your PC.



2.2. Operating environment

T-HEAD DebugServer can run in Windows 2000 and later and require that the input and output devices be equipped with USB ports. T-HEAD DebugServer can be used with all versions of T-Head GDB.

2.3. How to use T-HEAD DebugServer Console Edition

2.3.1.Operating parameters

1. T-HEAD DebugServer Console Edition allows you to set T-HEAD DebugServer by running operating parameters of T-HEAD DebugServer. The following table lists common input parameters.

Table 11-2 Operating parameters of T-HEAD DebugServer

User interface	Description	Default value

【-setclk xxx】	Sets the JTAG clock frequency for the ICE. The default unit is MHz. The data measured in kHz is also supported. For example, you can run "-setclk 12000k" to set the frequency to 12 MHz. Each type of ICE has a fixed upper frequency limit: CKLink Pro 24 MHz CKLink-V1 24 MHz CKLink-Lite 2.5 MHz	The default value is 12. That is, the frequency of the ICE is 12 MHz by default.
【-port XXX】	Sets a socket port.	1025
【-prereset】	Specifies to initiate a reset operation to the target board by using the ICE after T-HEAD DebugServer is connected to the ICE.	By default, this operation is not performed.
【-noddc】	Specifies not to use the DDC.	By default, the DDC is used.
【-nocacheflush】	Specifies that T-HEAD DebugServer does not flush cache during single-stepping or while exiting the debug mode.	By default, cache is flushed.
【-setcdi 2/5】	Sets the operating mode of the ICE before ICE is connected to the target board. The value 2/5 following the parameter indicates 2 lines or 5 lines.	By default, this operation is not performed.
【-mtcrdelay/delay】	Sets the time that the ICE waits for after data is written to the CPU control register.	The default value is 1 ms.
【-targetinit filepath】	Specifies the initialization script to execute for the target board after T-HEAD DebugServer is connected to the ICE. The script is of the GPIO or JTAG type. The startup process of the server continues after the script is executed.	None.
【-scr filename】	Specifies a GPIO or JTAG script. After this parameter is specified and the script is executed, T-HEAD DebugServer automatically exists. For more information about GPIO and JTAG scripts, see subsequent sections. Note that only CKLink Pro supports GPIO scripts.	None.

【-configpath/-configfilepath filepath】	Specifies the path to obtain the firmware of the ICE during configuration of the ICE.	The default value is the directory where the executable program of T-HEAD DebugServer is located.
【-tdescfile filepath】	Specifies an .xml file for T-Head GDB to describe the register of the target board.	By default, a default tdesc-xml file is specified based on the CPUID.
【-{no-}trst】	Specifies whether to run TReset when running the reset command.	By default, this operation is performed.
【-nrstdelay】	Sets an NReset delay to ensure that the ICE can generate stable hardware reset (NReset) signals. Unit: 10 us	The default value is 1 ms.
【-trstdelay】	Sets a TReset delay to ensure that the ICE can generate stable reset signals to reset the HAD state machine. Unit: 10 us	The default value is 1.1 ms.
【-ndmrstdelay】	Sets the length of time required to generate an RISC-V DM DMCONTROL.ndmreset signal.	The default value is 0.
【-hartrstdelay】	Sets the length of time required to generate an RISC-V DM DMCONTROL.hartreset signal.	The default value is 0.
【-rstwait】	Sets a delay to ensure that the reset process of the target board lasts to the end after the target board receives a reset signal.	The default value is 50 ms.

【-no-multicore-threads】	Sets to enable a debug port for each core when connecting to C860 for multi-core debugging. T-Head GDB can connect to a port to debug the core that corresponds to the port. The default value is false, only one debug port is enabled on T-HEAD DebugServer, and multiple cores are packaged to send multiple pieces of thread information to T-Head GDB. When T-Head GDB debugs a thread, T-Head GDB is debugging the core that corresponds to the thread.	The default value is false.
【-local-semi/-ls】	Specifies that T-HEAD DebugServer implements semihosting requested by a program.	By default, semihosting is implemented by T-Head GDB. You can set the [-local-semi/-ls] option to implement semihosting in T-HEAD DebugServer. Note: When you add the -local-semi/-ls option in Windows, isatty and system operations are not supported.
【-dcomm=ldcc】	Enables JTAG input and output channels. Hardware support is required.	The feature is disabled by default.

【-disable-cmdline】	Disables the command line feature.	The feature is enabled by default.
【-set-isa_version v1/v2/v3/v4/v5】	Sets the HAD version of the ICE.	By default, T-HEAD DebugServer automatically sets the value.
【-set-hacr_width 8/16】	Specifies the width of the HACR used to connect to the target board.	By default, T-HEAD DebugServer automatically detects the width.
【-no-cpuid-check】	Specifies not to read CPUID information during connection to the target board.	By default, CPUID information is read and checked.
【-cacheflush-delay xxx】	Sets the delay of cache flushing in T-HEAD DebugServer to ensure that cache flushing properly ends.	The default value is 100 ms.
【--debug usb/connect/target/remote/djp/sys/flash/all】	usb: records the protocol packets used for interaction between T-HEAD DebugServer and the ICE. connect: describes the detailed process of connecting to a development board. target: records the information about function calls at the abstraction layer of the target board. remote: records the information about interactions that are based on the remote protocol. djp: records the information about interactions that are based on the DJP protocol. sys: records the information about the main loop of the T-HEAD DebugServer program. flash: records the information about flash programming and flash breakpoints. all: prints all the preceding log information.	By default, no log information is generated.

【-arch thead /ricsv】	Selects a debug architecture to connect to. The value thead indicates the T-Head HAD debug architecture and the value riscv indicates the RISC-V DM architecture.	The default value is thead.
【-cmd-script file】	Executes the command line script of T-HEAD DebugServer. That is, you can write commands at the cmdline into a file and execute the file after T-HEAD DebugServer is connected to the target board.	None.
【-list-ice】	Lists ICEs connected to the PC.	None.
【-select-ice xxx】	Specifies an ICE to connect based on the ICEs listed in the -list-ice option.	None.
【-list-vendor】	Displays supported ICE vendors, such as CKLink.	
【-select-vendor】	Selects the vendor of the specified ICE.	By default, CKLink is selected.
【-skip-enter】	Specifies that you no longer need to press the Enter key when you want to exit the T-HEAD DebugServer program.	By default, you need to press the Enter key.
【-set-logfile FILENAME】	Reports the error log generated for T-HEAD DebugServer in runtime to the file specified by FILENAME.	None.
【-idle COUNT】	Sets the number of cycles for which the idle state of the JTAG state machine persists.	For 800 series CPU, the value is 0. For 900 series CPUs, this value is automatically set based on the configurations in the DTM.
【-sampling cpu_n port freq】	Enables the socket port that is used for sampling. The following parameters are provided: Cpu_n: performs sampling on cpu_n. Port: the number of the port. The port number is used to connect to the CPF. Freq: the sampling frequency.	None.

【-sampling-type pcfifo-link/pcfifo-host】	Sets the sampling method of the PC. Valid values: pcfifo-link or pcfifo-host. pcfifo-link: performs sampling on CKLink. pcfifo-host: performs sampling by using the host T-HEAD DebugServer.	The default value is pcfifo-link.
【-flash-algorithm FILENAME】	Specifies a flash algorithm file, which is used for flash programming and setting flash breakpoints.	None.
【-flash-timeout TIME】	Specifies the timeout period for a function to run to a breakpoint specified by the __bkpt_label option after the flash algorithm file is used to perform one flash operation.	The default value is 60 seconds.
【-disable-flashbp】	Disables the flash breakpoint feature.	By default, the feature is enabled.
【-disable-simbp】	Disables instruction emulation in flash breakpoints.	By default, the feature is enabled.
【-abscmd-busy-delay TIME】	Specifies the waiting time after ABSTRACTCS.busy becomes 1 during the execution of the abstract command.	None.
【-enter-debug-time TIME】	Sets the timeout period for making multiple CPUs synchronously enter the debug mode. The unit of the TIME parameter is seconds. The maximum value is 20.	0
【-v/-version】	Views the version number of a program.	
【-h/--h/--help】	Views help information.	

2. Sample parameter settings

This part illustrates common input examples. T-HEAD DebugServer is an application. The following input examples are for your reference:

- T-HEAD DebugServer

  This command is equivalent to T-HEAD DebugServer –setclk 12M –port 1025. The default option is used.

- T-HEAD DebugServer –stclk 13000k –port 1111

  Sets the number of the socket port to 1111 and the frequency of the hardware ICE to 13 MHz.

- T-HEAD DebugServer –scr E:\jtag.ini

  Executes the specified T-HEAD DebugServer script.

2.3.2.Script configuration feature of T-HEAD DebugServer

T-HEAD DebugServer Console Edition allows you to run scripts. Scripts are divided into JTAG scripts and GPIO scripts. A JTAG script can perform direct operations on a JTAG scan chain register. You can customize a combination of JTAG operations to access a specified HAD register and manage other hardware. A GPIO script can control the outputs of JTAG pin levels of an ICE and obtain JTAG pin levels of an ICE. You can generate custom waveforms by using the script. You can generate waveforms required for the pins or read the level state of JTAG pins.

The following sections describe the usages of the two types of scripts in detail.

2.3.2.1.JTAG script

T-HEAD DebugServer allows you to read and write the HAD register by executing A JTAG script.

The following content describes the JTAG script file.

1． No special requirements are set for the file extension. The file name can be a relative path.

2． The syntax must meet the following rules:

(1) Multiple JTAG operations can be described in the script. Each operation unit must use [JTAGx] as a keyword, where x represents a number. Note that [JTAGx] must be capitalized and the numbers must be consecutive. If inconsecutive numbers are used, the script will not be executed.

(2) IR and DR must be capitalized in the following format:

IR=[ir length], byte0, byte1 … ——IR write

DR=W,[dr length] byte0, byte1 … —— DR write

DR=R, [dr length] ——DR read

(3) The IR length and the DR length must be a multiple of 8 bits in versions earlier than V5.14.2. In V5.14.2 and later, non-8-bit aligned values are supported. Here, the IR length and the DR length can be n-byte aligned.

(4) The content in bytes must be in hexadecimal, with or without the prefix 0x.

(5) The bytes are written into JTAG DR in order and read from the DR in order.

3. Script examples

```
[JTAG0]
IR=8, 0x8
DR=R, 32

[JTAG1]
IR=8, 0x8E
DR=R, 16

[JTAG2]
IR=8, 0x8E
DR=W, 16, 0x12, 0x34
```

2.3.2.2. GPIO script

T-HEAD DebugServer allows you to use a GPIO script to control the outputs of JTAG pin levels and generate waveforms required for the pins. In addition, you can use a GPIO script to read the level state of JTAG pins.

The following content describes the GPIO script file:

1. The T-Head GPIO script is represented by the [THEAD_ICE_GPIO] field. Only a script file starting with this field is deemed as a T-Head GPIO script.

2. No special requirements are set for the file extension. The file name can be a relative path.

3. The syntax must meet the following rules:

   (1) The script takes behavioral parsing and execution as a unit. Each line is parsed and executed once. One line may contain multiple statements separated with commas (,). The last statement must not be followed by a comma. Otherwise, a syntax error occurs.

(2) Multiple statements in the same line can be used to perform operations on the same bit of the same register. A latter statement overwrites a former statement. For example, if TDI=1, TMS=1, NRST=0, and TDI=0 exist in the same line, the final parsing and execution result controls the output level of TDI to be 0.

(3) In the assignment statement for a single pin, such as TMS=1, the value must be 0 or 1. Otherwise, a syntax error occurs. Values starting with 0x are deemed as hexadecimal. Values not starting with 0x are deemed as decimal.

(4) During the execution of the script, the initial level of each JTAG pin is 0. In this operation, no assignment is performed on the pin. Therefore, the level of the pin remains the output level in the previous operation.

(5) The default value of the GPIO_OE register is 0x3f. That is, each JTAG pin corresponds to the output mode. You can assign a value to the GPIO_OE register to change the input/output mode of the pin. After the GPIO_OE register is assigned a value, the value takes effect until the next time a value is assigned to the GPIO_OE register.

(6) The script supports repeat loops, whose syntax is REPT=x.....ENDR. Nested loops are supported. REPT and ENDR must be used in pairs; otherwise, a syntax error occurs. The equal sign (=) that follows REPT cannot be omitted. The number of loops is indicated by x and cannot be less than 0; otherwise, a syntax error occurs. The REPT statement and the ENDR statement must be on separate lines; otherwise, a syntax error occurs.

(7) The PRINT statement is used to read the value of the GPIO_IN register and must comply with the syntax PRINT=GPIO_IN. Otherwise, a syntax error occurs.

(8) The content of the script must start with START and end with END. Otherwise, a syntax error occurs.

(9) The script is not case-sensitive.

(10) The statement following a number sign (#) is a comment.

4. Multiple types of statements are supported.

The THEAD_ICE_GPIO script supports the following types of statements:

- Control statements used to control the input/output directions of each pin. For example, GPIO_OE=0x1f is supported.
- Assignment statements used to control the output of a single pin. For example, the statement TMS=1 is used to assign the value 1 to the bit that corresponds to the TMS pin in the GPIO_OUT register.
- One-time assignment statements for all pins. For example, the statement GPIO_OUT=0x1d is used to assign the value to the GPIO_OUT register.
- PRINT=GPIO_IN statement, which is used to read and print the value of the GPIO_IN register.
- REPT and ENDR statements, that is, repeat loops. The statement REPT=x, where x is the number of loops, is used to repeat the statement between REPT and ENDR x times.

Note: In the script, the output assignment statement such as TMS=1 and the input print statement PRINT=GPIO_IN only read data from and write data to the GPIO_OUT and GPIO_IN registers in the ICE. JTAG pins set bits only during operations on the GPIO_OE register. For example, when GPIO_OE[3] for a TMS pin is 1, which indicates the output mode, the value of GPIO_OUT[3] is reflected on the TMS pin. Similarly, when GPIO_OE[3] is 0, which indicates the input mode, the value of GPIO_IN[3] that is read by using the PRINT=GPIO_IN statement is the real level of the TMS pin. Otherwise, the value is invalid.

5. Script examples

```
[THEAD_ICE_GPIO]
START
GPIO_OE = 0x1f
TMS=1, TDO=0, TDI=1
TDI=1,TMS=1,NRST=0, TDI =0
PRINT= GPIO_IN        #print the value of GPIO_DATA
GPIO_OE = 0x13
```

```
REPT = 10
TMS =0
TMS = 1
ENDR
TRST=1
#GPIO_CTRL=0x1d
END
```

2.3.2.3. Script execution

To execute a script in T-HEAD DebugServer Console Edition, use the [-scr filename] parameter among the operating parameters of T-HEAD DebugServer (see Section 2.3.1).

To execute a script in T-HEAD DebugServer UI Edition, click the  icon in the menu bar toolbar (see Section 2.4.2).

Note: To execute a script in T-HEAD DebugServer, T-HEAD DebugServer must be connected to an ICE and the development board. Therefore, before you execute a script in T-HEAD DebugServer UI Edition, run T-HEAD DebugServer to check whether the hardware connections are correct and then stop T-HEAD DebugServer and click the Script icon to select the script to execute.

2.3.3. How it works

1. Steps

- Click "Start->All Programs->T-Head->T-Head DebugServer->bin->DebugServerConsole.exe" to open T-HEAD DebugServer by using default operating parameters. Figure 2-12 shows the user interface.
- If you need to modify an operating parameter, you can press "CTRL+C" to cancel the connection. The current directory is the installation directory. You need only to enter the command for running the parameters that you need to run, such as "DebugServerConsole.exe -port 1028".
- Run the T-Head GDB application to perform debugging:

1. Use the T-Head toolchain to generate the T-Head elf program named a.out.

2. Start T-Head GDB, such as csky-*-gdb a.out.

3. As prompted by T-HEAD DebugServer, type the connection command in the command line of T-Head GDB. For example, type target remote 172.16.28.158:1025.

4. After T-Head GDB is connected to T-HEAD DebugServer, perform common operations on T-Head GDB. For example, you can perform the following operations:

① load // Downloads the program to the development board.

② break main // Sets a breakpoint in the main function.

③ continue // Runs the program.

④ info registers r0 // Views Register r0.

⑤ print var_a // Views the program variables indicated by var_a.

5. How to use T-Head GDB is similar to how to use GNU GDB.

- If you use CDK or CDS, see the user documentation for the development environment.

