# T-Head Debugger Server

# User Guide

**Revision** 5.18

**Security** Pubilc

### 平头哥半导体有限公司 T-Head Semiconductor Co.,LTD

网址： https://t-head.cn

# 目录

# 图表目录

# 1. 引言

本用户手册主要针对 T-Head Debugger Server 进行功能使用介绍，本软件支持 Windows 和 Linux 平台，用户手册则分 Windows 篇和 Linux 篇分别做介绍。

## 1.1. 定义

表格 1-1 简写定义列表

| 名词 | 含义 |
|---|---|
| ddc | 下载直通通道，可大幅提高数据下载速度 |
| thserver | T-Head Debugger Server 简写，调试代理服务程序 |
| ICE | 在线仿真器（In Circuit Emulator ），在这里指的是 CKLINK |
| HAD | T-Head 私有硬件调试模块 |
| DTM&DM | RISC-V 开源设计的硬件调试模块 |

## 1.2. 功能简述

Thserver 接收 T-Head GDB 发送的调试原语操作命令，按照 JTAG 协议发送相应命令到调硬件试接口（支持 T-HEAD HAD 模块和 RISC-V Debug Spec V0.13.2 标准 DTM&DM 模块），并控制调试执行指令，获取调试数据返回给 T-Head GDB。T-Head GDB 与 thserver 采用 socket 方式进行通信，调试器 GDB 和调试代理服务器可以运行在不同的主机上。Thserver 和目标机之间通过 ICE 按照 JTAG 协议进行通信。

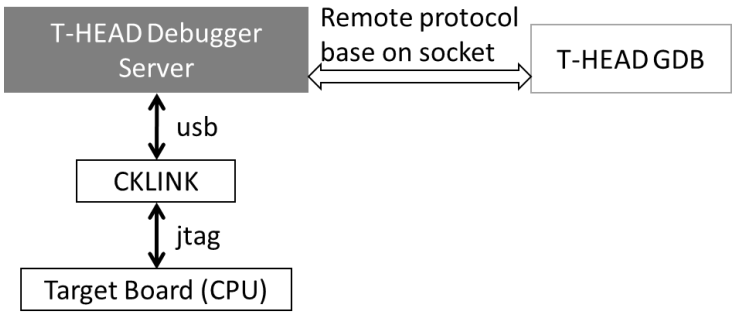



图 1-1 T-HeadDebugServer-Server 通信方式

# 2. Windows 篇

Thserver 在 Windows 主机系统下支持图形用户界面（GUI 版本）和命令行（console 版本）两种运行方式。

## 2.1. thserver 及 ICE 驱动安装

ICE 的驱动打包在 thserver 的安装包中，安装 thserver 时，勾选 ICE Driver 选项，安装时会将驱动文件拷贝到系统目录下。插上 ICE 设备后，Windows 会自动为 ICE 安装驱动。

### 2.1.1. 安装包获取

从 T-Head 公司的 OCC 平台 https://xuantie.t-head.cn/community/download 或技术支持处获取安装包，安装包中包含：Windows 平台的 T-Head-DebugServer-windows*.zip 压缩文件和 Linux 平台的两个 T-Head-DebugSever-linux-*.sh 安装文件（分别对应 32 位和 64 位系统）。

### 2.1.2. thserver 安装步骤

1.解压 T-Head-DebugServer-windows*.zip 文件，双击运行解压出的 Setup.exe 将弹出 thserver 安装界面，点击 next。

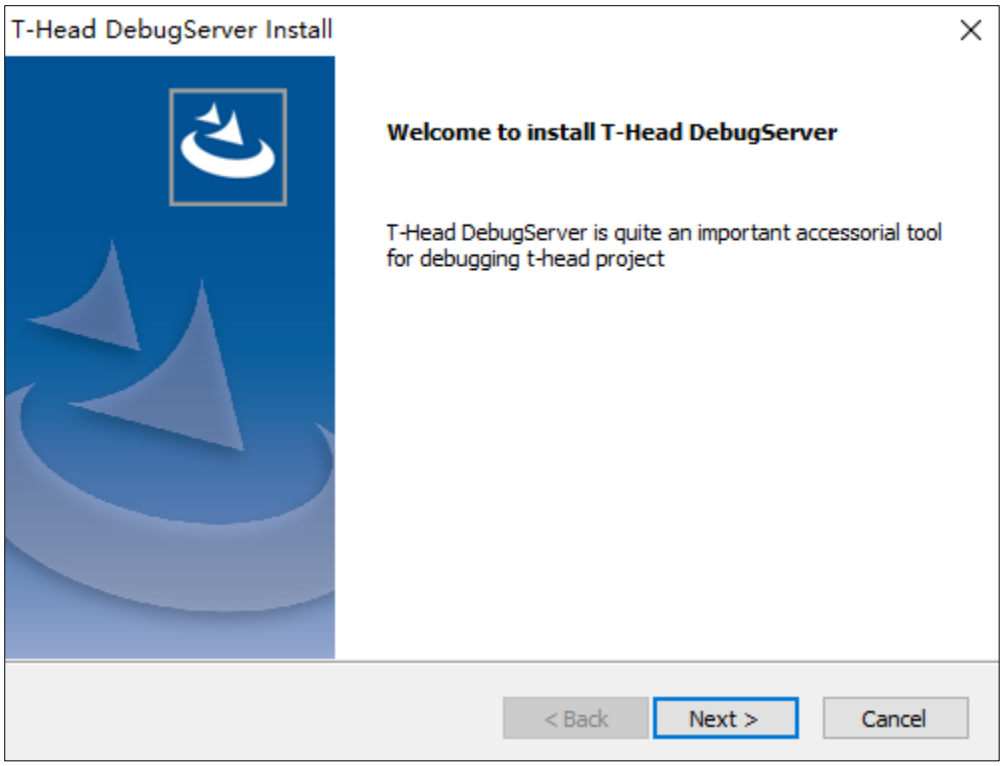



图 2-1 thserver 安装步骤-1

2.请认真查看 thserver 的软件使用协议，确认后点击 yes 继续。

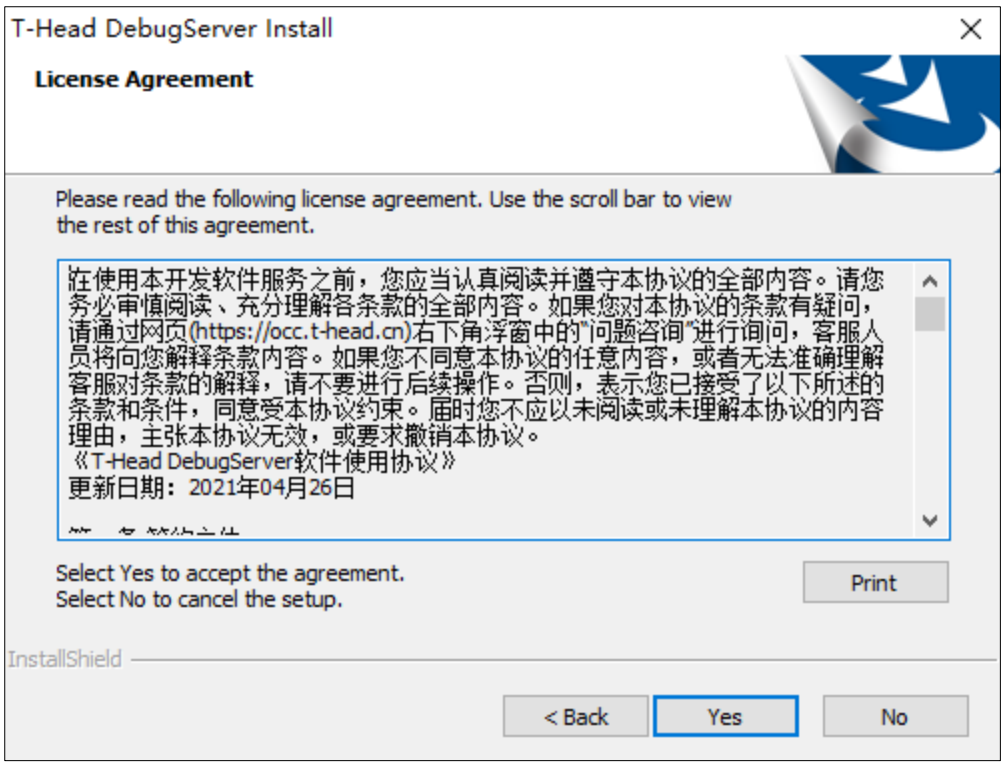


图 2-2 thserver 安装步骤-2

3.输入用户名和企业名称，选择用户，点击 next。

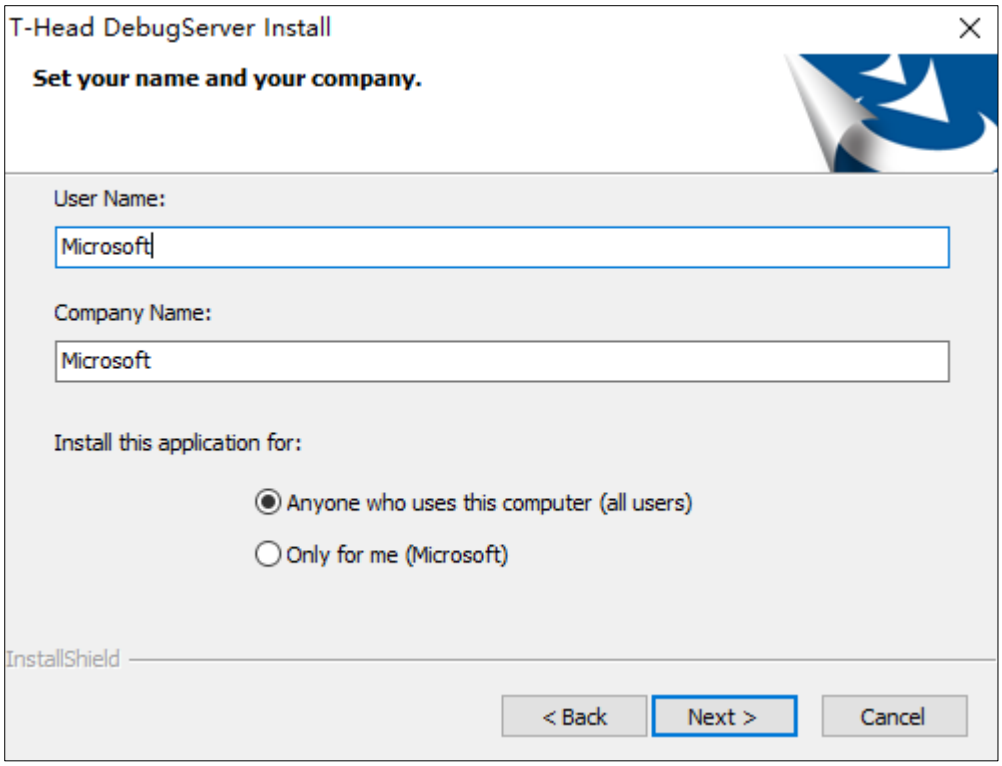



图 2-3 thserver 安装步骤-3

4.选择安装目录。

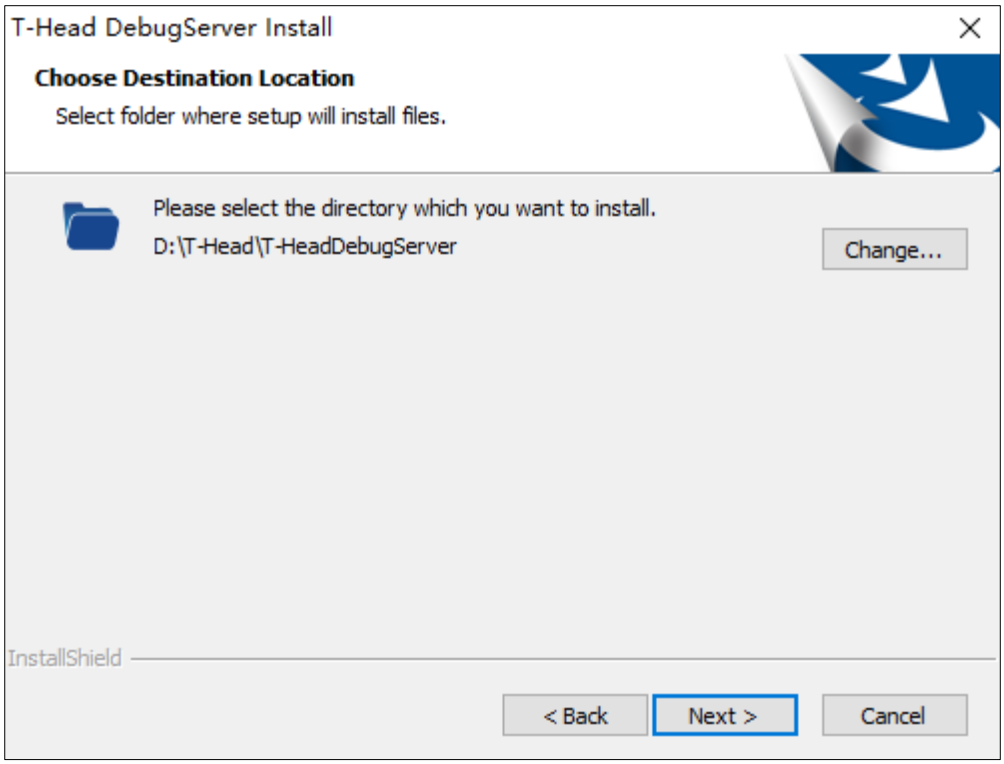



图 2-4 thserver 安装步骤-4

5. 选择安装内容，其中： Main App 是 T-Head 调试代理程序；ICE Driver 是 ICE 设备驱动安装；Tutorial 是用户手册。建议全选，点击 next。