2. Operating UI

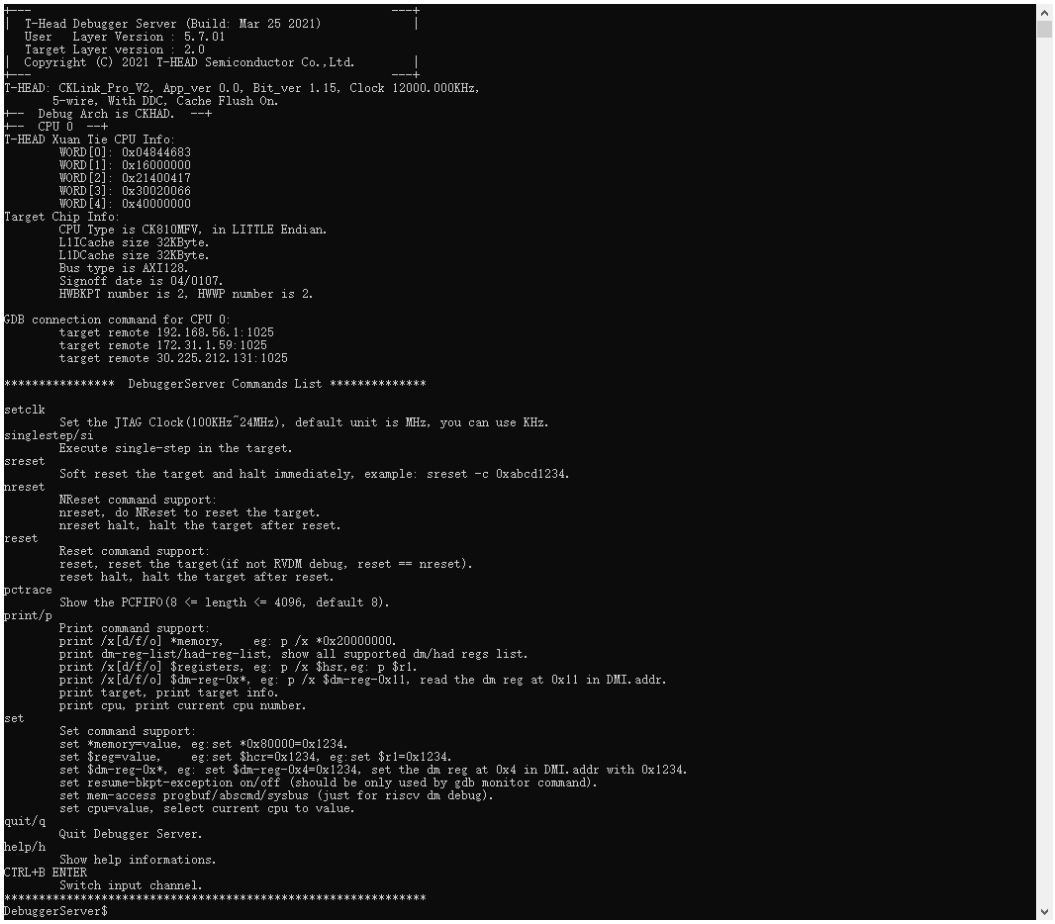


Figure 11-4 Operating UI of T-HEAD DebugServer Console Edition

3. Note

(1) During the running of T-HEAD DebugServer, if a message that indicates "ICE Upgrade" appears, click Yes to upgrade the firmware. After the upgrade is complete, you need to unplug and re-plug the ICE and connect T-HEAD DebugServer to the ICE.

(2) When the target board is multi-core, T-HEAD DebugServer enables multiple consecutive ports based on the serial number of the CPU in the structure, to connect to T-Head GDB. The first port number is specified by users and is 1025 by default.

2.4. How to use T-HEAD DebugServer UI Edition

2.4.1.Main UI

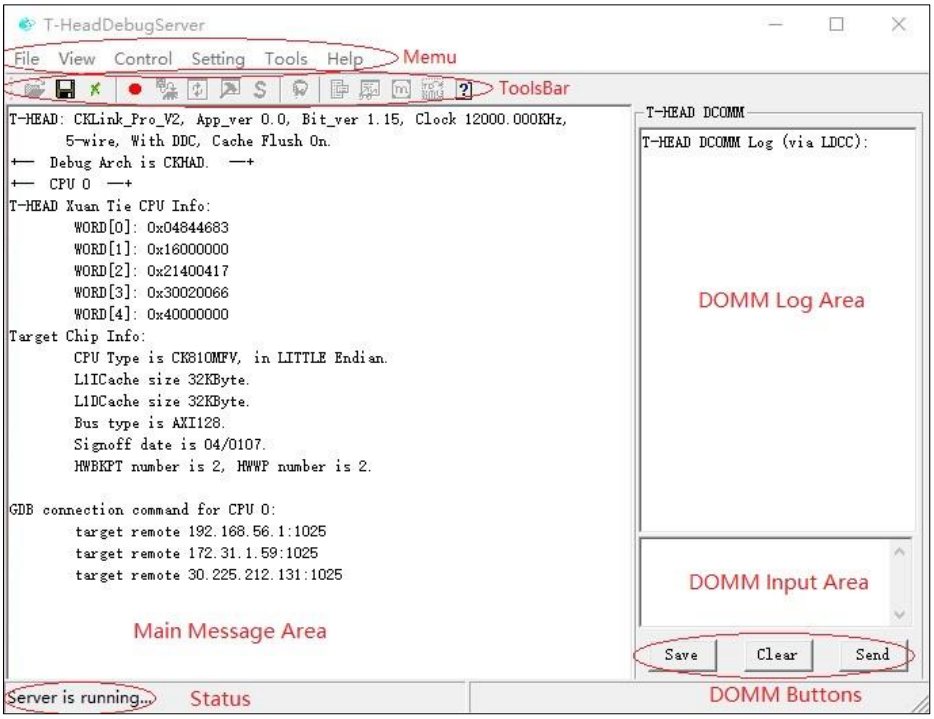


Figure 11-5 Main UI of T-HEAD DebugServer UI Edition

2.4.2.Menu bars and toolbars

Table 11-3 File menu bar

Menu	Description	Toolbar icon
Open Config File	Opens the configuration files of the target board and socket ports.	
Save Config File	Saves the configuration files of the target board and socket ports.	
Save Log file	Saves log information to the log file.	/
Exit	Closes a program.	/

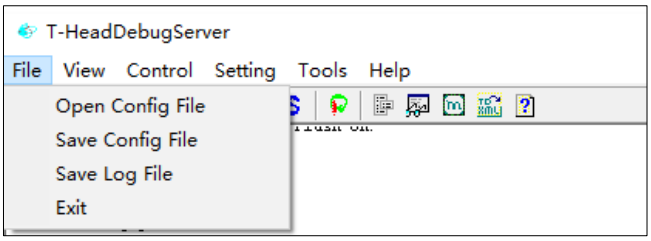


Table 11-4 View menu bar

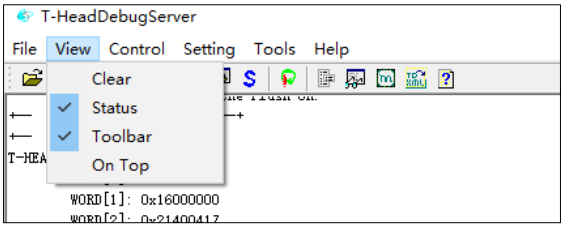
Menu	Description	Toolbar icon
Clear	Clears the message area.	
Status	Shows or hides the status bar.	/
Toolbar	Shows or hides the toolbar.	/
On Top	Pins the UI on top of the screen.	/
		

Table 11-5 Control menu bar

Menu	Description	Toolbar icon
Check Target	Checks the connected target board.	
Reset Target	Resets the target board. The reset feature depends on the method of connecting to nRST signals on JTAG ports of the target board.	
RunDebuggerServer	Runs or stops T-HEAD DebugServer.	 (Run)  (Stop)

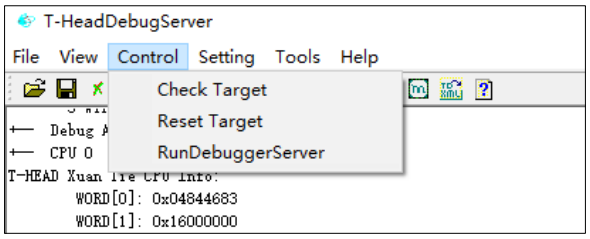


Table 11-6 Setting menu bar

Menu	Description	Toolbar icon
Target Setting	Sets the configuration parameters of an ICE, including the operating frequency of the ICE and whether to use DDC. For more information, see Figure 2-14 and its description.	
Socket Setting	Sets the communication port. The default port number is 1025. For more information, see Figure 2-15 and its description.	
Cpu Select	Selects a CPU. To view the registers of the HAD and CPU on the UI of T-HEAD DebugServer during debugging of a multi-core development board, you need to select a CPU first.	
TDFile Setting	Specifies an .xml file to describe the register of the target board to T-Head GDB.	

Multicore Threads	If this menu is selected, when T-HEAD DebugServer connects to C860 for multi-core debugging, only one debug port is enabled on T-HEAD DebugServer and multiple cores are encapsulated into multiple pieces of thread information and sent to T-Head GDB. When T-Head GDB debugs a thread, T-Head GDB is debugging the core that corresponds to the thread. If this menu is not selected, when T-HEAD DebugServer connects to C860 for multi-core debugging, one debug port is enabled for each core. T-Head GDB can connect to a port to debug the core that corresponds to the port.	No shortcut keys
Flash Setting	Configures the algorithm file used for flash programming, flash breakpoints, and simulated breakpoints.	No shortcut keys

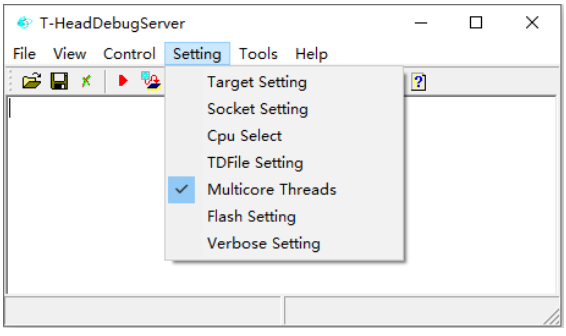
Verbose Setting	Sets the outputs of log information for T-HEAD DebugServer: usb: records the protocol packets used for interaction between T-HEAD DebugServer and the ICE. connect: describes the detailed process of connecting to a development board. target: records the information about function calls at the abstraction layer of the target board. remote: records the information about interactions that are based on the remote protocol. djp: records the information about interactions that are based on the DJP protocol. sys: records the information about the main loop of the T-HEAD DebugServer program. flash: records the information about flash programming and flash breakpoints. all: prints all the preceding log information. After you select the desired log option, click Select and reconnect T-HEAD DebugServer to the development board.	No shortcut keys
		

Table 11-7 Tools menu bar

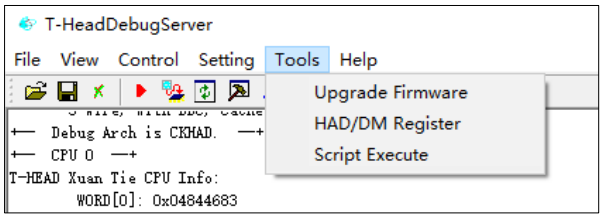
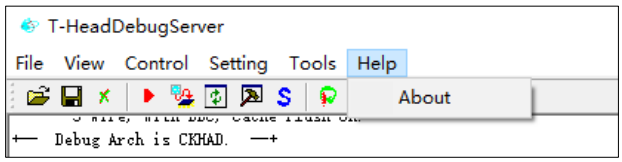
Menu	Description	Toolbar icon
Upgrade Firmware	Upgrades the firmware of the ICE. Note that the upgrade file must be consistent with the box type of the ICE. The following correspondences are for your reference: CKLINK_LITE_V2: cklink_lite.hex CKLINK_PRO_V1: cklink_v1.iic CKLINK_PRO_V2: cklink_pro.iic	
HAD/DM Register	Operates the registers of the HAD module or the Debug Module register in the RISC-V debug architecture. The detailed operations depend on the actual Debug Arch.	
Execute Script	Executes a T-HEAD DebugServer script.	
		

Table 11-8 Help menu bar

Menu	Description	Toolbar icon
About	Displays the version information of T-HEAD DebugServer.	



2.4.3.Startup configuration file

By default, the startup file is the default.ini file in the directory where T-HEAD DebugServer is located. The following content describes the file content:

[TARGET] tags:

- JTAGTYPE=USBICE; // Configures the type of the in-circuit debugger. Only USBICE is supported.
- ICECLK=1200; // Sets the operating frequency of the in-circuit debugger to 12 MHz.
- DDC=TRUE; // Enables the DDC for hardware of T-Head.
- CACHEFLAG=TRUE; // Specifies whether to flush cache during single-stepping and while exiting the debug mode.
- MTCRDELAY=10; // Specifies the delay for executing an MTCR instruction.
- TARGETINIFILE=; // Specifies the initialization script to execute for the target board after T-HEAD DebugServer is connected to the ICE. The script is of the GPIO or JTAG type. The startup process of the server continues after the script is executed.
- PRESETCDI=2/5; // Sets the operating mode of the target board after T-HEAD DebugServer is connected to the ICE. The value 2/5 indicates 2 lines or 5 lines.
- PRERESET=FALSE; // Enables to send NReset signals to the target board after T-HEAD DebugServer is connected to the ICE.
- TDESCXMLFILE=; // Specifies an .xml file to describe the register of the target board.
- NRESETDELAY=100; // Sets the time that T-HEAD DebugServer takes to send an NReset signal.
- TRESETDELAY=110; // Sets the time that T-HEAD DebugServer takes to send a

TReset signal.

- RESETWAIT=50; // Sets the delay for waiting for a reset operation to complete.
- MULTICORETHREADS=TRUE; // Sets the debugging mode of C860MP.
- DCOMMTYPE=LDCC; // Enables checks on output information of DCOMM. The check method is LDCC.
- LOCALSEMIHOST=FALSE; // Specifies that T-HEAD DebugServer implements semihosting requested by a program.
- DEBUGARCH=AUTO; // Sets the architecture of the connected debug module. The value CKHAD indicates T-Head HAD, the value RISCV indicates RISC-V DM, and the value AUTO indicates automatic detection.
- DMSPEEDUP=TRUE; // Specifies whether to read data from and write data to general purpose registers (GPRs). Memory read and write operations are encapsulated by the ICE. When the value is FALSE, the operations are performed by the upper-layer of T-HEAD DebugServer through JTAG ports.

- CACHEFLUSHDELAY=10; // Specifies the delay of cache flushing, in milliseconds.
- TRST=TRUE; // Enables TReset.
- IDLEDELAY= ; // Sets the idle delay to a value ranging from 0 to 7.
- SAMPLINGCPF= ; // Enables CPF sampling. Valid values: TRUE or FALSE
- SAMPLINGCPU= ; // Specifies the sampled CPU.
- SAMPLINGPORT= ; // Specifies the port number used for CPF sampling.
- SAMPLINGFREQ= ; // Sets the sampling frequency.
- SAMPLINGTYPE= ; // Sets the sampling type. The value PCFIFO-HOST indicates the host and the value PCFIFO-LINK indicates CKLink.
- NDMRESETDELAY= ; // Sets the delay of DMCONTROL.ndmreset.
- HARTRESETDELAY= ; // Sets the delay of DMCONTROL.hartreset.
- FLASHALGORITHMPATH=; // Specifies a flash algorithm file.
- FLASHTIMEOUT=60; // Specifies the timeout period for a function to run to a breakpoint specified by the __bkpt_label option after the flash algorithm file is used to perform one flash operation.

- DISABLEFLASHBP=FALSE; // Disables flash breakpoints.
- DISABLESIMBP=FALSE; // Disables the instruction simulation feature in flash breakpoints.
- ENTERDEBUGTIME=; // Sets the timeout period for initiating synchronization to make the CPU enter the debug mode.

  [SOCKETSERVER]

  - SOCKETPORT=1025; // Sets a port on T-HEAD DebugServer to enable the socket service.

2.4.4.Common features

1. Open and save the configuration file

T-HEAD DebugServer allows you to configure T-HEAD DebugServer based on an existing configuration file by choosing File->Open Config File and save the current configurations as a configuration file by choosing File->Save Config File. The configurations include the CPU frequency and whether to use DDC.

2. Specify target settings

In addition to using an existing configuration file, you can manually specify configurations. Choose Setting->Target Setting to open the Target Setting dialog box.

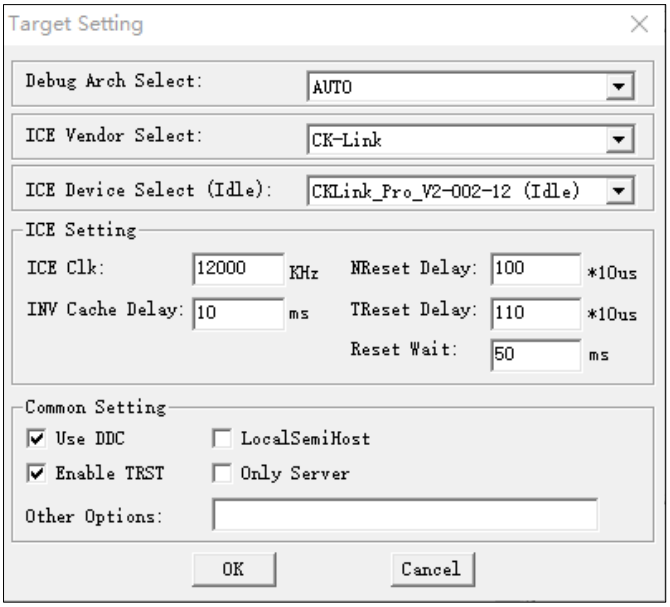


Figure 11-6 Target Setting dialog box

The following content describes the parameters:

- Debug Arch Select: selects debug architecture. Valid values: T-Head HAD, RISCV DM, and AUTO.
- ICE Vendor Select: selects an ICE vendor. The default value is CK-Link.
- ICE Device Select (Free): specifies an ICE connection.
- ICE Setting: sets the information about an ICE that you want to use as a connection target, including the ICE frequency and MTCR delay.
- NReset Delay: sets an NReset delay to ensure that the ICE can generate stable hardware reset (NReset) signals. The unit is 10 us. The default value is 1 ms.
- TReset Delay: sets a TReset delay to ensure that the ICE can generate stable reset signals to reset the HAD state machine. The unit is 10 us. The default value is 1.1 ms.
- Reset Wait: sets a delay to ensure that the reset process of the target board lasts to the end after the target board receives a reset signal. The default value is 50 ms.
- Use DDC: sets whether to use the DDC for download.
- Enable TRST: specifies whether to execute TRST when executing Reset Target.
- LocalSemiHost: specifies whether to enable T-HEAD DebugServer to implement semihosting requested by a program. By default, semihosting is implemented by T-Head GDB.