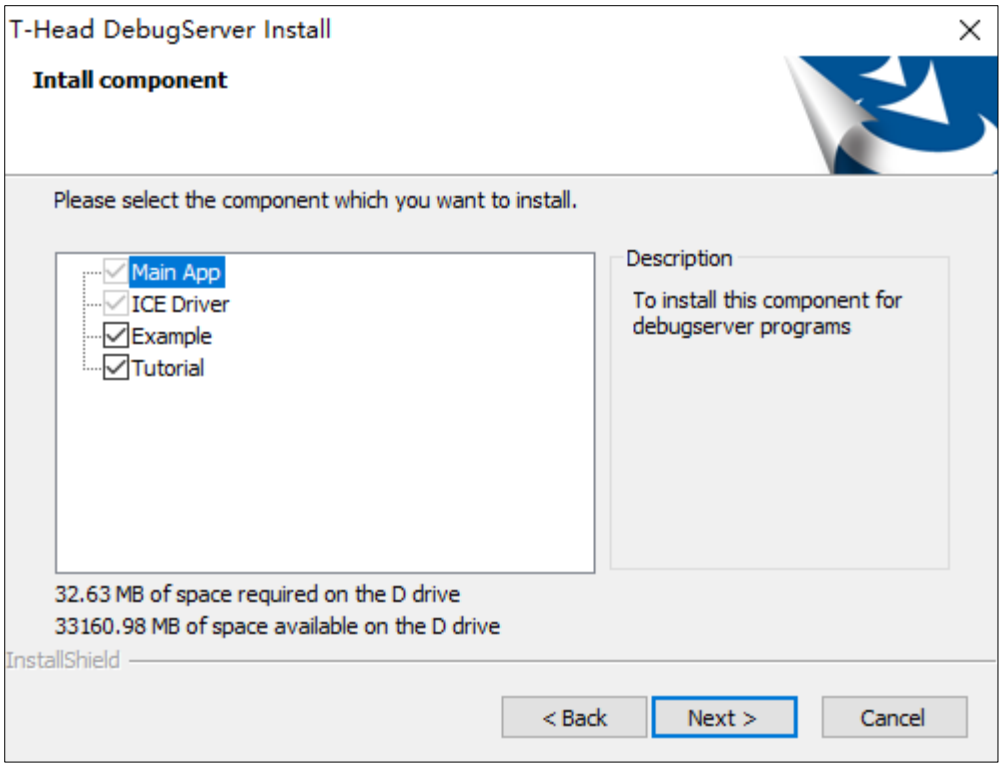



图 2-5 thserver 安装步骤-5

6.该页面将显示用户信息及安装目录，确认无误后点击 Next 开始安装。

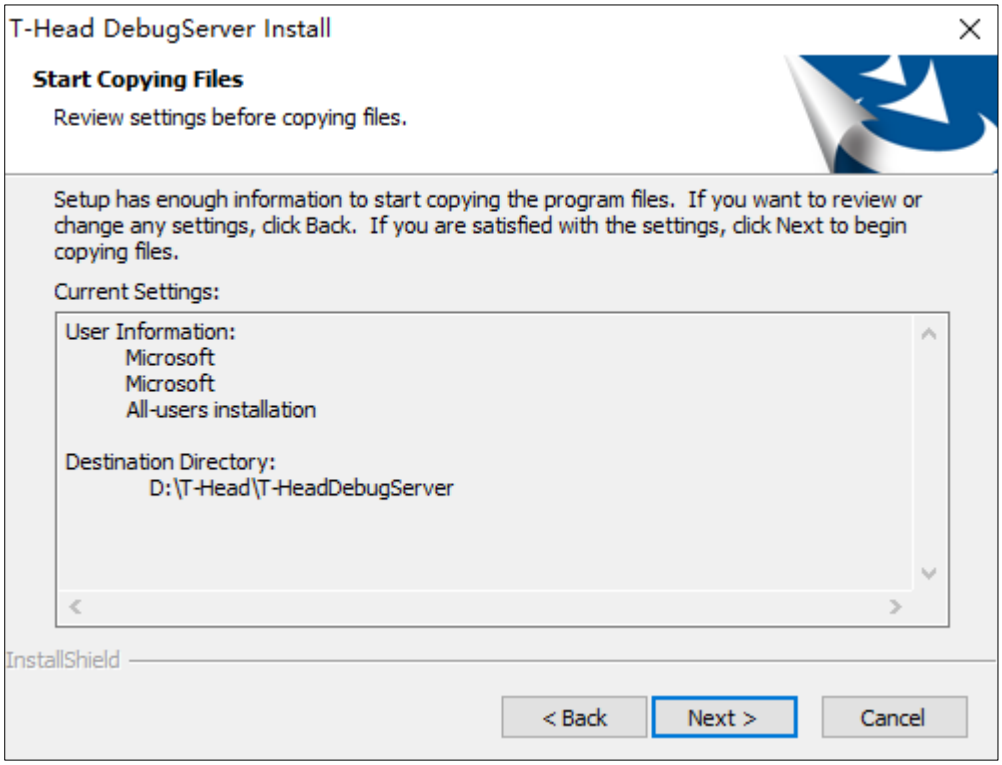



图 2-6 thserver 安装步骤-6

7.开始安装。

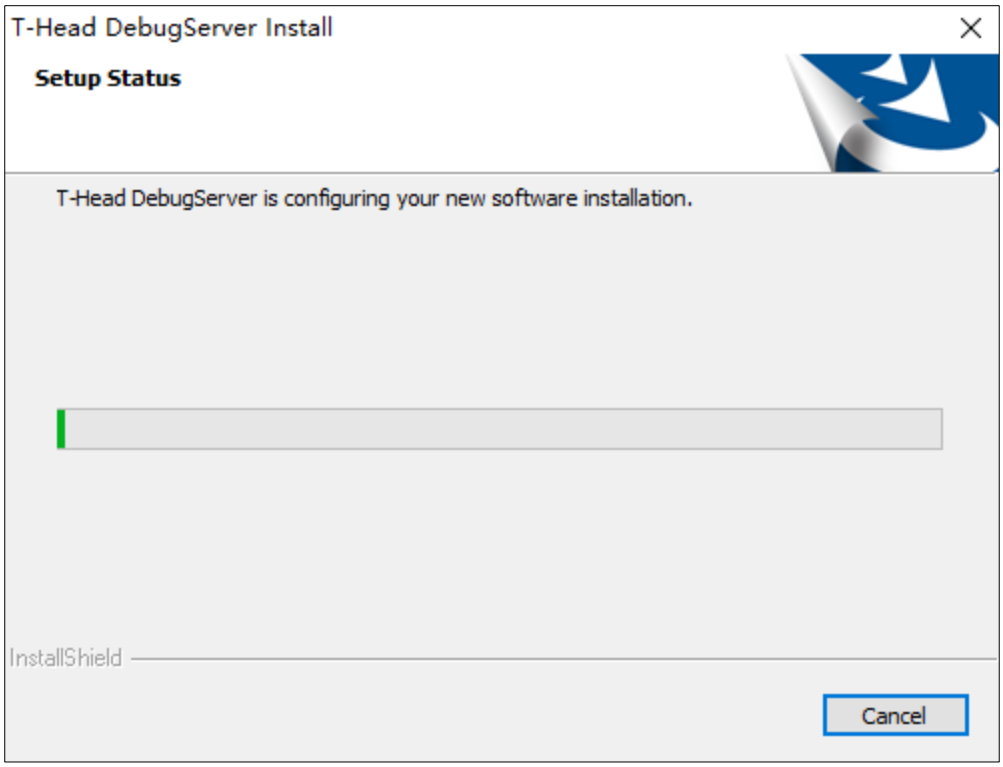



图 2-7 thserver 安装步骤-7

8.若您的 PC 上已经安装过 ICE 驱动，会提示是否进行驱动更新，建议选择“是”，将为您的 ICE 更新最新的驱动。

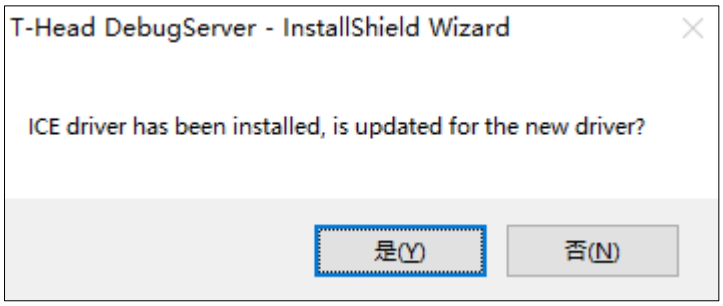



图 2-8 驱动更新选择对话框

9.thserver 安装完成，点击 finish。

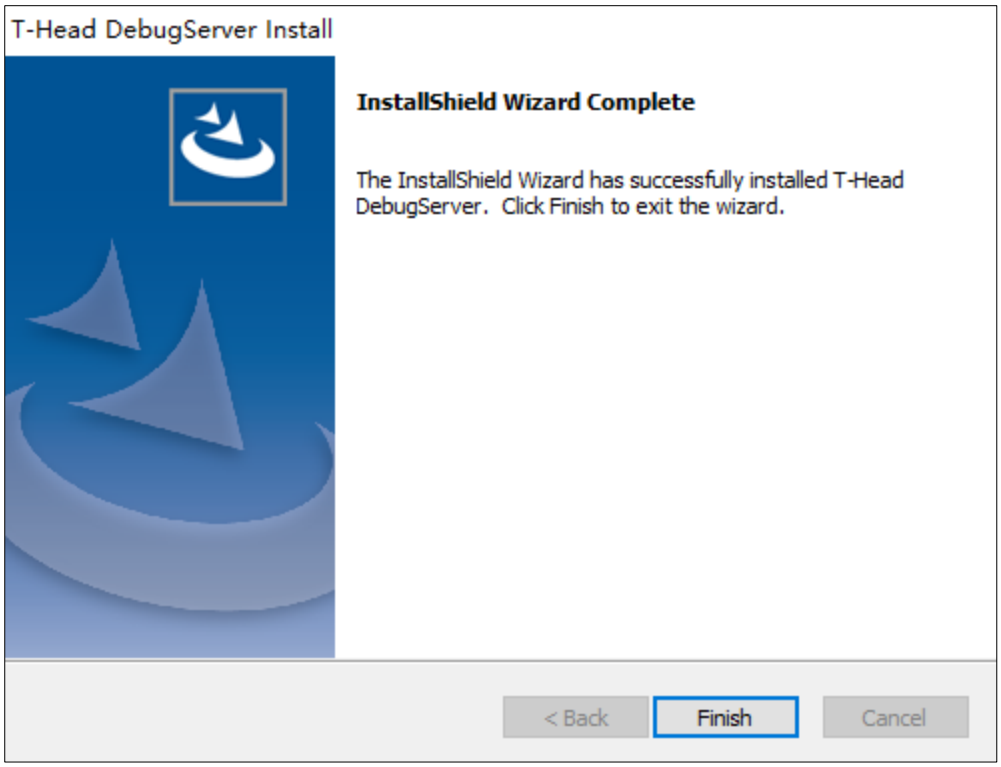
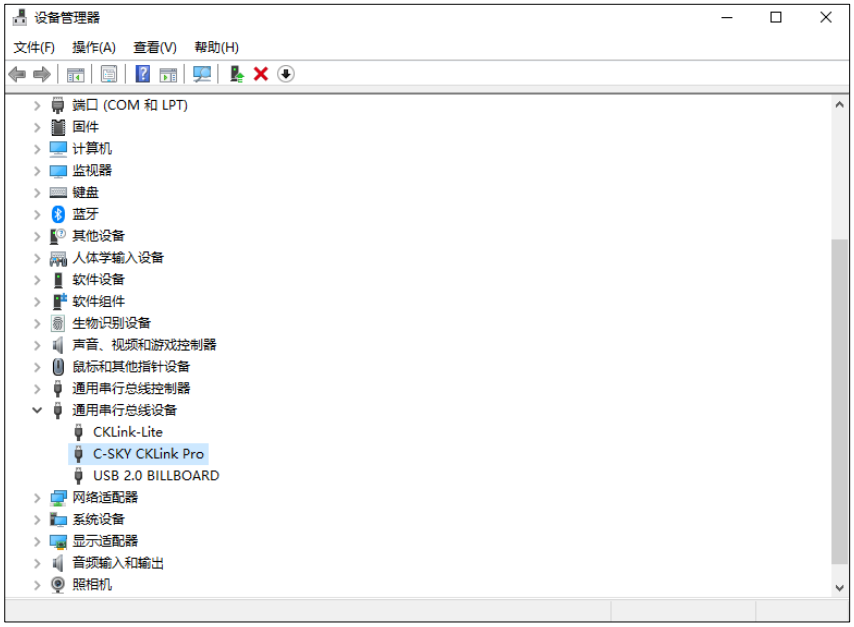