3. Set the socket port

Choose Target Setting->Socket Setting. The Socket Server Setting dialog box appears, as shown in the following figure. Enter the port number. The default value is 1025.

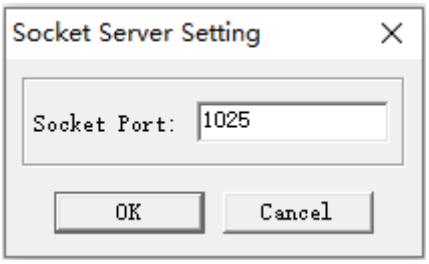


Figure 11-7 Socket Server Setting dialog box

4. Upgrade the firmware

Choose Tools->UpgradeFirmware. The Upgrade Firmware dialog box appears, as shown in

the following figure.

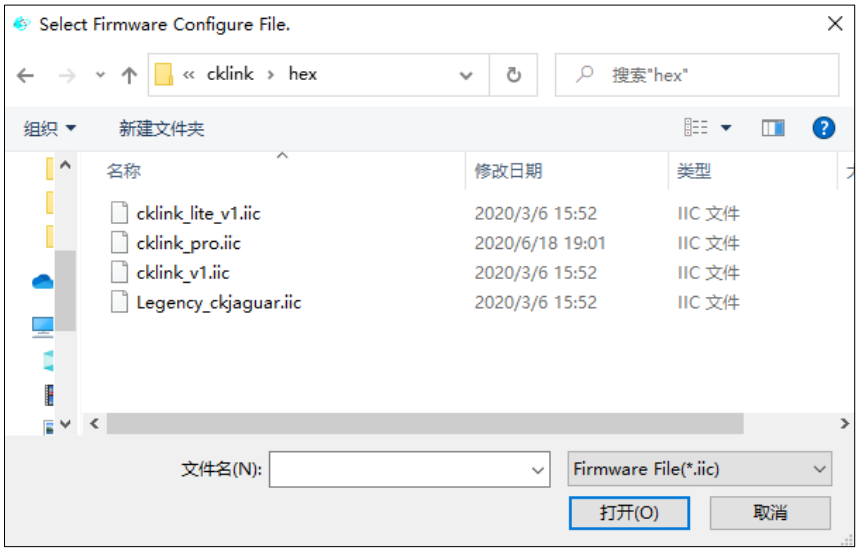


Figure 11-8 Upgrade Firmware dialog box

Select the upgrade file that corresponds to your ICE. The upgrade file must be consistent with the box type of the ICE. The following correspondences are for your reference:

CKLINK_LITE_V2: cklink_lite.hex

CKLINK_PRO_V1: cklink_v1.iic

CKLINK_PRO_V2: cklink_pro.iic

5. Set the HAD or DM register

When the current Debug Arch is T-Head HAD, perform the following steps:

Choose Tools->HAD/DM Register. The Had Register Operator dialog box appears. You can read data from and write data to the HAD register.

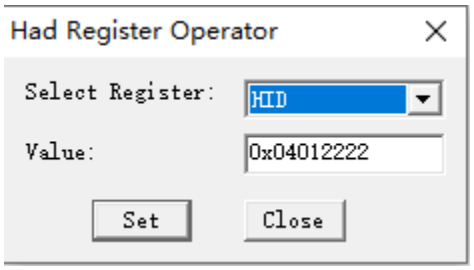


Figure 11-9 HAD Register Operator dialog box

Read operation:

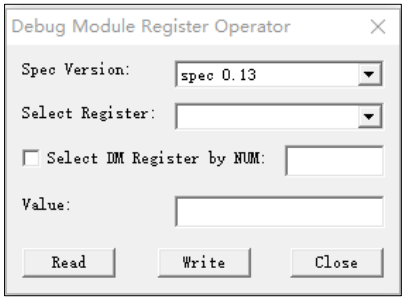
Select a HAD register to read from the Select Register drop-down box. The Value field automatically displays the value of this register.

Write operation:

Select the HAD register to write data to from the Select Register drop-down box. In the Value field, enter the desired value and then click Set.

When the current Debug Arch is RISC-V DM, perform the following steps:

Choose Tools->HAD/DM Register. The Debug Module Register Operator dialog box appears. You can read data from and write data to the DM register.



The Spec Version field is automatically set to the detected DM version.

In the Select Register field, select a DM register and click Read or Write as needed. If the DM registers you selected from the Select Register drop-down box does not contain the register that you want to manage, you can select "Select DM Register by NUM" and specify the number. Then, click Read or Write as needed.

2.4.5.How it works

1. Steps

- Choose "Start->All Programs->C-Sky->T-Head DebugServer->T-HeadDebugServer. T-HEAD DebugServer is opened with the default configurations, as shown in the following figure.
- If you need to modify the configurations of T-HEAD DebugServer, click RunDebuggerServer to stop T-HEAD DebugServer. Configure T-HEAD DebugServer based on the description in Section 2.4.2 "Menu bars and toolbars." Then, click RunDebuggerServer to run T-HEAD DebugServer again.

- Run the T-Head GDB application to perform debugging:

  1. Use the T-Head toolchain to generate the T-Head elf program named a.out.
  2. Start T-Head GDB, such as csky-*-gdb a.out.
  3. As prompted by T-HEAD DebugServer, type the connection command in the command line of T-Head GDB. For example, type target remote 172.16.28.158:1025.
  4. After T-Head GDB is connected to T-HEAD DebugServer, perform common operations on T-Head GDB. For example, you can perform the following operations:

     ① load // Downloads the program to the development board.

     ② break main // Sets a breakpoint in the main function.

     ③ continue // Runs the program.

     ④ info registers r0 // Views Register r0.

     ⑤ print var_a // Views the program variables indicated by var_a.

  5. How to use T-Head GDB is similar to how to use GNU GDB.

- If you use CDK or CDS, see the user documentation for the development environment.

2. Operating UI

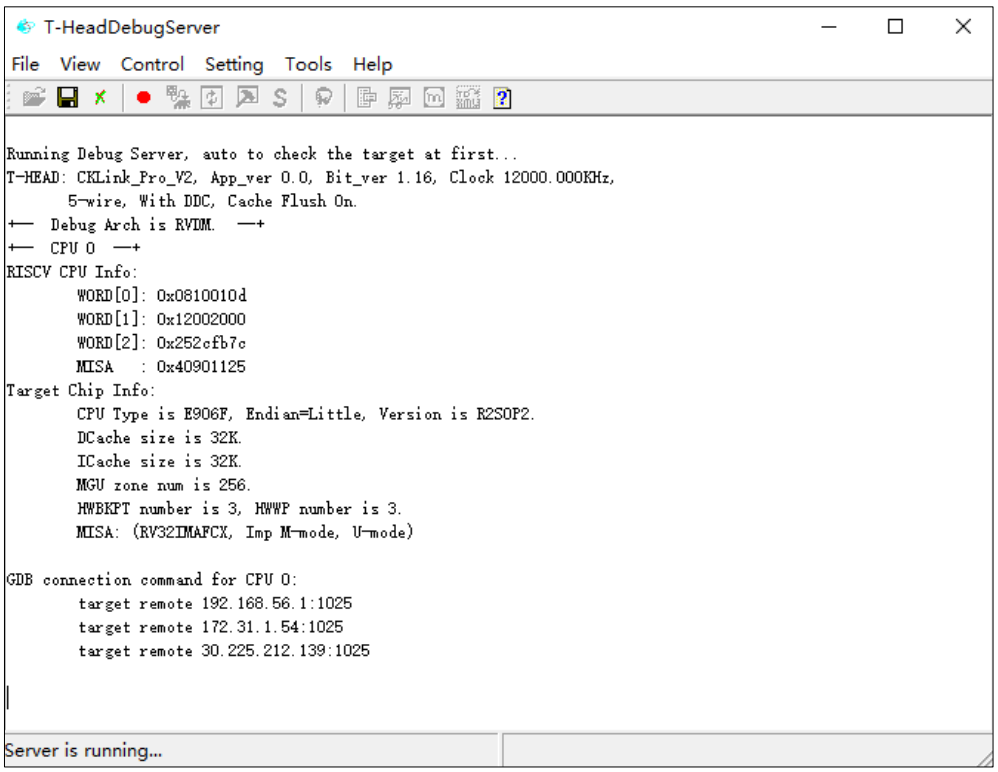


Figure 11-10 Operating UI of T-HEAD DebugServer UI Edition

3. Note

(1) During the running of T-HEAD DebugServer, if a message that indicates "ICE Upgrade" appears, click Yes to upgrade the firmware. After the upgrade is complete, you need to unplug and re-plug the ICE and connect T-HEAD DebugServer to the ICE.

3. T-HEAD DebugServer for Linux

T-HEAD DebugServer for Linux does not supports the UI edition, but supports command lines. Therefore, the operations on T-HEAD DebugServer for Linux are the same as those on T-HEAD DebugServer Console Edition for Windows.

3.1. Install T-HEAD DebugServer and the ICE driver

3.1.1.Obtain the installation package

Obtain the installation package from the OCC platform of T-Head at

https://occ.T-Head.cn/community/download_detail?id=616215132330000384 or from Customer Service. The installation package includes the T-Head-DebugServer-windows*.exe file for Windows systems and two T-Head-DebugSever-linux-*.sh files for 32-bit and 64-bit Linux systems.

The T-Head-DebugServer-linux-i386-*.sh file is the installation package for 32-bit systems and the T-Head-DebugServer-linux-x86_64-*.sh file is the installation package for 64-bit systems. Select an installation package based on the host system.

3.1.2.Install T-HEAD DebugServer

1. During installation, you must obtain sudo permissions.

2. Increase execution permissions for the installation package by running the chmod+x command.

3. Run the following command to start installation: sudo ./T-Head-DebugServer-linux-*.sh –i.

4. When the message "Do you agree to install the DebugServer[yes/no]" appears, type "yes".

5. When the message "Set full installing path:" appears, set the installation path.

(1) Install T-HEAD DebugServer to a user-specified directory: Enter the absolute installation path. During installation, the message "This software will be installed to the path: (User-specified path)? [yes/no/cancel]:" appears. After you confirm that the path is correct, type "yes" and press "Enter". At this time, the installation package starts being installed. After the installation is complete, the following message appears:

"Done!

You can use command "DebugServerConsole" to start DebugServerConsole."

(Note: The full path of "DebugServerConsole.elf" is "User-specified path/T-Head_DebugServer".)

(2) Install T-HEAD DebugServer to the default path: Press "Enter". The message "This software will be installed to the default path: (/usr/bin/)?[yes/no/cancel]:" appears. Type "yes". The software will be installed to the default path "/usr/bin/". After the installation is complete, the following message appears:
"Done!

You can use command "DebugServerConsole" to start DebugServerConsole."

(Note: The full path of "DebugServerConsole.elf" is "/usr/bin/T-Head_DebugServer".)

3.2. Operating environment

The T-HEAD DebugServer can run on most Linux platforms such as Ubuntu. Input and output devices must be equipped with a USB port. The T-HEAD DebugServer can be used with all versions of T-Head GDB.

3.3. Operating parameters

The operating parameters are the same as those described in Section 2.3.1 "Operating parameters."

3.4. JTAG script configuration feature

The JTAG script configuration feature is the same as that described in Section 2.3.2 "Script configuration feature of T-HEAD DebugServer."

3.5. How it works

1. Steps

- After T-HEAD DebugServer is installed, run the "DebugServerConsole + Operating parameters" command to open T-HEAD DebugServer in a directory. If no operating parameters are added, the default operating parameters are used to run T-HEAD DebugServer.
- Run the T-Head GDB application to perform debugging:

  1. Use the T-Head toolchain to generate the T-Head elf program named a.out.
  2. Start T-Head GDB, such as csky-*-gdb a.out.
  3. As prompted by T-HEAD DebugServer, type the connection command in the command line of T-Head GDB. For example, type target remote 172.16.28.168:1025.
  4. After T-Head GDB is connected to T-HEAD DebugServer, perform common operations on T-Head GDB. For example, you can perform the following operations:

     ⑥ load // Downloads the program to the development board.

     ⑦ break main // Sets a breakpoint in the main function.

⑧ continue // Runs the program.

⑨ info registers r0 // Views Register r0.

⑩ print var_a // Views the program variables indicated by var_a.

(1) How to use T-Head GDB is similar to how to use GNU GDB.

- If you use CDK or CDS, see the user documentation for the development environment.

2. Operating UI: The operating UI is the same as that described in Section 2.3.3 "How it works" for T-HEAD DebugServer Console Edition for Windows.

3. Note

(1) Before you run T-HEAD DebugServer, check the /etc/hosts file to see whether the host IP and host name already exist. If no, add them in the first line.

(2) During the running of T-HEAD DebugServer, if a message that indicates "ICE Upgrade" appears, click Yes to upgrade the firmware. After the upgrade is complete, you need to unplug and re-plug the ICE and connect T-HEAD DebugServer to the ICE.

4. Semihosting feature

Semihosting is a debugging method that replaces the input and output of a device with the input and output of the host during debugging. For example, the input and output devices of the host, such as the keyboard, disk, and screen can be used as the input and output of the target board.

Semihosting can reduce the dependence on hardware devices during development. You need only to connect to the target board by using a debugging method.

Semihosting supports the following common operations:

1.open

2.close

3.read

4.write

5.lseek

6.rename

7.unlink

8.stat

9.fstat

10.gettimeofday

11.isatty

12.system

To use the semihosting feature of T-Head 800 series CPUs, perform the following steps:

1. Write a bare T-Head program that contains file operations or inputs and outputs.
2. When you compile a project, add the compilation option "-lsemi" to the compiler. The toolchain must be V3.8.x or later.
3. Use T-HEAD DebugServer to download and debug the program. T-HEAD DebugServer must be V5.2.0 or later.
4. Run the program. Semihosting operations involved in the program will be implemented by T-Head GDB. For example, the output of printf is displayed on the UI of T-Head GDB and the fopen command can be used to open a file in the host where T-Head GDB is located.

By default, semihosting is implemented by T-Head GDB. If you want to use T-HEAD DebugServer to process semihosting requests of a program, add the [-local-semi/-ls] parameter during startup of T-HEAD DebugServer. For more information about how to start T-HEAD DebugServer UI Edition, see Section 2.4 "How to use T-HEAD DebugServer UI Edition." For example, the output of printf is displayed on the UI of T-HEAD DebugServer and the fopen command can be run to open a file in the host where T-HEAD DebugServer is located. By default, semihosting is implemented by T-Head GDB. You can set the [-local-semi/-ls] option to implement semihosting in T-HEAD DebugServer.

To use the semihosting feature of RISC-V CPUs, perform the following steps:

1. For C files, you need to add init_semihosting() to the main function.
2. To compile code by using Toolchain V2.2.0 and later, add --specs=semihost.specs to the link option.
3. Run DebugServerConsole -ls.

4. Debug the code by using T-Head GDB. Semihosting requests that appear in elf will be processed by T-HEAD DebugServer.

Note:

1. T-Head GDB does not support semihosting requests from RISC-V CPUs. Only T-HEAD DebugServer can process these requests.

2. When you add the -local-semi/-ls option in Windows, isatty and system operations are not supported.

5. Debug output feature

T-Head Debug Communication (DCOMM) is a channel that supports output information based on the T-Head JTAG channel during debugging. Only CPUs with the T-Head Light Debug Communication Channel (LDCC) feature are supported.

To use this feature, perform the following steps, where the CK802 with LDCC is used as an example:

1. Write a bare T-Head program and implement fputc. The following sample code is for your reference.

```
#define LDCC_DATA_P          0xe001105c                /* LDCC Register.    */
#define LDCC_BIT_STATUS      0x80000000                /* LDCC Status bit. */
int fputc (int ch, FILE *f)

{
  volatile unsigned int *pdata = LDCC_DATA_P;
  /* Waiting for data read.    */
  while (*pdata & LDCC_BIT_STATUS);
   *pata = ch;
  return 0;
```

```
}
```

2. Use the T-Head ELF toolchain to compile the code and generate an ELF file.
3. Start T-HEAD DebugServer and execute "DebugServerConsole.exe -dcomm=ldcc" For more information about the UI edition, see Section 2.4.3 "Startup configuration file."
4. Connect T-Head GDB to T-HEAD DebugServer and run the program at full speed.
5. The output of printf in the program is transmitted to the UI of T-HEAD DebugServer through an LDCC for display.