图 2-9 thserver 安装步骤-8

### 2.1.3. ICE 驱动安装步骤

1. 在“2.1.2 节 thserver 安装步骤”中，如果勾选了 ICE Driver 选项。那么 thserver 安装过程中会将驱动相关文件拷贝到系统的目录下。

2. 将您的 ICE 设备重新连接至 PC，则可以在电脑的设备管理器中看到类似下图的设备（CKLink Lite 与 CKLink Pro）：




## 2.2. 运行环境

Thserver 运行要求 windows2000 及以上的版本，输入输出设备要求具有 USB 接口。Thserver 可配合 T-Head GDB 使用，支持 RISC-V GDB 和 CSKY GDB。

## 2.3. console 版 thserver 使用说明

### 2.3.1. 运行参数

1. Console 版 thserver 通过启动参数来实现运行时的配置。常用输入参数见下表。

表格 2-1 thserver 运行参数列表

| 用户接口 | 定义 | 默认值 |
| --- | --- | --- |
| -setclk iceclk | 设定 ICE Jtag 时钟频率，默认单位 Mhz，支持 Khz 为单位的数据格式，如设置-setclk 12000k 表示频率设置为 12M。每种类型 ICE 有固定的上限频率，具体如下：<br>CKLINK-PRO 24Mhz<br>CKLINK-V1 24Mhz<br>CKLINK-Lite 2500khz<br>最低均为 10khz | 默认值为 12，即 ICE 频率为 12M。 |

| | | |
|---|---|---|
| -port port | 设定起始的 socket 通信端口。 | 1025。 |
| -prereset | 指定在获取 ICE 控制后发起 nreset 操作。 | 默认不执行。 |
| -noddc | 不使用数据下载直通通道，仅 T-HEAD HAD 有效。 | 默认使用。 |
| -nocacheflush | Thserver 在连接过程中，执行单步前和退出调试模式前不刷 cache。 | 默认刷 cache。 |
| -setcdi 2/5/cJTAG2 | 参数可以是 2/5/cJTAG2，即连接 TARGET 之前设置 ICE 的工作方式，2 线,5 线或 cJTAG OSCAN1 的 2 线模式。 | 默认不执行。 |
| -mtcrdelay/delay | 设置 ICE 写完 CPU 控制寄存器后等待的时间。 | 默认为 1ms。 |
| -targetinit filepath | 在获取 ICE 后执行的目标初始化脚本，脚本类型为 GPIO 或 JTAG 脚本，执行脚本后继续 Server 启动流程。 | 无。 |
| -scr filepath | 指定执行 thserver 操作脚本(gpio 或 jtag 脚本)，这个参数指定后，server 功能不再开启，运行完脚本后程序退出。 | 无。 |
| -configpath/-configfilepath filepath | 指定配置 ICE 时，获取 ICE 固件的路径。 | 默认为 thserver 可执行程序所在的目录。 |
| -tdescfile filepath | 为 gdb 指定描述目标板寄存器的 xml 文件。 | 默认根据 CPUID 来指定一个对应的 tdesc-xml 文件。 |
| -{no-}trst | 在执行 nreset 命令时是否执行 treset。 | 默认执行。 |
| -nrstdelay | 设置 Nreset 延时，确保 ICE 可以生成稳定的硬件复位（nreset）信号，单位 10us。 | 默认为 1ms。 |
| -trstdelay | 设置 Treset 延时，确保 ICE 可以产生稳定的复位信号，用于复位 HAD 状态机，单位 10us。 | 默认为 1.1ms。 |
| -ndmrstdelay | 设置产生 RISC-V DM DMCONTROL.ndmreset 信号的时间长度。 | 无。 |
| -hartrstdelay | 设置产生 RISC-V DM DMCONTROL.hartreset 信号的时间长度。 | 无。 |
| -rstwait | 设置延时，确保在目标板收到复位信号后，等待目标板复位流程执行结束。 | 默认为 50ms。 |
| -no-multicore-threads | 设置连接多核调试时，将每个核看作一个独立的个体，为每个核开启一个调试端口， | 默认为假。thserver 只会开 |

| | | |
|---|---|---|
| | GDB 连接指定端口即为调试该端口对应的核。端口与核一一对应。 | 启一个调试端口，同时封装多核为多个线程信息发送给 GDB。GDB 调试线程，则在调试线程对应的核。 |
| -local-semi/-ls | 程序发生的 Semihost 请求由 thserver 执行。 | 默认情况下，Semihosting 的请求在 GDB 中完成，可通过[-local-semi/-ls]选项重设为在 thserver 中执行。<br>注意： 在 Windows 中加-local-semi/-ls 时，不支持 isatty 和 system 操作。 |
| -dcomm=ldcc | 启动 jtag 输入输出通道（需要硬件支持）。 | 默认不开启。 |
| -disable-cmdline | 不开启命令行功能。 | 默认开启。 |
| -set-isa_version v1/v2/v3/v4/v5 | 设置 ICE 的 HAD Version 版本。 | 默认由 thserver 自动设置。 |
| -set-hacr_width 8/16 | 指定设置连接目标板时 HACR 的宽度。 | 默认为 thserver 自动探测。 |
| -no-cpuid-check | 连接 xuantie 800 系列 CPU 目标板过程中不读取 CPUID 信息。 | 默认读取并检查。 |
| -cacheflush-delay time | 设置 thserver 中刷 cache 行为的延时，确保刷 cache 行为正常结束。 | 默认 100ms。 |
| --debug usb/connect/target/remote/djp/sys/flash/all | usb:记录 thserver 与 ICE 之前交互的协议包。<br>connect:描述连接开发板的细致过程。<br>target:记录 target 抽象层函数调用信息。<br>remote:记录 remote 协议交互信息。<br>djp:记录 djp 协议交互信息。<br>sys:记录 thserver 程序主循环信息。 | 默认没有 log 信息输出。 |

| | | |
|---|---|---|
| | flash:记录 flash 烧录和 flash 断点操作信息。<br>all:打印以上所有 Log 信息 。 | |
| -arch thead(csky)/riscv/auto | 选择连接的调试架构，thead 指代 T-Head HAD 调试架构，riscv 指代 RISCV DM 调试架构。 | 默认为 auto，连接过程自动探测。 |
| -cmd-script file | 执行 thserver 命令行脚本，即可以将 cmdline 处的命令写成文件，在 thserver 连接上目标板之后执行。 | 无。 |
| -list-ice | 罗列当前连接在 PC 上的 ICE。 | 无。 |
| -select-ice serial_number | 根据-list-ice 罗列的 ICE 列表，指定 ICE 进行连接。注意：如果 serial_number 中有空格，需要给 serial_number 加双信号。 | 无。 |
| -list-vendor | 显示支持的 ICE 设备产商，如 CKLink 。 | 无。 |
| -select-vendor vendor_name | 选择 ICE 的 Vendor。 | 默认选择 CK-LINK。 |
| -skip-enter | Console 版本 thserver 程序退出时不再需要输出 enter，直接退出。 | 默认需要用户输入 enter。 |
| -set-logfile filepath | 报错 DebugServer 运行的 log 到 filepath。 | 无。 |
| -idle idle | 设置 JTAG 的 IDLE 状态的持续周期数。 | 800 系列 CPU 为 0;900 系列 CPU 根据 DTM 内的配置自动设置。 |
| -sampling cpu_n port freq | 启动 Sampling 的 socket，对象参数:<br>Cpu_n：对 cpu_n 进行采样<br>Port：端口号，用于 CPF 的连接<br>Freq：采样频率 | 无。 |
| -sampling-type pcfifo-link/pcfifo-host | 设置 PC 采样的方式，pcfifo-link 或者 pcfifo-host，<br>pcfifo-link： 在 cklink 中进行采样<br>pcfifo-host：主机 DebugServer 进行采样 | 默认为 pcfifo-link。 |
| -flash-algorithm filepath | 指定 flash 算法文件，用于 flash 烧录和设置 flash 断点。 | 无。 |
| -flash-timeout time | 指定 flash 算法文件执行一次 flash 操作后函数调用后运行至__bkpt_label 的超时时间。 | 60 秒。 |
| -disable-flashbp | 关闭 flash 断点功能。 | 使能。 |

| | | |
|---|---|---|
| -disable-simbp | 关闭 flash 断点中的指令模拟功能。 | 使能。 |
| -abscmd-busy-delay time | 指定执行抽象命令发生 ABSTRACTCS.busy 为 1 后的等待时间。 | 无。 |
| -enter-debug-time time | 设置发起同步进调试的超时时间，time 单位为秒，最大值为 20。 | 0。 |
| -dmi-busy-adjust-times time | 设置当出现 DMI.op 为 busy 时再次尝试发起 DMI 操作的次数。每 2 次尝试将增加一个 IDLE Delay，设置范围 10~500。 | 10. |
| -print-sd-info addr size | 在 MPJTAG 中包含有 SD INFO 的设计中，可以通过该命令打印 SD INFO 接口中的信息。 | 默认不执行。 |
| -setresethaltreq-always-on | 在写入 RISC-V DM.dmcontrol 寄存器时，总是写 DM.dmcontrol.setresethaltreq 为真。<br>为真的情况下，CPU 测发起的 CPU 复位会被调试模块监测到。 | 默认配置 0。 |
| -no-hwbp | thserver 禁止使用硬件断点资源。 | 默认使用。 |
| -smp clusterx:clustery | T-Head 在使用 RVDM 设计中，默认一个 DM 内的所有核均为 SMP。当多个 DMs 构成一个 SMP 多核时，则需要加入-smp 参数。<br>clusterx:clustery 中的 x 和 y 是对应 DM 在 DM 链路中的编号，第一个 DM 的编号为 0，第二个为 1，之后的依次加 1。中间使用 ： 隔开<br><br>非 SMP 多 DMs 多核不需要配置该参数；-smp 可以出现多组，如果实际硬件也有多组的情况下；x 和 y 不重复出现在多个-smp 参数内，且不可出现不存在的 DM 编号。 | -smp cluster0:cluster1<br>-smp cluster0:cluster1:<br>cluster2:cluster3 |
| 【-v/-version】 | 查看程序版本号信息。 | 无。 |
| 【-h/--h/--help】 | 查看帮助信息。 | 无。 |

2．参数配置示例

这里将列举出常用的输入举例，thserver 是我们应用程序。如下所示:

- thserver

该命令相当于 thserver –setclk 12M –port 1025。使用默认选项。

- thserver –stclk 13000k –port 1111
  设置 socket 通信端口为 1111，硬件 ICE 的频率设置为 13M。
- thserver –scr E:\jtag.ini

  执行指定的 thserver 脚本。

## 2.3.2. thserver 脚本配置功能说明

Console 版 thserver 支持脚本运行功能，脚本分为 JTAG 脚本和 GPIO 脚本。JTAG 脚本可直接操作 Jtag 扫描链寄存器，通过自定义组合，即可访问相应的 HAD 寄存器，亦可以用于操作其他硬件； GPIO 脚本可控制 ICE JTAG 引脚电平的输出，也可以获取 ICE JTAG 引脚的电平，用户通过该脚本可产生自定义波形。可产生对应引脚上需要的波形，也可以读取 JTAG 引脚上的电平状态。

以下两个小节对两种类型脚本的使用方法进行详细介绍：

### 2.3.2.1.JTAG 脚本

thserver 支持通过执行 Jtag 脚本来读写 HAD 寄存器。

JTAG 脚本文件介绍说明如下：

1. 文件后缀没有要求，可以是相对路径

2. 语法规则

（1）脚本中可以描述多次的 jtag 操作，每个操作单元必须以[JTAGx]作为关键字，其中 x 表示数字。[JTAGx]要求必须大写，并数字连续，一旦出现不连续，后续不执行。

（2）IR/DR 必须大写，格式如下：

IR=[ir 长度], byte0, byte1 … ——IR 写

DR=W,[dr 长度] byte0, byte1 … —— DR 写

DR=R，[dr 长度] ——DR 读

（3）IR/DR 长度单位是 bit，但必须为的 8 倍数（版本号大于等于 V5.16.0 会支持非 8 对齐的 bit 数），这里对 ir/dr 长度支持 byte 倍数。