6. Command line feature

T-Head allows you to debug a target by using command lines. After T-HEAD DebugServer is started, you can read data from and write data to an HAD register, a CPU registers, and the memory by running specified commands. The following operations are supported:

Command	Description	Example
setclk	Modifies the frequency of CKLink. The default unit is MHz. You can set the unit to kHz.	setclk 3 setclk 3KHz
singlestep/si	Single-steps the instruction execution.	singlestep
sreset	Runs the soft reset command. The number following -c depends on the specific implementation.	sreset -c 0x1234abcd
nreset	Executes an NReset operation on a JTAG connector. During the execution, you can add the halt parameter to make the CPU enter the debug mode again after the reset.	nreset nreset halt
reset	Executes a hard reset operation. During the execution, you can add the halt parameter to make the CPU enter the debug mode again after the reset.	reset reset halt
pctrace	Exports the data of PCFIFO.	pctrace

p/print	1. Prints the values of the register ($) and memory (*). The memory value can be printed only in the word size. If the meaning of the bit field of a register is known, the value of the register is displayed based on the bit field. 2. target: prints the information about the target. 3. cpu: prints the serial number of the selected CPU. This feature is valid only for multi-core CPUs.	p $psr p *0x10000000 p target p cpu
flash xxx	Specifies a flash algorithm file or performs a flash operation. For more information, see Chapter 9 "Flash programming and flash breakpoints."	
set	1. Sets the values of the register ($) and memory (*). The new memory address must belong to a RAM. 2. Specifies whether to set CSR.fdb when exiting the debug mode. The CSR.fdb file is applicable only when the T-Head HAD debug architecture is used. In other words, you can choose whether to make a CPU enter the debug mode or raise a breakpoint exception when a CPU hits the software breakpoint instruction bkpt after the CPU exits the debug mode. 3. Sets how to read data from the memory. Valid values include progbuf, abscmd, and sysbus. After execution, check whether the switching is successful based on the message. 4. Selects the serial number of the current CPU.	set $r0=0x10000000 set *0x10000000=0x1 set resume-with-fdb on set resume-with-fdb off set mem-access progbuf set mem-access abscmd set mem-access sysbus set cpu=1
set resume-bkpt-exception on/off	Modifies the behavior of the CPU after a software breakpoint instruction is executed. Valid values: on: raises a breakpoint exception. off: enters the debug mode.	set resume-bkpt-exception on set resume-bkpt-exception off

set mem-access progbuf/abscmd/sysbus	Modifies the current memory access method of T-HEAD DebugServer when the current architecture is RISC-V DM.	set mem-access progbuf set mem-access abscmd set mem-access sysbus
set virtual-mem-access on/off	Sets whether the memory access method of T-HEAD DebugServer is consistent with the current program. Valid values: on: The memory access method is consistent with the program. off: The address that T-HEAD DebugServer uses to read data from and write data to the memory is a physical address. The default value is on.	
set mem-access-max-mode auto/dword/word/hword/byte	Sets the maximum bit width of memory data to be accessed by T-HEAD DebugServer at a time. The value auto specifies to use the maximum bit width supported by the debug architecture. If the bit width you specify is greater than the maximum bit width supported by the current debug architecture, the maximum bit width supported by the debug architecture is used. For example, for E906, the maximum bit width that the debugging architecture supports at a single time of access is per word. If the bit width is set to word, when a large chunk of memory is accessed, the word-aligned part is first accessed per word. In the non-word-aligned part, the hword-aligned part is accessed per hword and the remaining part is accessed per byte. If the bit width is set to hword, the hword-aligned part is accessed per hword and the remaining part is accessed per byte.	set mem-access-max-mode auto set mem-access-max-mode dword set mem-access-max-mode word set mem-access-max-mode hword set mem-access-max-mode byte

p/print	1. Reads the value of a register ($). If the meaning of the bit field of the register is known, the value of the register is displayed based on the bit field. Such registers include T-HEAD HAD and RISC-V DM registers and all registers of the current CPU. 2. Reads the value of the memory (*). The memory value can be printed only in the word size. 3. Reads the DM register located at the specified DMI address. 4. Views the serial number of the selected CPU. This feature is valid only for multi-core CPUs.	p $psr p $mstatus p *0x10000000 p $dm-reg-0x10 p cpu
p/print target	Prints the information about connected target boards.	p target
p/print dm-reg-list	Prints the names of all viewable DM registers when the current debug architecture is RISC-V DM.	p dm-reg-list
p/print had-reg-list	Prints the names of all viewable HAD registers when the current debug architecture is T-Head HAD.	p had-reg-list

p/print virtual-mem-access	A CPU in the RISC-V DM architecture is in M-mode after the CPU enters the debug mode. However, before the CPU enters the debug mode, the CPU may be in S/U-mode and MMU has been enabled. To ensure that the memory value shown to the debugger is consistent with the memory value shown to a program running on the CPU, when DCSR.prv! is set to 3, T-HEAD DebugServer sets DCSR.mprven to 1, MSTATUS.mpp to DCSR.prv, and MSTATUS.mprv to 1 by default. The preceding features are enabled by default. You can modify the settings by running "set virtual-mem-access on/off" or view the current status by running "p virtual-mem-access".	p virtual-mem-access
pctrace	Exports the data of PCFIFO.	pctrace
reset	Executes a hard reset operation. During the execution, you can add the halt parameter to make the CPU enter the debug mode again after the reset.	reset reset halt
nreset	Executes an NReset operation on a JTAG connector. During the execution, you can add the halt parameter to make the CPU enter the debug mode again after the reset.	nreset nreset halt
sreset	Runs the soft reset command. The number following -c depends on the specific implementation.	sreset -c 0x1234abcd
singlestep/si	Single-steps the instruction execution.	singlestep
setclk	Modifies the frequency of CKLink. The default unit is MHz. You can set the unit to kHz.	setclk 3 setclk 3KHz
q/quit	Exits T-HEAD DebugServer.	q
help	Prints the help information.	help

This feature is provided only in T-HEAD DebugServer Console Edition. After you start T-HEAD DebugServer Console Edition, you can run the preceding commands to perform the required operations. T-HEAD DebugServer UI Edition has implemented the preceding operations on the UI.

The following content lists available registers.

Registers in the T-Head HAD debug architecture: (You can view the registers by running pad-reg-list.)

HAD registers: hid, htcr, mbca, mbcb, pcfifo, baba, babb, bama, bamb, cpuscr, bypass, hcr, hsr, ehsr, wbbr, psr, pc, ir, csr, dccdata, ldccdata, ddcaddr, ddcdata, bsel, hcdi, cpusel, cpust, and hacr

Registers in the RISC-V DM debug architecture: (You can view the registers by running p dm-reg-list.)

Registers in the DM V0.13 architecture: data0, data1, data2, data3, data4, data5, data6, data7, data8, data9, data10, data11, dmcontrol, dmstatus, hartinfo, haltsum1, hawindowsel, hawindow, abstractcs, command, abstractauto, confstrptr0, confstrptr1, confstrptr2, confstrptr3, nextdm, progbuf0, progbuf1, progbuf2, progbuf3, progbuf4, progbuf5, progbuf6, progbuf7, progbuf8, progbuf9, progbuf10, progbuf11, progbuf12, progbuf13, progbuf14, progbuf15, authdata, haltsum2, haltsum3, sbaddress3, sbcs, sbaddress0, sbaddress1, sbaddress2, sbdata0, sbdata1, sbdata2, sbdata3, and haltsum0 In T-Head, itr, customcs, customcmd, and compid registers are also supported.

Registers in the DM V0.11 architecture: dmcontrol, dminfo, authdata0, authdata1, serdata, serstatus, sbaddress0, sbaddress1, sbdata0, sbdata1, haltsum, sbaddress2, sbdata2, sbdata3, cleardebint, serhaltnot, serinfo, sersend0, serrecv0, and serstat0

CPU registers: To view the list of CPU registers, check the register descriptions in the tdescriptions folder in the installation directory of T-HEAD DebugServer.

7. How to use XML files

7.1. Introduction

T-Head GDB implements a set of extended mechanism that uses XML files to describe to-be-debugged registers. This feature is based on T-Head GDB's support for describing registers by using XML files. T-Head GDB builds internal register operation requirements based on the register name, quantity, register group, and other information described in XML

files. This mechanism allows you to use an XML file to describe the information of a target register in the specified UI or command lines.

7.2. XML file formats

7.2.1.Writing rules

T-Head GDB requires that a description file comply with the following rules:

- The description file must comply with the rules for XML files.
- In the XML file, target is the root element with the version attribute. This attribute is 1.0 by default.
- The target element contains the architecture and feature elements at the same level.
- The architecture element indicates the system architecture described by the file. Valid values: arm, mips, and csky.
- The feature element has a name attribute. The name attribute indicates the division of registers.
- The main element contained by the feature element is the reg element. The reg element contains name (register name), bitsize (bit width), regnum (internal serial number of T-Head GDB), type (data type), group (group to which the register belongs), save-restore (internal attribute of T-Head GDB), and other attributes. Among these attributes, the name and bitesize attributes are required and the other attributes are optional.
- The feature element also contains vector, struct, union, and flags elements. These elements are extensions to the description of the target register type and are described in the appendix.

T-Head GDB supplements the following rules on the basis of the preceding rules:

- The target element has the version attribute. This attribute is "1.0" by default. T-Head-GDB supports "1.0" only.
- The value of the architecture element must be set to csky.
- The name attribute of the feature element is org.gnu.csky.abiv1.xxx or org.gnu.csky.abiv2.xxx.
- In each feature element, the first reg element must contain the regnum

attribute. If the regnum attribute in a subsequent reg element is a number following the value of the previous regnum attribute, this reg element can contain no regnum attributes. The value of this regnum attribute increases by 1 based on the value of the regnum attribute in the previous reg element. If the regnum attribute in the subsequent reg element is not a number following the value of the previous regnum attribute, the reg element must contain a regnum attribute.

- The group attribute is required when you need to display the reg element to a group.
- The attribute values of the reg element must not contain special characters.
- The reg element of the Pseudo register is described in the feature element named "org.gnu.csky.pseudo".
- The valid character range of the name and group attributes is "a/A-z/Z, 0-9, _" and the name attribute is up to 15 characters in length.

For T-Head ABI V2 GDB, the name of the feature element must be one of the following strings. If a string that is different from the following strings is used, registers with the feature element cannot be received by T-Head GDB:

1. "org.gnu.csky.abiv2.gpr"
2. "org.gnu.csky.abiv2.fpu"
3. "org.gnu.csky.abiv2.cr"
4. "org.gnu.csky.abiv2.fvcr"
5. "org.gnu.csky.abiv2.mmu"
6. "org.gnu.csky.abiv2.tee"
7. "org.gnu.csky.abiv2.fpu2"
8. "org.gnu.csky.abiv2.bank0"
9. "org.gnu.csky.abiv2.bank1"
10. "org.gnu.csky.abiv2.bank2"
11. "org.gnu.csky.abiv2.bank3"
12. "org.gnu.csky.abiv2.bank4"
13. "org.gnu.csky.abiv2.bank5"

14. "org.gnu.csky.abiv2.bank6"
15. "org.gnu.csky.abiv2.bank7"
16. "org.gnu.csky.abiv2.bank8"
17. "org.gnu.csky.abiv2.bank9"
18. "org.gnu.csky.abiv2.bank10"
19. "org.gnu.csky.abiv2.bank11"
20. "org.gnu.csky.abiv2.bank12"
21. "org.gnu.csky.abiv2.bank13"
22. "org.gnu.csky.abiv2.bank14"
23. "org.gnu.csky.abiv2.bank15"
24. "org.gnu.csky.abiv2.bank16"
25. "org.gnu.csky.abiv2.bank17"
26. "org.gnu.csky.abiv2.bank18"
27. "org.gnu.csky.abiv2.bank19"
28. "org.gnu.csky.abiv2.bank20"
29. "org.gnu.csky.abiv2.bank21"
30. "org.gnu.csky.abiv2.bank22"
31. "org.gnu.csky.abiv2.bank23"
32. "org.gnu.csky.abiv2.bank24"
33. "org.gnu.csky.abiv2.bank25"
34. "org.gnu.csky.abiv2.bank26"
35. "org.gnu.csky.abiv2.bank27"
36. "org.gnu.csky.abiv2.bank28"
37. "org.gnu.csky.abiv2.bank29"
38. "org.gnu.csky.abiv2.bank30"
39. "org.gnu.csky.abiv2.bank31"
40. "org.gnu.csky.linux"

T-Head ABI V2 GDB has the following additional requirements for register names:

A control register in bank0 can be described by using cr0 to cr31. In T-Head GDB, control registers in bank1 to bank31 can be named in the format "cpxcry" by default.

In the format "cpxcry", x represents a bank ID ranging from 1 to 31, and y represents the serial number of the control register in the group, ranging from 0 to 31. For example, Register 21 in bank4 is described as cp4cr21.

If you need to use other names, you need to add support in T-Head GDB first.

7.2.2.Sample description

The following code is a sample XML file that T-Head uses to describe a register:

```
<?xml version=”1.0”?>
<!DOCTYPE target SYSTEM “gdb-target.dtd”>
<target version=”1.0”>
<architecture>csky</architecture>
  <feature name=”org.gnu.csky.abiv1.gpr”>
    <reg  name=”r0” bitesize=”32”  regnum=”0”  type=”uint32”
group=”gpr” save-restore=”yes”/>
    <reg  name=”r1” bitesize=”32”  regnum=”1” group=”gpr”/>
    <reg  name=”r2” bitesize=”32”  regnum=”2” group=”gpr”/>
    …
  </feature>
  <feature name=”org.gnu.T-Head.pseudo”>
    <reg name=”r01” bitesize=”64” type=”uint64” group=”pseudo” regs
=”0,1”/>
    <reg name=”memr” bitesize=”32” type=“uint32” addr=”0x1234”
group=”pseudo”/>
    …
  </feature>
</target>
```

The following content describes the XML file:

- In the XML file, target is the root element with the version attribute. This attribute is 1.0 by default.

- The architecture element indicates that this file describes the T-Head architecture.
- The name attribute of the feature element is set to org.gnu.csky.abiv1.gpr. The value indicates that the feature element describes a GPR of ABI V1 in the T-Head architecture.
- For the first register, the name is r0, the bit width is 32, the internal serial number in T-Head GDB is 0, the data type is uint32, and this register belongs to Register Group gpr.
- For the second register, the name is r1, the bit width is 32, and the internal serial number in T-Head GDB is 1.
- For the third register, the name is r2, the bit width is 32, and the internal serial number in T-Head GDB is 2.
- The name attribute of the second feature element is set to org.gnu.csky.pseudo. The value indicates that the feature element functions as a specific feature that describes a pseudo register.
- For the first register in the second feature element, the name is r01, the bit width is 64, the data type is uint64, and the serial numbers of the register are 0 and 1.

Register numbers used in the XML file, that is, the values of the regnum attribute, are the values of the Serial number in T-Head GDB column in Table 11-9 Serial numbers of ABI V1 registers and Table 11-10 Serial numbers of ABI V2 registers. The following two tables list register numbers that are supported and expected to be supported. Register names are not all listed. After T-Head-GDB supports XML, register names can be modified as required.

7.2.2.1. ABI V1 registers

The following ABI V1 registers are available:

- General registers r0 to r15
- Optional registers r0 to r15
- Control registers cr0 to cr31
- Hi and Lo registers
- FPUs

- PC
- FPCR
- MMUs

The following table lists the serial numbers of registers.

Table 11-9 Serial numbers of ABI V1 registers

Register name	Serial number in T-Head GDB	Description
R0~r15	0~15	
Hi,Lo	20, 21	
FPU	24~55	
PC	72	
Optional registers r0~r15	73~88	
Cr	89~119	No Register cr31 is available.
FPCR (The registers are named cp1cr0 to cp1cr6.)	121~127	
MMU (The registers are named cp15cr0 to cp15cr16 and cp15cr29 to cp15cr31.)	128~147	A total of 20 MMU registers are available. The serial numbers in T-Head GDB are continuous.

7.2.2.2. ABI V2 registers

The following ABI V2 registers are available:

- General registers r0~r31
- Optional registers r0~r15
- Hi,lo

- FPU/VPU
- Profiling
- PC
- Cr_Bank0
- Cr_Bank1
- Cr_Bank3
- Cr_Bank15

The following table lists the serial numbers of registers.

Table 11-10 Serial numbers of ABI V2 registers

Register name	Serial number in T-Head GDB	Description
R0~r15	0~15	
R16~r31	16~31	
Hi, lo	36, 37	
FPU/VPU	40~71	
PC	72	
Ar0~ar15	73~88	
Cr0~cr31	89~120	
FPU/VPU_CR	121~123	
Usp	127	
Mmu (bank15)	128~136	
Prof-soft-general	140~143	
Prof-cr	144~157	
Prof-arch	160~174	
Prof-exten	176~188	
Bank1	189~220	
Bank3	221~252	
All registers in Bank15 except the MMU	253~275	
Bank2	276~307	
Bank4	308~339	
Bank5	340~371	

Bank6	372~403	
Bank7	404~435	
Bank8	436~467	
Bank9	468~499	
Bank10	500~531	
Bank11	532~563	
Bank12	564~595	
Bank13	596~627	
Bank14	628~659	
Bank16	660~691	
Bank17	692~723	
Bank18	724~755	
Bank19	756~787	
Bank20	788~819	
Bank21	820~851	
Bank22	852~883	
Bank23	884~915	
Bank24	916~947	
Bank25	948~979	
Bank26	980~1011	
Bank27	1012~1043	
Bank28	1044~1075	
Bank29	1076~1107	
Bank30	1108~1139	
Bank31	1140~1171	
Cr16~cr31	1172~1187	

7.2.3.Extended TEE registers

7.2.3.1. Introduction to the extension

In the T-Head TEE (secure) programming model, the same register has a copy in both the TEE and REE worlds. To read data from and write data to these registers, you need to switch between the worlds or map the registers. For more information, see the manuals related to T-Head TEE. T-Head GDB and the xml files of T-HEAD DebugServer help users view register information of the current world and the other world in either world. T-Head GDB must be

V3.10.0 and later and T-HEAD DebugServer must be V4.5.0 and later. T-HEAD DebugServer helps switch between the worlds and read registers in the current world.

7.2.3.2. Writing rules

1. The description of a TEE register is provided in a pseudo register. Therefore, the rules for describing TEE registers must meet all the rules related to pseudo registers in Section7.2.1.

2. A TEE register uses the "env" attribute to indicate whether the register is a TEE or REE register. The attribute can only be set to "ree" or "tee".

3. Use the "regs" attribute to indicate the serial numbers of physical registers that correspond to the registers.

4. Set the values of bytesize and type based on the actual attributes.

7.2.3.3. Examples

The following content is a sample register description file, with CK810T being an example. You can refer to the file in the tdescriptions folder in the installation directory of T-HEAD DebugServer V5.4.0 and later.

```
<?xml version="1.0"?>
<target>
  <architecture>csky</architecture>
  <feature name="org.gnu.csky.abiv2.gpr">
    <reg name="r0"    bitsize="32" regnum="0"    group="gpr"/>
    <reg name="r1"    bitsize="32" regnum="1"    group="gpr"/>
    <reg name="r2"    bitsize="32" regnum="2"    group="gpr"/>
    <reg name="r3"    bitsize="32" regnum="3"    group="gpr"/>
    <reg name="r4"    bitsize="32" regnum="4"    group="gpr"/>
    <reg name="r5"    bitsize="32" regnum="5"    group="gpr"/>
    <reg name="r6"    bitsize="32" regnum="6"    group="gpr"/>
    <reg name="r7"    bitsize="32" regnum="7"    group="gpr"/>
    <reg name="r8"    bitsize="32" regnum="8"    group="gpr"/>
    <reg name="r13" bitsize="32" regnum="13" group="gpr"/>
    <reg name="r14" bitsize="32" regnum="14" group="gpr"/>
    <reg name="r15" bitsize="32" regnum="15" group="gpr"/>
    <reg name="pc"    bitsize="32" regnum="72"/>
  </feature>
  <feature name="org.gnu.csky.abiv2.cr">
    <reg name="psr"    bitsize="32" regnum="89"    group="cr"/>
    <reg name="vbr"    bitsize="32" regnum="90"    group="cr"/>
    <reg name="epsr" bitsize="32" regnum="91"    group="cr"/>
    <reg name="epc"    bitsize="32" regnum="93"    group="cr"/>
    <reg name="ss4"    bitsize="32" regnum="99"    group="cr"/>
    <reg name="gcr"    bitsize="32" regnum="100" group="cr"/>
    <reg name="gsr"    bitsize="32" regnum="101" group="cr"/>
```

```
    <reg name="cpuid"    bitsize="32" regnum="102" group="cr"/>
    <reg name="ccr"    bitsize="32" regnum="107" group="cr"/>
    <reg name="capr"    bitsize="32" regnum="108" group="cr"/>
    <reg name="pacr"    bitsize="32" regnum="109" group="cr"/>
    <reg name="prsr"    bitsize="32" regnum="110" group="cr"/>
    <reg name="chr"    bitsize="32" regnum="120" group="cr"/>
  </feature>
  <feature name="org.gnu.csky.abiv2.tee">
    <reg name="nt_usp"    bitsize="32" regnum="127" group="ree"/>
    <reg name="ebr"    bitsize="32" regnum="190" group="cr"/>
    <reg name="dcr"    bitsize="32" regnum="229" group="cr"/>
    <reg name="t_usp"    bitsize="32" regnum="228" group="tee"/>
    <reg name="t_pcr"    bitsize="32" regnum="230" group="tee"/>
  </feature>
  <feature name="org.gnu.csky.pseudo">
    <reg name="t_psr"      bitsize="32" regs="89"      group="tee" type="int32" env="tee"/>
    <reg name="t_vbr"      bitsize="32" regs="90"      group="tee" type="int32" env="tee"/>
    <reg name="t_epsr"    bitsize="32" regs="91"      group="tee" type="int32" env="tee"/>
    <reg name="t_epc"      bitsize="32" regs="93"      group="tee" type="int32" env="tee"/>
    <reg name="t_ebr"      bitsize="32" regs="190"    group="tee" type="int32" env="tee"/>
    <reg name="nt_psr"    bitsize="32" regs="89"      group="ree" type="int32" env="ree"/>
    <reg name="nt_vbr"    bitsize="32" regs="90"      group="ree" type="int32" env="ree"/>
    <reg name="nt_epsr"    bitsize="32" regs="91"    group="ree" type="int32" env="ree"/>
    <reg name="nt_epc"      bitsize="32" regs="93"    group="ree" type="int32" env="ree"/>
    <reg name="nt_ebr"      bitsize="32" regs="190"    group="ree" type="int32" env="ree"/>
  </feature>
```

</target>

As described in the preceding XML file, the psr that you view in T-Head GDB is the psr of the current world, the t_psr is the psr of the secure world, and the nt_psr is the psr of the non-secure world. In other words, psr may be the same as t_psr or nt_psr, depending on the current world.

7.2.4.Specify an XML file in T-HEAD DebugServer UI Edition

T-HEAD DebugServer UI Edition allows you to specify an XML file in the startup configuration file default.ini and specify an XML file in the UI.

In the default.ini file:

The default.ini is the default startup configuration file of T-HEAD DebugServer UI Edition. The file provides the variable "TDESCXMLFILE" for you to set the path to an XML file. You can assign this variable the prepared path to the XML file. When T-Head GDB initiates a request to obtain the XML file that describes the information about a target, T-HEAD DebugServer opens the user-specified XML file and interacts with T-Head GDB.

After you modify the path to the XML file in the UI, a message appears when you close T-HEAD DebugServer, asking you whether you are sure to modify the default configuration. If you click Yes, the current relevant configurations of T-HEAD DebugServer are saved as a configuration file.

In the UI edition:

T-HEAD DebugServer UI Edition provides a UI for you to select an XML file. The TDFile Setting option is added to the Setting menu, as shown in the following figure.

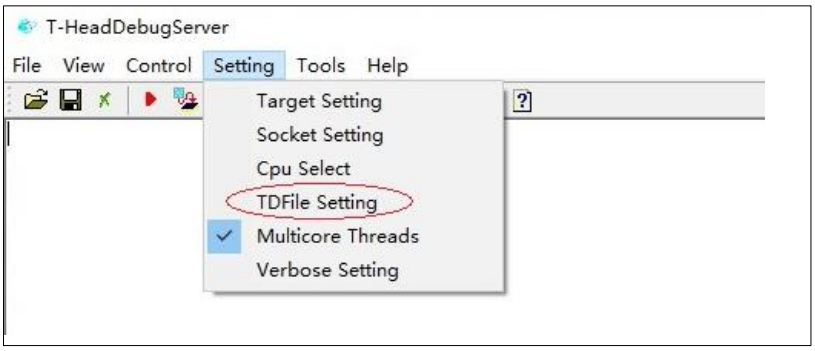


Figure 7-11-11 TDFile Setting option in the menu

A shortcut icon is added to the toolbar to select an XML file, as shown in the following figure.

Figure 7-11-12 Shortcut to TDFile Setting

After you click either the preceding option or icon, the Target Description File dialog box appears, where you can select an XML file.

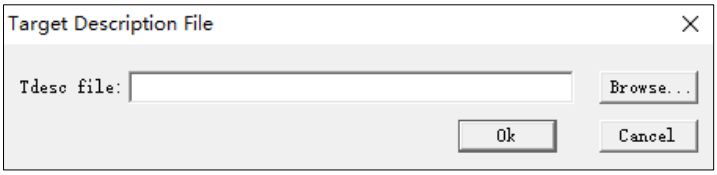


Figure 7-11-13 Target Description File dialog box

Click Browse, select a local XML file that describes the target, and click OK.

7.2.5.Specify an XML file in T-HEAD DebugServer Console Edition

In T-HEAD DebugServer Console Edition, the startup parameter "-tdescfile +filepath" is added to specify the xml file that describes the target register.

In Windows environments:

1) Create a shortcut for T-HEAD DebugServer Console Edition.

2) Right-click the shortcut and click Properties.

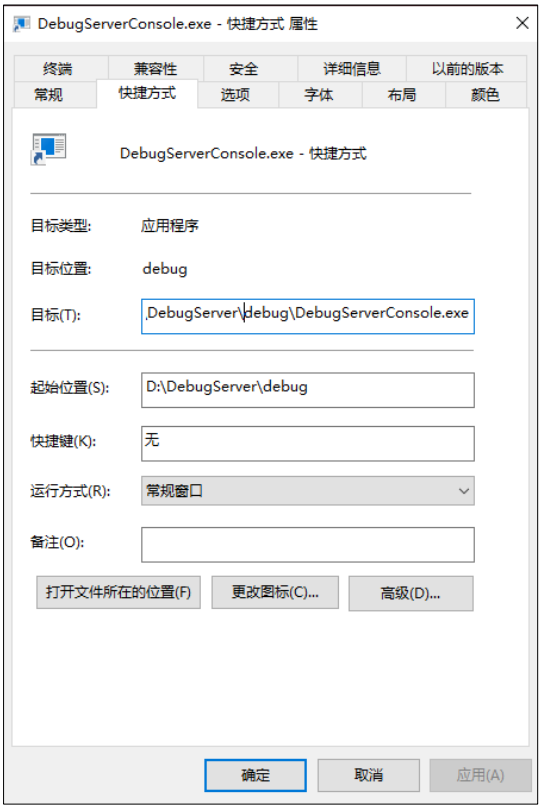


Figure 7-11-14 Properties of the shortcut to T-HEAD DebugServer

3) On the shortcut properties page, append "-tdescfile xmlpath" to the program name in the Target field, where xmlpath is the path to the XML file. You can specify the path as required.

Figure 7-11-15 Add a startup parameter to the shortcut

4) Click "OK". Start the shortcut and connect T-Head GDB to T-HEAD DebugServer for debugging.

Figure 7-11-16 Clicking OK after modifying the startup parameter

In Linux environments:

Run the following command to start T-HEAD DebugServer for Linux: sudo ./Debugserver.elf -tdescfile /home/xxx/gdbxml/csky-abiv2.xml. Then, connect T-Head GDB to T-HEAD DebugServer for debugging.

8. Multi-core debugging

8.1. Introduction

T-Head C860MP is a 32-bit ultra-high-performance embedded multi-core processor. It is designed for embedded systems and SoC applications. During debugging, a debug framework with multiple cores and a single JTAG connector is used. T-Head C860MP accesses the auxiliary debug unit (HAD) of each core by using a shared JTAG connector, to trigger the cores to enter and exit the debug mode and use processor resources.

In addition, T-Head C860MP provides a centralized event transmission module (ETM) to support the transmission of debug events (entering and exiting the debug mode) between multiple cores. When a C860 core receives a debug command sent by an ICE, C860 generates a debug entry event or a debug exit event and send the event to the ETM. The ETM determines whether to forward the event to other cores. This way, the debug entry and exit signals are exchanged between multiple cores and are responded by other cores.

In other words, when one core enters or exits the debug mode, you can choose to send this signal to other cores. Other cores can choose to respond or not to respond to this signal and enter or exit the debug mode. The following figure shows the hardware model.

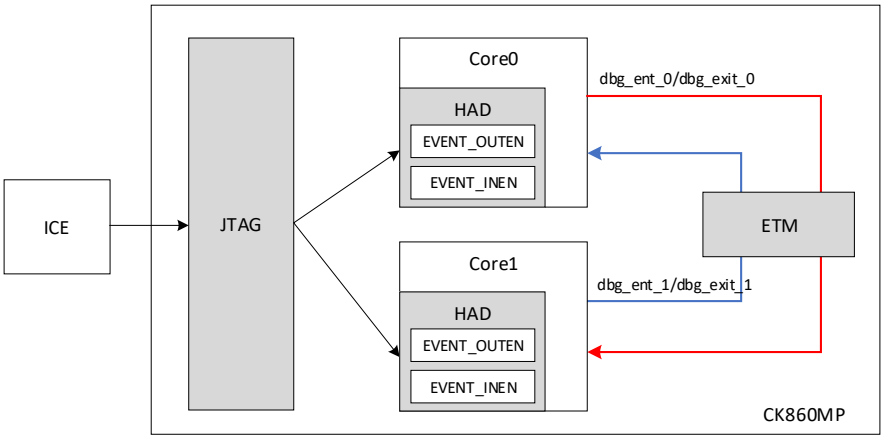


Figure 8-11-17 Overall multi-core debugging framework

T-Head has also implemented the ETM module for RISC-V multi-core C910MP. The implementation and debugging method are the same as those of T-Head C860MP.

In the multi-core debugging framework, T-HEAD DebugServer supports two debug modes: multi-core single-port debugging and multi-core multi-port debugging. The following sections describe the two debugging modes.

8.2. Debugging environment requirements

Hardware requirements

1. T-Head C860MP development board

2. One T-Head ICE CKLINK_PRO_V2 box with supporting FFC and USB cables

3. PC running in the Windows system with MinGW or running in the Linux system

Software requirements:

1. csky-*abiv2*-gdb V3.6.x Toolchain

2. T-HEAD DebugServer V4.4.0 (Obtain T-HEAD DebugServer for Windows or T-HEAD DebugServer for Linux as required.)

8.3. Multi-core single-port mode

In this mode, T-HEAD DebugServer provides only one service port to connect to T-Head GDB. After you connect T-Head GDB to T-HEAD DebugServer by running "target remote ip:port", T-HEAD DebugServer encapsulates the information about multiple cores into thread information and sends the information to T-Head GDB. You can view and debug multiple CPUs by using the thread command on T-Head GDB.

In this mode, multiple CPUs send debug signals to each other and respond to the debug signals of other cores. In other words, when one CPU enters or exits the debug mode, other CPUs also enter or exit the debug mode at the same time.

The following figure shows the multi-core single-port model.

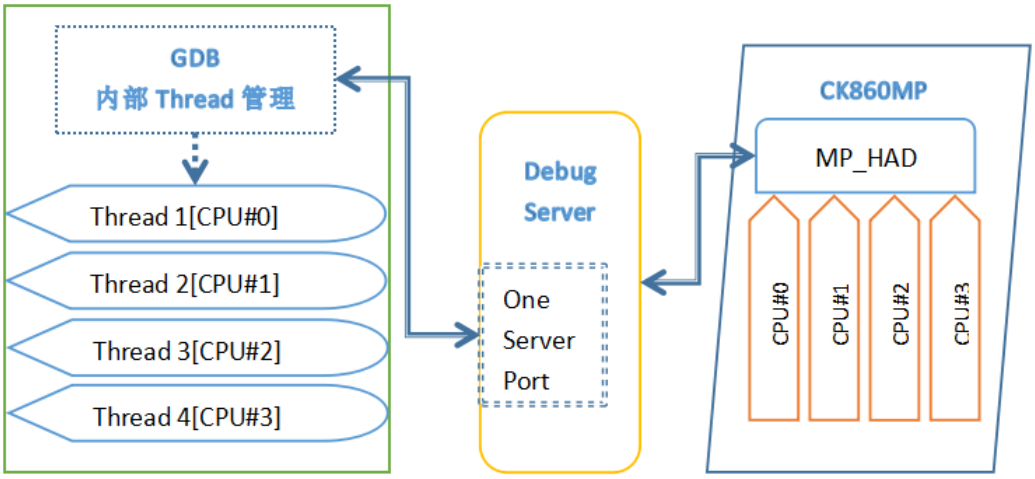


Figure 8-11-18 Multi-core single-port model

8.3.1.Procedure

In the following steps, two-core C860MP is used as an example.

1. Power on the development board, connect to the ICE, and connect the USB cable from the ICE to the PC.

2. Run T-HEAD DebugServer. The following figures show the outputs in different editions.

(1) Output on the UI edition

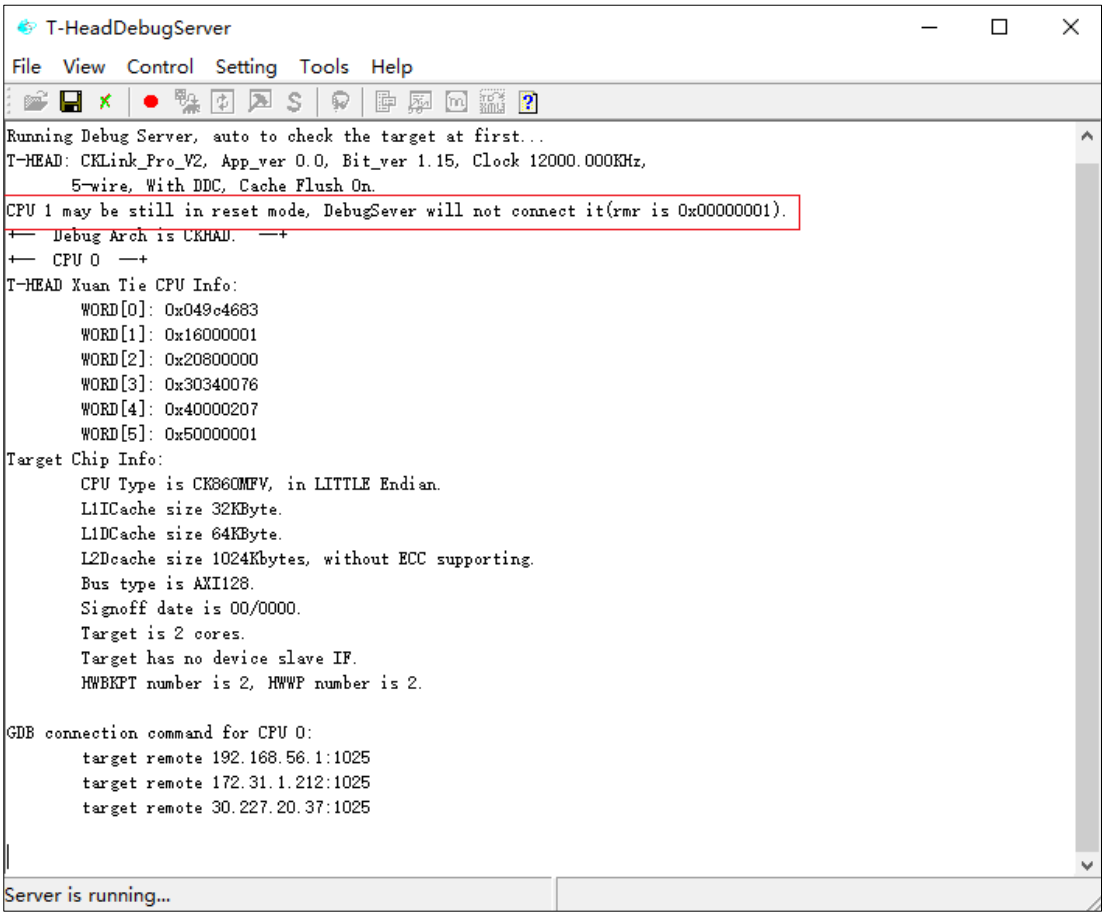


Figure 8-11-19 Output on T-HEAD DebugServer UI Edition after first connection to C860MP that is powered on soon

(2) In this case, the output on the console edition for Linux and Windows is the same as that on the UI edition for Windows.