（4）byte 内容必须为十六进制，可以有前缀 0x 也可以没有。

（5）移进 jtag dr 的字节顺序按照先后顺序，同样读 DR 时，输出顺序也是按照字节先后顺

序。

3. 脚本实例

```
[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 脚本

Thserver 支持通过 GPIO 脚本控制 JTAG 引脚电平的输出，可产生对应引脚上需要的波形；也可以读取 JTAG 引脚上的电平状态。

GPIO 脚本文件介绍说明如下：

1. T-Head GPIO 脚本以[THEAD_ICE_GPIO]字段表征，如果不是以该字段开头的脚本文件，则认为不是 T-Head GPIO 脚本。
2. 文件后缀没有要求，可以是相对路径。
3. 语法规则：
   (1) 脚本以行为解析执行单位，每一行进行一次解析执行。同一行中可以有多个语句，语句之间逗号隔开。最后一条语句后面不得加逗号，否则为语法错误。
   (2) 同一行的多条语句，允许对同一个寄存器的同一个位进行操作。采取后面的覆盖前面的原则。如 TDI=1,TMS=1,NRST=0, TDI =0，那么最终解析执行的结果控制 TDI

的电平输出为 0；

(3) 对单个引脚赋值语句中如 TMS=1，赋值必须是 0 或者 1，否则语法错误。以 0x 开头的值视为 16 进制，否则 10 进制处理。

(4) 在执行脚本过程中，各个 JTAG 引脚电平的初始化值为 0，本次操作中没有对该引脚进行赋值输出操作，那么该引脚的电平保持上次操作时的电平；

(5) GPIO_OE 寄存器默认的值位 0x3f，即 JTAG 各个引脚对应的均为输出模式。对 GPIO_OE 寄存器进行赋值操作更改引脚的输入输出模式，GPIO_OE 寄存器一经赋值后一直有效，直到下一次对 GPIO_OE 进行赋值；

(6) 脚本支持 repeat 循环语句，语法为：REPT=x.....ENDR。支持嵌套循环，但是 REPT 和 ENDR 必须得匹配，否则语法错误；REPT 后面的等号不能省略，循环次数 x 不能小于 0，否则语法错误；REPT 语句和 ENDR 语句必须单独一行，否则语法错误；

(7) PRINT 语句用来读取 GPIO_IN 寄存器的值,语法为 PRINT=GPIO_IN,否则语法错误；

(8) 脚本内容以 START 开始，以 END 结尾，否则提示语法错误；

(9) 脚本不区分大小写；

(10) #后的语句为注释部分。

4. 支持的语句类型

THEAD_ICE_GPIO 脚本支持的几类语句：

- 对引脚输入/输出方向控制语句，对各引脚的输入输出方向进行控制,如 GPIO_OE = 0x1f；
- 对单个引脚进行输出的赋值语句，控制单个引脚的输出。如 TMS = 1，将 GPIO_OUT 寄存器中 TMS 引脚对应的位赋值为 1；
- 对所有引脚一次性赋值语句，如 GPIO_OUT=0x1d，将值赋值给 GPIO_OUT 寄存器；
- PRINT=GPIO_IN 语句，读取 GPIO_IN 寄存器的值，并且打印；
- REPT，ENDR 语句，即 repeat 循环语句，REPT=x,其中 x 为循环次数，即将 REPT 和 ENDR 之间的语句循环 x 次。

**注意：**脚本中的输出赋值语句如 TMS=1 和输入打印 PRINT=GPIO_IN，仅仅是对 ICE 中的

GPIO_OUT 和 GPIO_IN 寄存器进行读写，只有在 GPIO_OE 寄存器的操作中，JTAG 引脚才对对应的位进行正确的设置。比如，TMS 对应的 GPIO_OE[3]=1 时，即输出模式，GPIO_OUT[3]的值才会反应在 TMS 引脚上；同样的，只有当 GPIO_OE[3]=0，即输入模式时，PRINT=GPIO_IN 语句读回来的 GPIO_IN[3]才是真正 TMS 引脚的电平值，否则该值无效。

5. 脚本实例

```
[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.脚本执行

Console 版 thserver 执行脚本为：在 thserver 运行参数（2.3.1）中【-scr filename】。

GUI 版 thserver 执行脚本在菜单栏工具栏（2.4.2）中设有对应按钮  。

注：由于脚本执行需要连接 ICE 及开发板，故 GUI 版 thserver 执行脚本前先运行 thserver 确认硬件已连接正确，再停止 thserver，点击脚本按钮选择脚本执行。

### 2.3.3. 运行说明

1. 步骤

- 点击“开始->所有程序->T-Head->T-Head DebugServer->bin->DebugServerConsole.exe 以默认运行参数打开，运行界面如图 2-12 所示。
- 如果用户需要更改运行参数，可以“CTRL+c”断开连接；此时当前目录为安装目录，只需输入待运行参数的运行命令，如“DebugServerConsole.exe -port 1028”即可。
- 运行 T-Head GDB 应用程序进行调试：