Figure 8-11-20 Output on T-HEAD DebugServer Console Edition after first connection to C860MP that is powered on soon (Same output for Windows and Linux)

Then, T-HEAD DebugServer prints the CPUID information of CPU 0, but fails to print the CPUID information of CPU 1. This error message is normal, because all cores except CPU 0 are in the reset state after C860MP is powered on for the first time. The JTAG state machine cannot obtain the information about CPU 1 and therefore the information about CPU 1 cannot be printed.

3. Start T-Head GDB to wake up CPU 1.

(1) Start csky-*abiv2*-gdb.

(2) In the command line of T-Head GDB, type "target remote ip:port". This command is the connection command shown on the UI of T-HEAD DebugServer.

(3) After T-HEAD DebugServer is connected, type the following command line to wake up CPU 1: set $cr29 = 3. For the description of cr29, see the user manual of C860MP.

```
GNU gdb (C-SKY Tools V3.7.4-ck805 Minilibc abiv2) 7.12
Copyright (C) 2016 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=x86_64-pc-linux-gnu --target=csky-elfabiv2".
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".
(cskygdb) target remote 172.16.150.77:1025
Remote debugging using 172.16.150.77:1025
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x1fbd819c in ?? ()
(cskygdb) set $cr29=0x3
(cskygdb) quit
A debugging session is active.

        Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) y
Detaching from program: , Remote target
Ending remote debugging.
[toolsbuild@yulinglong ~]$
```

Figure 8-11-21 Start T-Head GDB and wake up CPU 1

(4) Type quit in the command line to exit T-Head GDB.

4. Restart T-HEAD DebugServer to identify multiple cores.

(1) Perform the following steps on T-HEAD DebugServer UI Edition:

① Click the Pause icon.

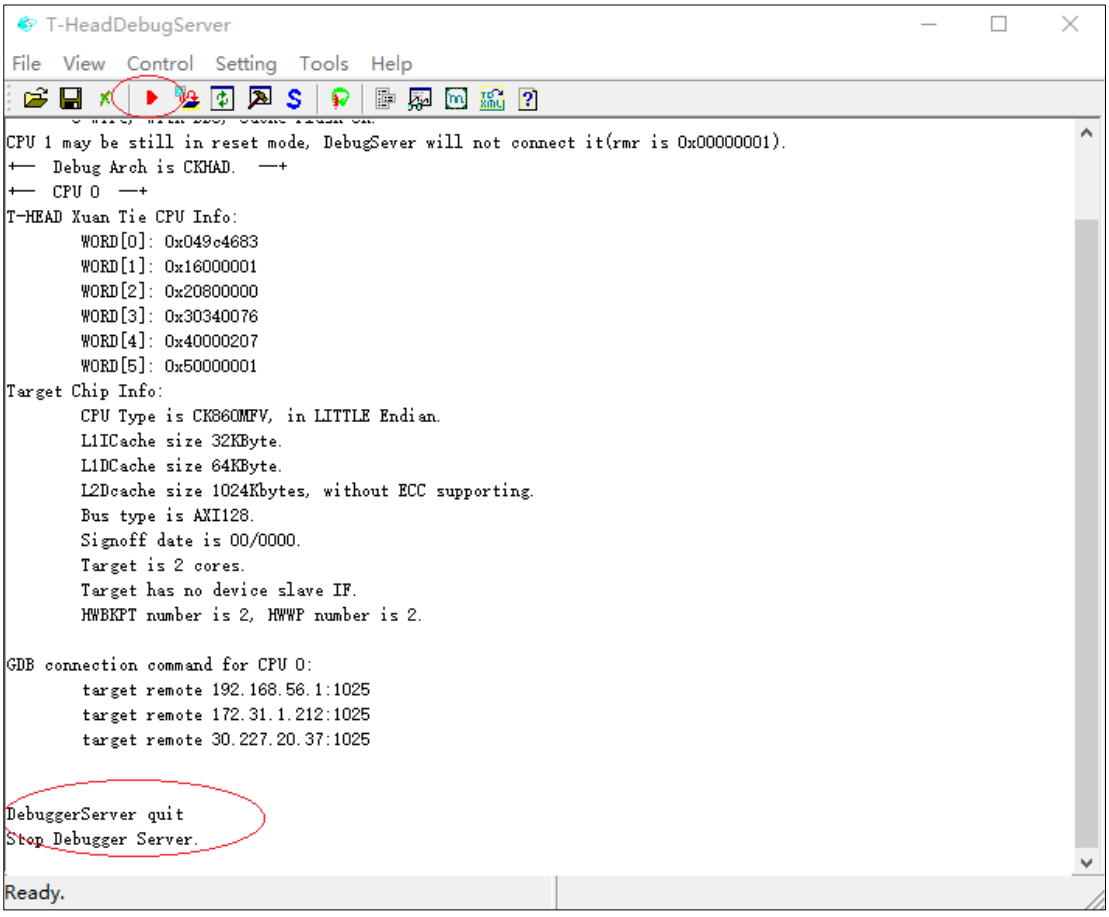


Figure 8-11-22 Disconnect from T-HEAD DebugServer UI Edition

② Reconnect to T-HEAD DebugServer. The following figure shows the output.

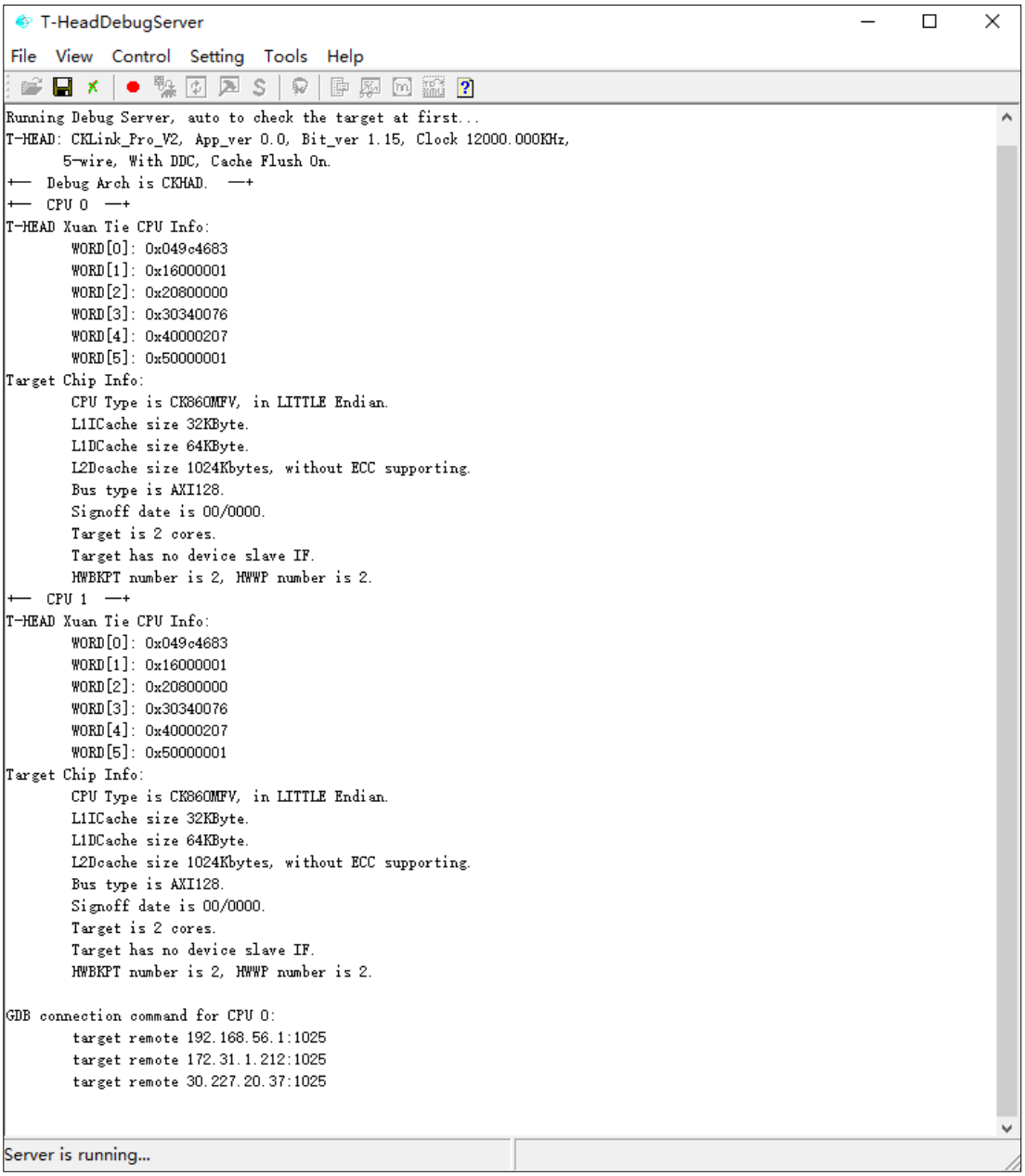


Figure 8-11-23 Output on T-HEAD DebugServer UI Edition after connection to C860MP in the multi-core single-port mode

At this time, T-HEAD DebugServer prints the CPUID information of CPU 0 and CPU 1.

(2) Perform the following steps on T-HEAD DebugServer Console Edition for Windows or Linux:

① Press Ctrl+C to stop T-HEAD DebugServer Console Edition.

② Run T-HEAD DebugServer Console Edition again.

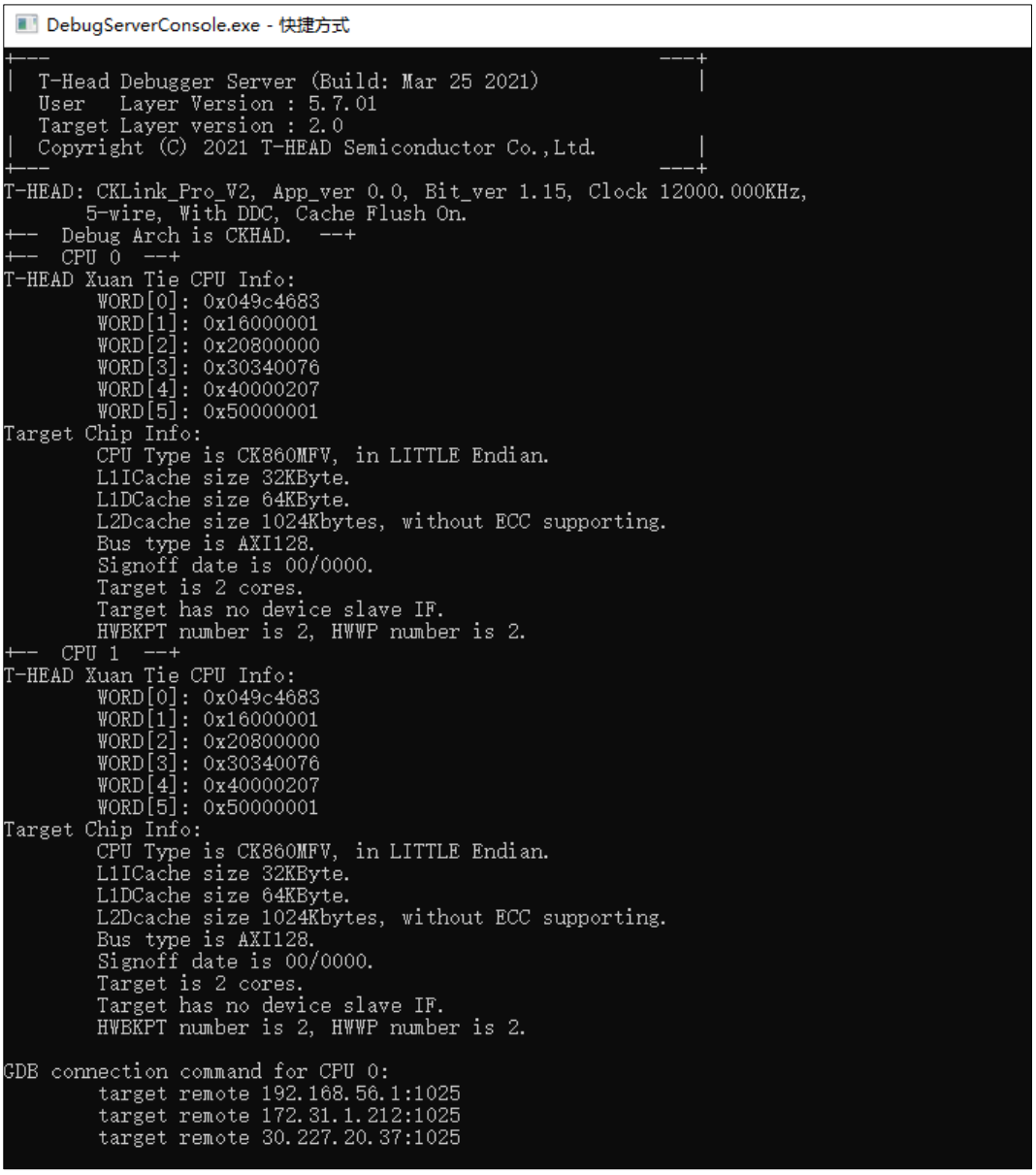


Figure 8-11-24 Output on T-HEAD DebugServer Console Edition after connection to C860MP in the multi-core single-port mode (Same output for Windows and Linux)

At this time, T-HEAD DebugServer prints the CPUID information of CPU 0 and CPU 1.

5. Start T-Head GDB to debug multiple cores.

(1) Start csky-*abiv2*-gdb.

(2) In the command line of T-Head GDB, type "target remote ip:port". This command is the connection command shown on the UI of T-HEAD DebugServer.

(3) In the command line of T-Head GDB, type "info threads". The following figure shows the output.

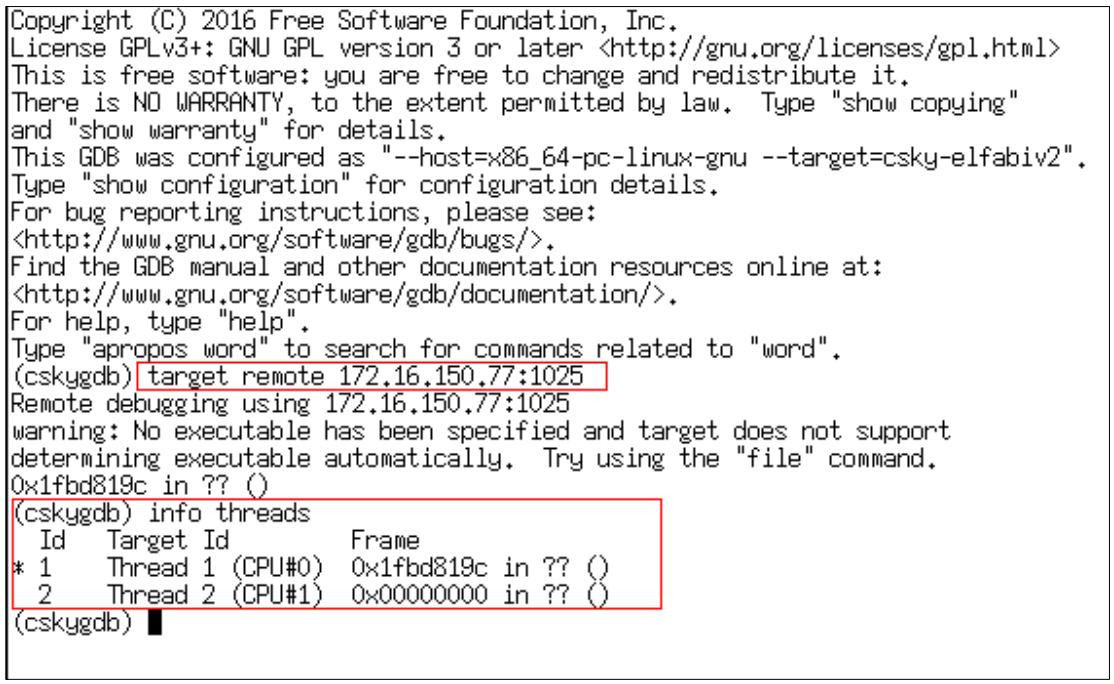


Figure 8-11-25 View threads by running "info thread" in T-Head GDB

As shown in the preceding figure, T-Head GDB displays Thread 1 and Thread 2. Thread 1 corresponds to CPU 0, and Thread 2 corresponds to CPU 1.

8.3.2.Thread-based operations

After you perform the steps described in Section 8.3.1, threads of T-Head GDB correspond to CPUs on a one-to-one basis. You can perform operations as needed.

1. View the register information of CPU 1 by typing the following command in the command line of T-Head GDB:

(1) thread 2 (Switches an internal thread of T-Head GDB to Thread 2.)

(2) info registers (T-Head GDB displays the information about registers such as the GPR, PC, and PSR of CPU 1.)

```
Copyright (C) 2016 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=x86_64-pc-linux-gnu --target=csky-elfabiv2".
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".
(cskygdb) target remote 172.16.150.77:1025
Remote debugging using 172.16.150.77:1025
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x1fbd819c in ?? ()
(cskygdb) info threads
  Id   Target Id         Frame
* 1    Thread 1 (CPU#0)  0x1fbd819c in ?? ()
  2    Thread 2 (CPU#1)  0x00000000 in ?? ()
(cskygdb) thread 2
[Switching to thread 2 (Thread 2)]
#0  0x00000000 in ?? ()
(cskygdb) i r
r0             0xb60cd4 11930836
r1             0x0      0
r2             0x80c9f8f6       -2134247178
r3             0x0      0
r4             0x1      1
r5             0x1fff   8191
r6             0x965ba8 9853864
r7             0x2      2
r8             0x0      0
r9             0x9f060000       -1626996736
r10            0x9f060000       -1626996736
r11            0x0      0
r12            0x9f060000       -1626996736
r13            0xffffe000       -8192
r14            0x1dcd6500       0x1dcd6500
r15            0x80ec3d30       -2132001488
r16            0xb60cd4 11930836
r17            0x0      0
r18            0x80c9f8f6       -2134247178
r19            0x0      0
r20            0x1      1
r21            0x1fff   8191
r22            0x965ba8 9853864
r23            0x2      2
r24            0x0      0
r25            0x9f060000       -1626996736
r26            0x9f060000       -1626996736
r27            0x0      0
r28            0x9f060000       -1626996736
r29            0xffffe000       -8192
r30            0x1dcd6500       500000000
r31            0x80ec3d30       -2132001488
pc             0x0      0x0
epc            0x0      0x0
psr            0x80000000       -2147483648
epsr           0x0      0
(cskygdb)
```

Figure 8-11-26 Switch a thread of T-Head GDB to view register information of CPU 1

2. View register information of CPU 0

(1) thread 1 (Switches an internal thread of T-Head GDB to Thread 1.)

(2) info registers (T-Head GDB displays the information about registers such as the GPR, PC, and PSR of CPU 0.)

```
(cskygdb) thread 1
[Switching to thread 1 (Thread 1)]
#0  0x1fbd819c in ?? ()
(cskygdb) i r
r0             0x33     51
r1             0xa      10
r2             0x20     32
r3             0x20     32
r4             0x16     22
r5             0x80e94000       -2132197376
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x80e95f94       -2132189292
r11            0x0      0
r12            0x3      3
r13            0x1      1
r14            0x1fc1ffe4       0x1fc1ffe4
r15            0x1fbd819c       532513180
r16            0x33     51
r17            0xa      10
r18            0x20     32
r19            0x20     32
r20            0x16     22
r21            0x80e94000       -2132197376
r22            0x0      0
r23            0x0      0
r24            0x0      0
r25            0x0      0
r26            0x80e95f94       -2132189292
r27            0x0      0
r28            0x3      3
r29            0x1      1
r30            0x1fc1ffe4       532807652
r31            0x1fbd819c       532513180
pc             0x1fbd819c       0x1fbd819c
epc            0x0      0x0
psr            0x80000101       -2147483391
epsr           0x0      0
(cskygdb)
```

Figure 8-11-27 Switch a thread of T-Head GDB to view register information of CPU 0

3. Set the PC value of CPU 0 to 0x10000

(1) If Thread 1 is active, type "Thread 1" again to switch to Thread 1. The thread number is

the serial number of the CPU plus 1. You can type "thread" or "info threads" in the T-Head GDB command line to display the current threads in T-Head GDB.

```
(cskygdb) thread
[Current thread is 1 (process <main>)]
(cskygdb) info threads
  Id   Target Id         Frame
* 1    process <main> (CPU#0 [running]) 0x1fbd819c in ?? ()
  2    process <main> (CPU#1 [running]) 0x00000000 in ?? ()
(cskygdb)
```

Figure 8-11-28 View current threads in T-Head GDB

In the printed information, the lines starting with an asterisk (*) is the current thread.

(2) set $pc = 0x10000 (Sets the PC value of CPU 0 to 0x10000.)

```
(cskygdb) thread
[Current thread is 1 (process <main>)]
(cskygdb) info threads
  Id   Target Id         Frame
* 1    process <main> (CPU#0 [running]) 0x1fbd819c in ?? ()
  2    process <main> (CPU#1 [running]) 0x00000000 in ?? ()
(cskygdb) thread 1
[Switching to thread 1 (process <main>)]
#0  0x1fbd819c in ?? ()
(cskygdb) set $pc=0x10000
```

Figure 8-11-29 Switch a thread of T-Head GDB and set the PC value of CPU 0 to 0x10000

4. During all viewing and modification operations, you need to check the registers of the specified CPU. Before you set the registers and check the backtrace and memory, you need to check whether threads that correspond to the cores are correct.

5. Run the program: In this mode, multiple cores respond to debug signals from each other. In other words, when you type step i, step, next, or continue in the command line of T-Head GDB to run the program, the command is sent to the current CPU. However, when a core exists the debug mode, the debug signal will be responded to by other cores.

(1) When you type si in the command line, all CPU cores execute the si command.

(2) When you type step or continue in the command line, the CPU that corresponds to the current thread exits the debug mode, and other CPUs also exit the debug mode at the same time. When one of CPUs enters the debug mode due to a breakpoint or other reasons, other CPUs are pulled into the debug mode by the CPU that first enters the debug mode. After T-Head GDB obtains the message that indicates the CPU enters the debug mode, T-Head GDB prints the relevant information and switches the current

thread to the thread that corresponds to the CPU that first enters the debug mode.

6. Set a breakpoint

(1) Set a software breakpoint, which is valid for all CPUs.

(2) Set a hardware breakpoint, which is valid for all CPUs.

(3) To make a specified CPU stop at a breakpoint, add thread information when you set the breakpoint. For example, type the following command:

break *0x10000 thread 2

This command means that when Thread 2, which corresponds to CPU 1, hits the breakpoint, T-Head GDB stops.

8.4. Multi-core multi-port mode

In this mode, T-HEAD DebugServer provides one service port for each CPU to connect to T-Head GDB. CPUs are divided by port. After you connect T-Head GDB to T-HEAD DebugServer by running "target remote ip:port", the CPU that corresponds to the port is specified. You can view the correspondence between the CPU and the port on T-HEAD DebugServer.

In this mode, multiple CPUs do not send debug signals to each other or respond to the debug signals of other cores. In other words, a debugged core is independent. When one CPU enters or exits the debug mode, other CPUs can still keep operating. You can deem that you are debugging multiple development boards, only that the multiple development boards share the same memory.

The following figure shows the multi-core multi-port model.

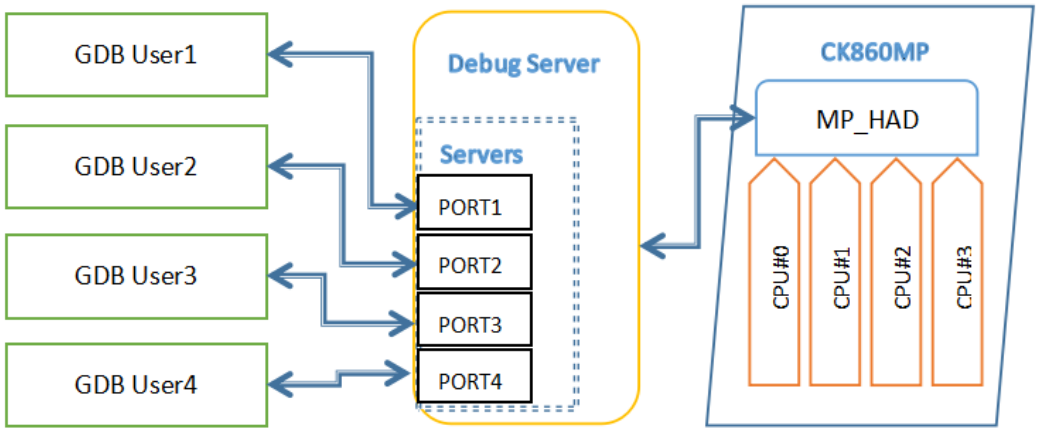


Figure 8-11-30 Multi-core multi-port model

8.4.1.Procedure

In the following steps, two-core C860MP is used as an example.

1. This step is the same as Step 1 in Section 8.3.1 "Procedure."

2. This step is the same as Step 2 in Section 8.3.1 "Procedure."

3. This step is the same as Step 3 in Section 8.3.1 "Procedure."

4. Restart T-HEAD DebugServer to identify multiple cores.

(1) Perform one of the following steps on T-HEAD DebugServer UI Edition:

① Start T-HEAD DebugServer and then pause T-HEAD DebugServer. Choose "Setting->Multicore Threads". Then, start T-HEAD DebugServer again.

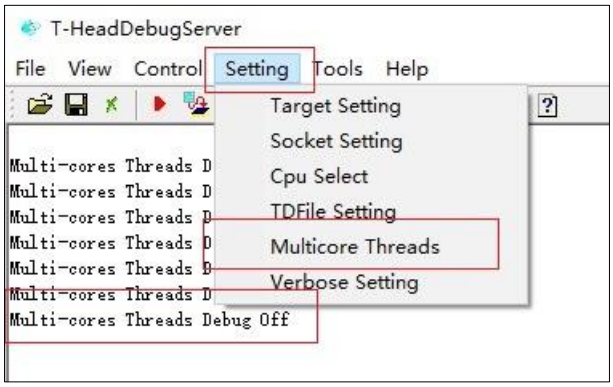


Figure 8-11-31 UI for setting the -no-multicore-threads mode in the UI

② Find the default.ini file in the directory where the T-HEAD DebugServer program is located, and change "MULTICORETHREADS=TRUE" to "MULTICORETHREADS=FALSE. Then, start T-HEAD DebugServer again.

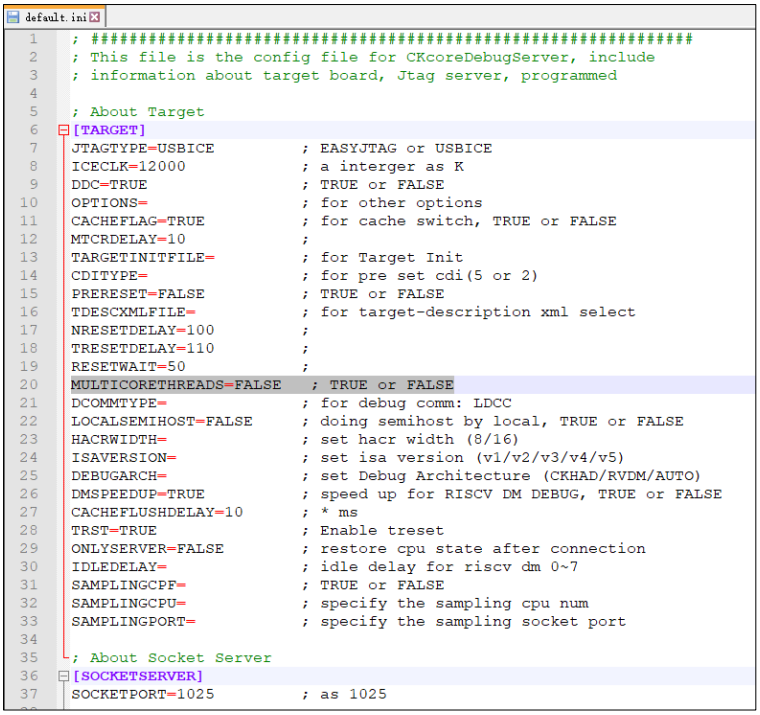


Figure 8-11-32 UIs for modifying default.ini to set the -no-multicore-threads mode

③ The following UI is displayed after T-HEAD DebugServer is started.

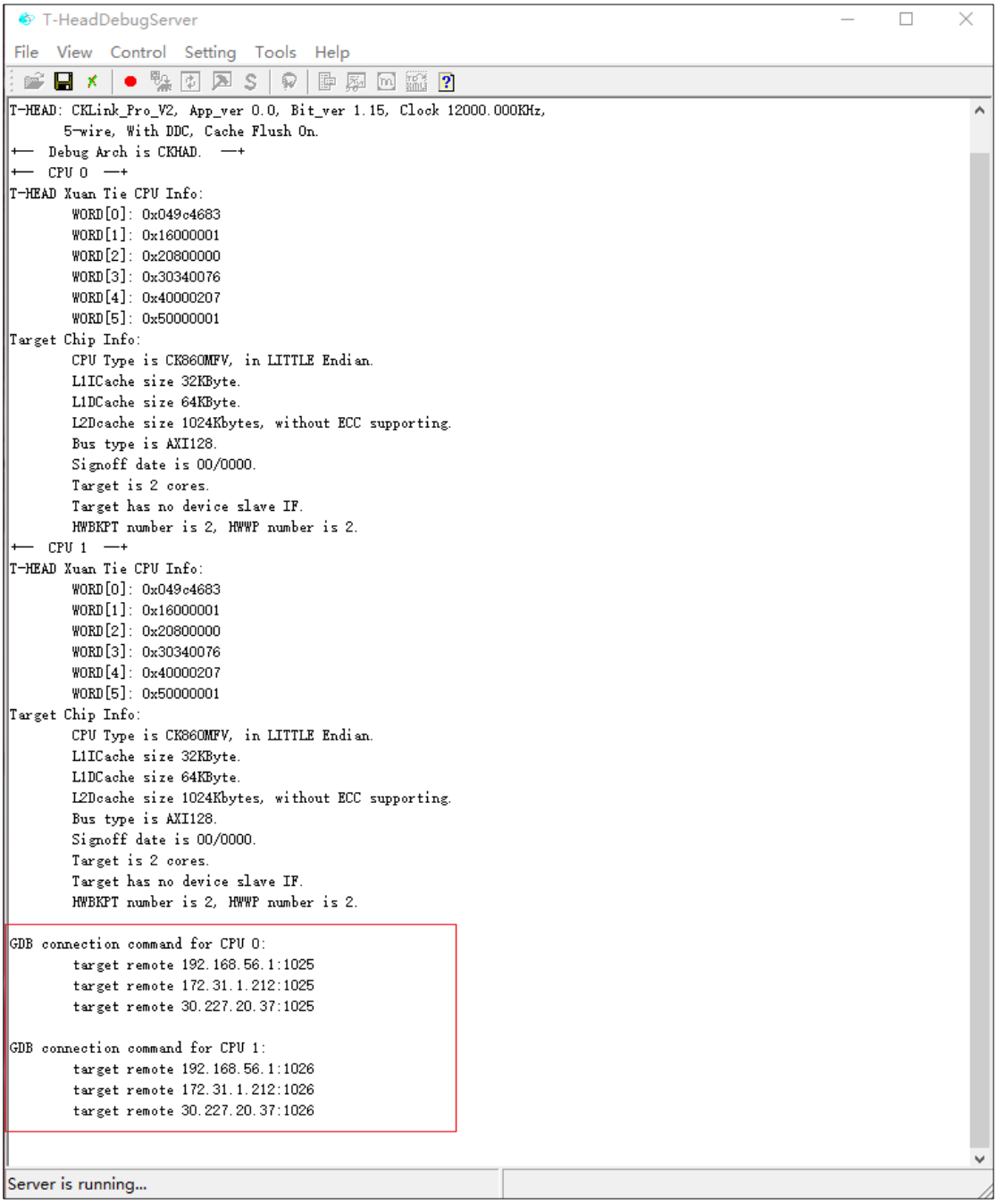


Figure 8-11-33 Output on T-HEAD DebugServer UI Edition after connection to C860MP in the multi-core multi-port mode

(2) Perform the following steps on T-HEAD DebugServer Console Edition for Windows or Linux:

Start T-HEAD DebugServer Console Edition by adding the -no-multicore-threads

parameter.

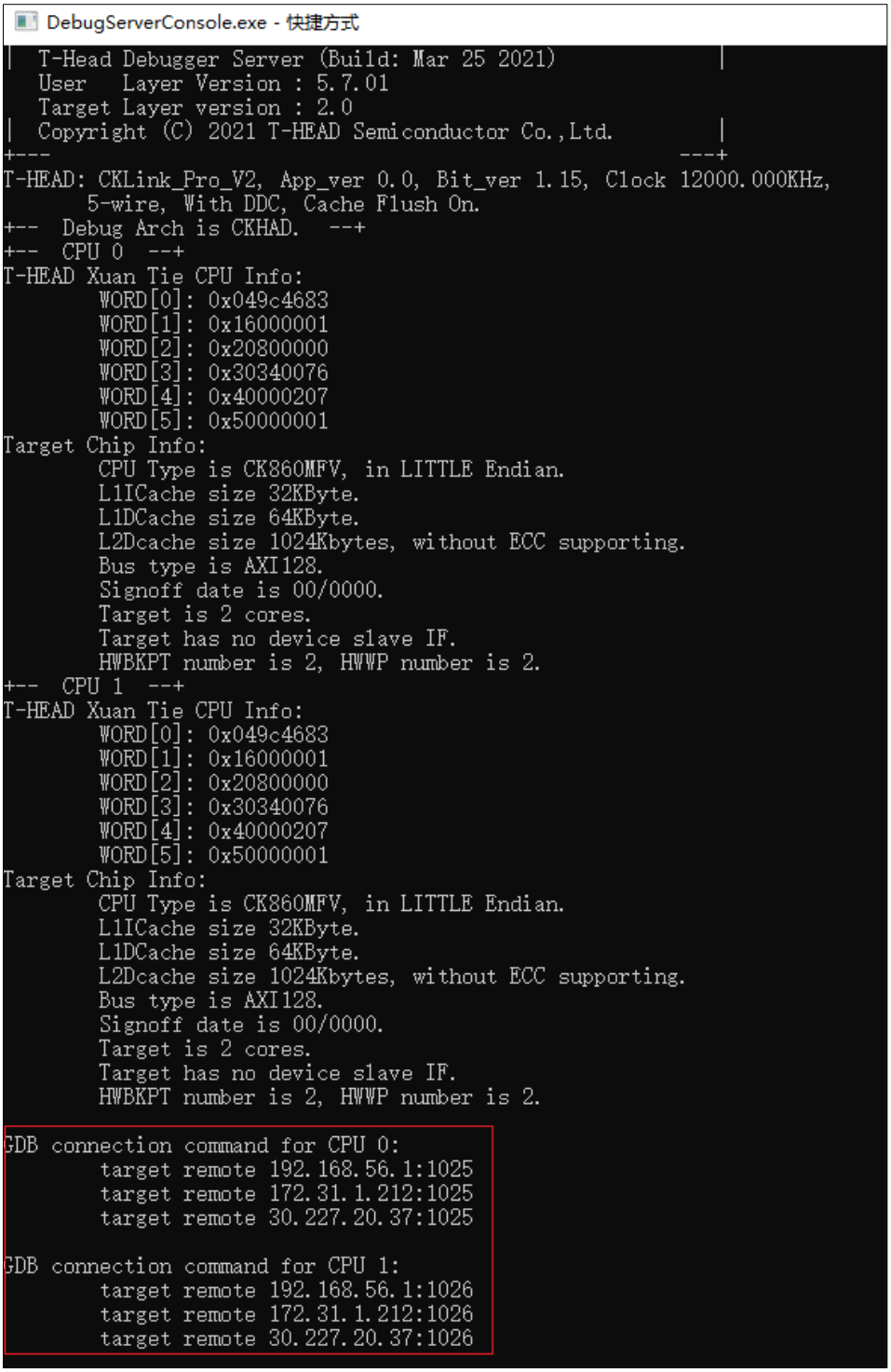


Figure 8-11-34 Output on T-HEAD DebugServer Console Edition after connection to C860MP in the multi-core multi-port mode (Same output for Windows and Linux)

5. Connect to T-Head GDB for debugging.

(1) The connection commands used to connect to CPU 0 and CPU 1 are already displayed on the UI of T-HEAD DebugServer. The IP addresses of the CPUs are the same. The port numbers are different and therefore are used for differentiation.

(2) When T-Head GDB is debugging, T-Head GDB is debugging a single core. This feature is the same as single-core debugging. You need only to know that all the cores are sharing the same memory.

9. Flash programming and flash breakpoints

Flash programming is a common operation in MCU debugging. Since addresses in the flash of most MCUs map to addresses in the CPU, MCU codes can be executed on the flash. Therefore, the flash programming feature is introduced to T-HEAD DebugServer.

9.1. Flash programming principles

The flash programming principles of T-HEAD DebugServer is consistent with those of the CDK.

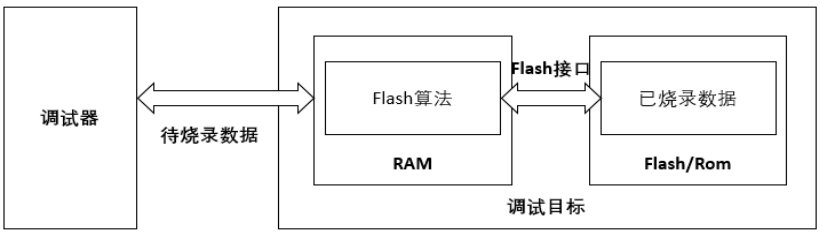


Figure 11-35 Flash programming on the debugger

As shown in the figure above, the debugger uses the flash algorithm file in the RAM area to erase and burn the flash or ROM area. For details, see chapters 5.2, 5.3, and 5.4 in the following link:

https://occ.t-head.cn/development/series/video?spm=a2cl5.25410618.0.0.4a53180f137C4G&id=386477535151142092 8&type=kind&softPlatformType=4#sticky

9.1.1.Algorithm file requirements

Since the hardware scene cannot be destroyed when T-HEAD DebugServer downloads and sets flash breakpoints, you need to know:

1. Registers that will be destroyed by the algorithm file. The registers include GPRS, PC, and the control register containing the interrupt enable control bit. These registers have no requirements on the T-HEAD algorithm file.

2. Memories that will be destroyed by the algorithm file. The memories include the flash algorithm program code execution memory and stack memory. The code execution memory can be obtained, but the stack memory requires that the version of the algorithm file be greater than or equal to 6.

It is required that the algorithm file use the CDK template and the version be greater than or equal to 6. Basic requirements are as follows:

1. The stack usage range is included in the .bss section.
2. __bkpt_label is a software breakpoint instruction.
a. 900 series CPU:

```
__bkpt_label:
        ebreak
        ret
```

b. 800 series CPU:

```
__bkpt_label:
        bkpt
        rts
```

3. __continue_label exists.
a. 900 series CPU:

```
__continue_label:
        mv      a0, a0
        ret
```

b. 800 series CPU:

```
        __continue_label:
        mov     a0, a0
        rts
```

For more information, see the CDK flash template project.

9.1.2.Flash operation commands supported by the command line

The functions related to flash operations are implemented in the command line of T-HEAD DebugServer. The commands integrated into T-HEAD DebugServer are as follows:

- flash
- flash info
- flash erase
- flash program
- flash dump
- flashbp-clear

flash: is used to specify the flash algorithm file. The parameter is as follows:

- -al/--algorithm file: specifies the flash algorithm file.

flash info: is used to view information about the current flash algorithm file. No parameters are involved.

flash program: is used to perform flash programming. The parameters are as follows:

- -al/--algorithm file: specifies the flash algorithm file used for flash programming. By default, the flash algorithm file specified when T-HEAD DebugServer is started or the algorithm file specified by flash -al is used.
- -f/--file: specifies the file to be flashed.
- -b/--binary: If a .bin file is to be flashed, this parameter needs to be specified.
- -a/--address ADDR: If a .bin file is to be flashed, this parameter specifies the starting address of flash programming.
- -v/--verify: checks whether the flash programming is successful.

flash erase: is used to erase the flash. The parameters are as follows:

- -al/--algorithm file: specifies the flash algorithm file used for erasing. By default, the flash algorithm file specified when T-HEAD DebugServer is started or the algorithm file specified by flash -al is used.
- -c/--chip: specifies chip erasing.
- -a/--address ADDR without -c/--chip: specifies the starting address of erasing and writing when chip erasing is not performed.
- -s/--size LENGTH without -c/--chip: specifies the length of erasing when chip erasing is not performed.

flash dump: is used to dump the flash content to a file. The parameters are as follows:

- -al/--algorithm file: specifies the flash algorithm file used for dumping. By default, the flash algorithm file specified when T-HEAD DebugServer is started or the algorithm file specified by flash -al is used.
- -o/--output: specifies the name of the file to which the flash content is dumped.
- -b/--binary: indicates that the dumped file is a .bin file.
- -h/--hex: indicates that the dumped file is a .hex file.
- -a/--address ADDR: specifies the starting address of dumping.
- -s/--size LENGTH: specifies the length of dumping.

flashbp-clear: is used to clear the flash breakpoints that have been inserted or that are not deleted in the flash memory. In the debugging process, instruction simulation will be performed to speed up the process, or the flash breakpoints will be deleted only at the appropriate time to reduce the number of operations on the flash. In this way, flash breakpoints that have been inserted or that are not deleted still exist in the current flash memory in the debugging process. This command will delete these breakpoints and restore the original values in the flash.

EXAMPLE:

flash -al ch2201_eFlash.elf

flash info

flash program -f download.elf

flash program -f download.hex

flash program -f download.hex -v

flash program -f download.bin -a 0x10000000 -b

flash program -f download.elf -al ch2201_eFlash.elf

flash program -f download.hex -al ch2201_eFlash.elf

flash program -f download.bin -a 0x10000000 -b -al ch2201_eFlash.elf

flash erase -c

flash erase -a 0x10000000 -s 0x2000

flash erase -c -al ch2201_eFlash.elf

flash erase -a 0x10000000 -s 0x2000 -al ch2201_eFlash.elf

flash dump -o dump.bin -b -a 0x10000000 -s 0x2000

flash dump -o dump.hex -h -a 0x10000000 -s 0x2000

flash dump -o dump.bin -b -a 0x10000000 -s 0x2000 -al ch2201_eFlash.elf

flash dump -o dump.hex -h -a 0x10000000 -s 0x2000 -al ch2201_eFlash.elf

9.2. Flash breakpoints

Breakpoints can be divided into three categories:

- Software breakpoints
- Hardware breakpoints
- Data observation points

Software breakpoints: The instruction set of the CPU contains a breakpoint instruction. When the instruction is executed on the CPU, a debugging exception is generated, and the CPU enters the debug mode. In this case, the CPU stops running. With the characteristics of this instruction, the debugger replaces the original instruction at the breakpoint location through the load or store instruction, so that the breakpoint takes effect.

Hardware breakpoints: The CPU contains an address comparator unit. When a hardware breakpoint is set, the address PC of the configuration comparator is compared with the breakpoint address. If the address PC matches the breakpoint address, the CPU enters the debug mode and the hardware breakpoint takes effect. Since the comparator occupies a large number of resources, the number of hardware breakpoint comparators in the MCU is limited.

Data observation points: The data address comparator in the CPU is similar to the hardware breakpoint comparator. The observation point comparator compares the load or store address and the data observation point address. When the addresses are matched, the CPU enters the debug mode and the hardware breakpoint takes effect. Generally speaking, the data address comparator is the same unit as the address comparator on the hardware.

In the MCU field, codes are stored in the flash. When a software breakpoint is made to codes, the instruction code is rewritten through the load or store instruction because the software breakpoint is used. Since codes are stored on the flash, the instruction code cannot be rewritten through the load or store instruction. In this case, only hardware breakpoints can be used. However, due to the limitation of the number of hardware breakpoints, the debugging work of developers is limited.

To solve the proceeding problems, flash breakpoints are introduced. Similar to software breakpoints, flash breakpoints also use the breakpoint instruction to enable the CPU to enter the debug mode. The difference is that software breakpoints use the load or store instruction to rewrite the instruction code, while flash breakpoints erase the flash to rewrite the instruction code.

9.2.1.Working principles

As mentioned above, the flash breakpoint provides a method to rewrite the instruction code by erasing the flash. Therefore, T-HEAD DebugServer will record the breakpoint information used by the user. When a new breakpoint needs to be inserted, T-HEAD DebugServer erases and burns the flash, and replaces the original instruction with the bkpt/ebreak instruction. For more information about flash programming principles, see 9.1Flash programming principles.

When the user needs to execute a command at the breakpoint, T-HEAD DebugServer integrates the simulation execution unit to simulate some of the instructions. In case some instructions cannot be simulated, the instruction codes will be written in the sp position and executed by the CPU.

9.2.2.Efficiency of breakpoints

In the debugging process, the flash breakpoints need to perform erasing and writing operations on the on-chip flash, resulting in increased time cost when compared with software breakpoints. To reduce the number of operations on the flash, a series of optimizations are made in T-HEAD DebugServer, including but not limited to the combined use of hardware breakpoints and instruction simulation.

To ensure data consistency, T-HEAD DebugServer saves the memory area used by the algorithm file before burning flash breakpoints. To improve efficiency, we recommend that the code size and data size of the flash algorithm file be as small as possible.

10. Vendor ICE support

In the process of business development, the customers are capable of developing and designing links. It is required that T-HEAD DebugServer allow the customers to design and develop links. Therefore, the link design and development function is added to T-HEAD DebugServer.

The function is implemented as follows:

Multiple Vendor directories are stored in the links directory in the installation directory. Each directory stores the link library file of a customer. The library file is required to provide some interfaces for porting so that T-HEAD DebugServer can complete debugging.

For porting interfaces, see Table 11-11 Link porting interface list. Details can also be found in Include/link.hof the sample program.

Interface	Interface prototype	Description
link_init	Int link_init (dbg_server_cfg_t *cfg);	Initialize a link. This interface is optional.
link_open	void*link_open(dbg_server_cfg_t *cfg, void *unique);	Open the link device.
link_close	void link_close (void *handle);	Close the link device.
link_config	int link_config (void *handle, enum LINK_CONFIG_KEY key, unsigned int value);	Configure the link, including frequency and reset time. Specific configuration items are optional.
link_upgrade	int link_upgrade (void *handle, const char *path);	Upgrade the firmware. This interface is optional.
link_memory_read	int link_memory_read (void *handle, uint64_t addr, int xlen, uint8_t *buff, int length, int mode);	Read the target board memory.
link_memory_write	int link_memory_write (void *handle, uint64_t addr, int xlen, uint8_t *buff, int length, int mode);	Write the target board memory.
link_register_read	int link_register_read (void *handle, int regno, uint8_t *buff, int nbyte);	Read the general purpose register of the target board.
link_register_write	int link_register_write (void *, int regno, uint8_t *buff, int nbyte);	Write the general purpose register of the target board.
link_jtag_operator	int link_jtag_operator (void *handle, int ir_len, unsigned char *ir, int dr_len, unsigned char *dr_r, unsigned char *dr_w, int read);	Perform a JTAG operation. This interface is optional.
link_gpio_operator	int link_gpio_operator (void *, int gpio_out, int *gpio_in, int gpio_eo, int gpio_mode);	Perform a GPIO operation. This interface is optional.
link_show_info	int link_show_info (void *, dbg_server_cfg_t *cfg, void (*func)(const char *, ...));	Display link information. This interface is optional.

link_reset	int link_reset (void *handle, int hard);	Reset the link.
link_get_device_list	int link_get_device_list (struct link_dev *dev, int *count);	Obtain the link list.
THE_NAME_OF_LINK	const char * THE_NAME_OF_LINK (void);	Identify the link name. This interface, as the DLL identification interface, is mandatory.

Table 11-11 Link porting interface list

11. Example project

Customers sometimes need to perform some operations to control or read and write chip information in different application scenarios. The following provides an example project to inform customers how to operate on the chip over target board interfaces.

There is an "Example" directory in the installation directory of the T-Head T-HEAD DebugServer tool, and the operations are different in the Linux and Windows systems.

In the Linux system, a Makefile compiling project is provided. The dependency tools are Make, GCC, and G++.

In the Windows system, a VS project is provided. The dependency tool is Virtual Studio.

Open the project, we can see some operations in the source file. The operations include:

1) Initialization process and connection to the target board
2) Register read and write operations
3) Memory read and write operations
4) Run or stop
5) Disconnects the connection.

For details, open the project.

12. Common problems and solutions

Table 11-1 Common problems and solutions

Problem	Error Message	Solution
ICE connection failed	ERROR: No C-SKY ICE connected to Your PC or your C-SKY ICE driver not installed correctly! Input enter to exit...	Reconnect the ICE and install the ICE driver.
Target board connection failed	ERROR: Fail to enter debug mode! Error: Can't enter debug mode, please check the target board physical link. Input enter to exit...	Reconnect the ICE to the target board, and ensure that the target board is powered on.
Port binding failed	ERROR: Fail to bind socke port 1025, please change another one. ERROR: Fail to create socket server. Input enter to exit...	Open the port settings tab, reselect the port number, and connect the port again.

Version upgrade failure	WARNING: ICE Upgrading ignored, it may cause function lose or error!!! ERROR: DebugServer can't implement ICE config! Input enter to exit...	Restart T-HEAD DebugServer and connect it to the ICE. When an ICE upgrade prompt appears, select 'y'. After the upgrade is successful, unplug and then replug the ICE, and reconnect T-HEAD DebugServer to the ICE.
Version compatibility issue between ICE and T-HEAD DebugServer	ERROR: You ICE's version is newer than Your DebugServer, Please update Your DebugServer! ERROR: DebugServer can't implement ICE config! Input enter to exit...	This issue occurs on the cklink_lite ICE. This issue will occur if T-HEAD DebugServer of an earlier version is connected. In this case, download the latest T-HEAD DebugServer from the T-Head official website.

Debug architecture connection issue	The debug architecture is automatically detected when the server is connected. If the architecture is not detected, the "-arch" parameter needs to be specified.	For the console version T-HEAD DebugServer, add the startup parameter "-arch riscv". For the UI version T-HEAD DebugServer, choose Setting > Target Setting and select RISCV DM for Debug Arch Select.