  1. 使用 T-Head 工具链生成 T-Head elf 程序 a.out。

  2. 启动 GDB，比如 csky-*-gdb a.out。

  3. 根据 thserver 界面上的提示，在 GDB 的命令行输入连接命令，比如 target remote 172.16.28.158:1025。

  4. 当 GDB 连接上 thserver 之后，可以进行 GDB 常用的操作，比如：

     ① load // 下载程序至开发板

     ② break main // 在 main 函数处设置断点

     ③ continue // 运行程序

     ④ info registers r0 // 查看寄存器 r0

     ⑤ print var_a // 查看程序变量 var_a

  5. GDB 的使用方法 GNU GDB 保持一致。

- 如果使用 CDK 或 CDS，请参考开发环境所带的用户文档。

2. 运行界面：

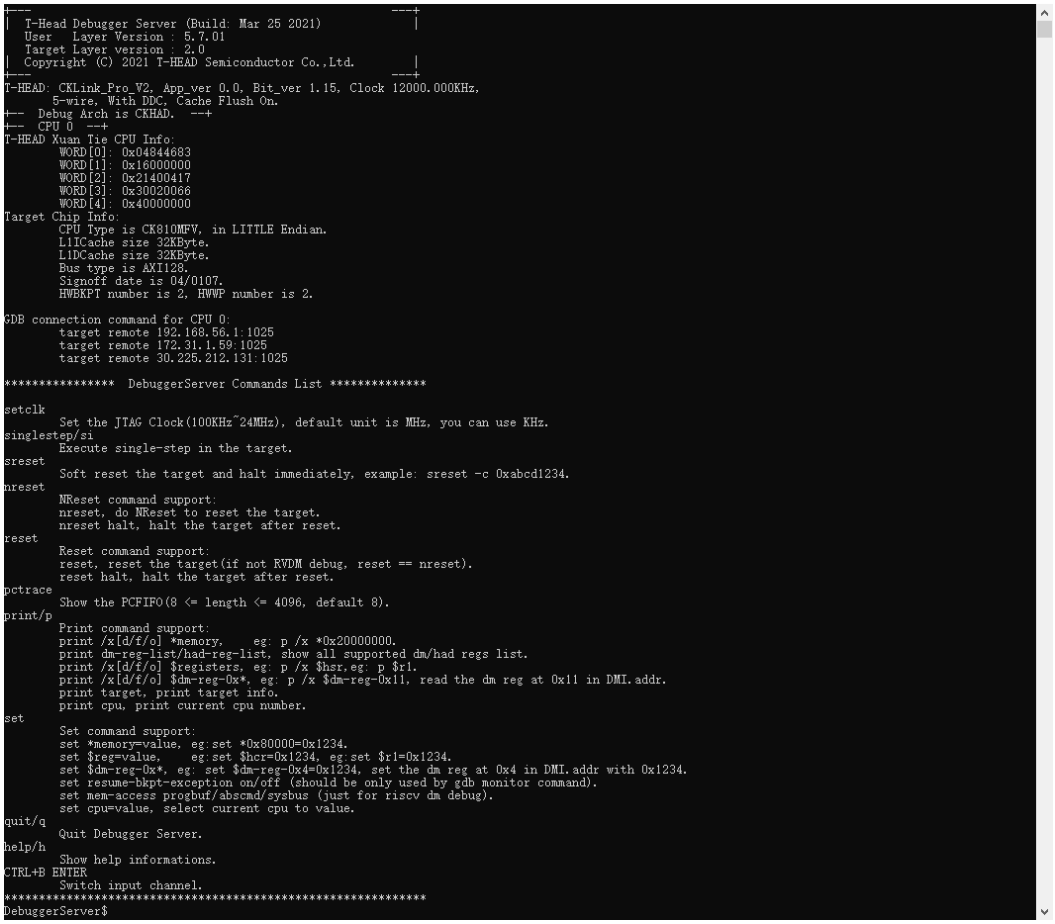



图 2-10 console 版 thserver 运行界面

3.注意事项

（1）如果运行 thserver 时提示 ICE Upgrade，选择“是”，进行固件升级，升级完成后需要重新插拔 ICE，使用 thserver 重新连接。

（2）当目标为多核的情况下，thserver 针对 CPU 在结构中的编号，从用户指定的端口（默认从 1025 开始）开启连续的多个端口供 GDB 连接。

## 2.4. GUI 版使用说明

### 2.4.1. 主界面介绍

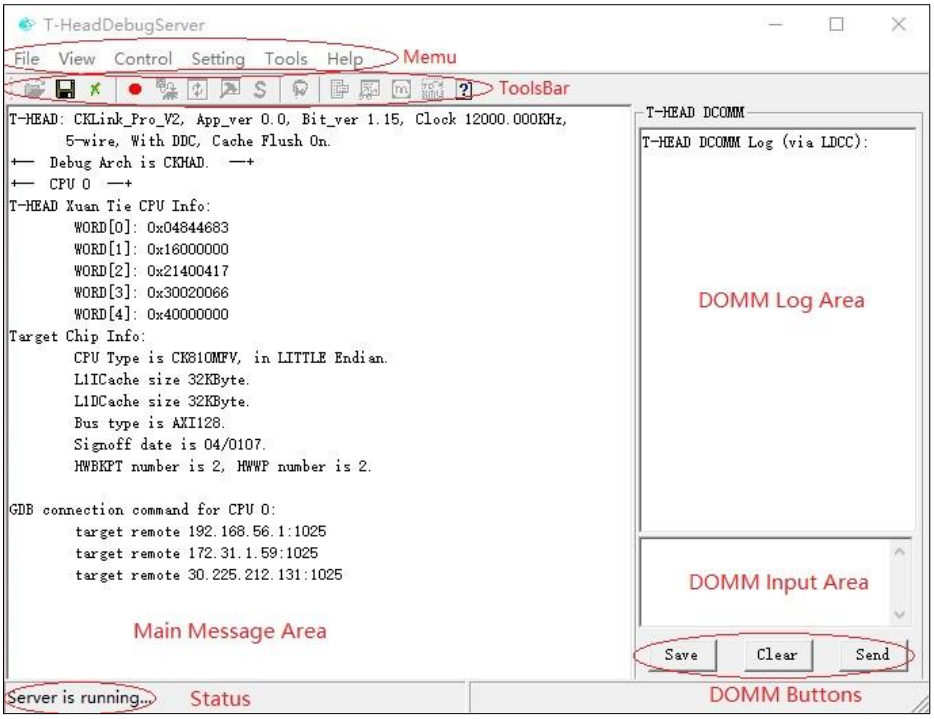
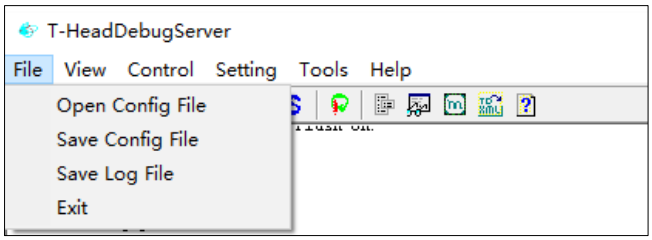



图 2-11 thserver GUI 主界面

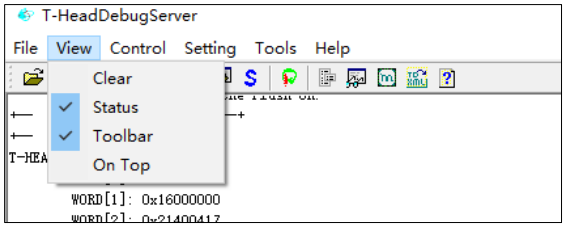
### 2.4.2. 菜单栏工具栏介绍

表格 2-2 File 菜单栏

| 菜单名称 | 功能介绍 | 工具栏按钮 |
|---|---|---|
| Open Config File | 打开 Target 和 Socket 信息的配置文件。 |  |
| Save Config File | 保存 Target 和 Socket 配置信息文件。 |  |
| Save Log file | 保存 log 信息到 log 文件。 | / |
| Exit | 关闭程序。 | / |



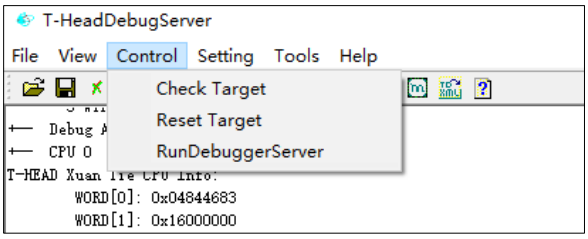

表格 2-3 View 菜单栏

| 菜单名称 | 功能介绍 | 工具栏按钮 |
|---|---|---|
| Clear | 清空 message area。 |  |
| Status | 显示/隐藏状态栏。 | / |
| Toolbar | 显示/隐藏工具栏。 | / |
| On Top | 显示总在最前。 | / |
|  | | |



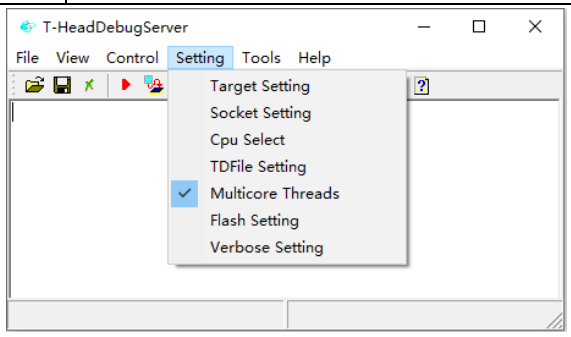
表格 2-4 Control 菜单栏

| 菜单名称 | 功能介绍 | 工具栏按钮 |
|---|---|---|
| Check Target | 检查连接的 Target 目标。 |  |
| Reset Target | 对目标板进行复位，复位功能依赖于目标板 jtag nrst 信号连接方式。 |  |
| RunDebuggerServer | 运行/停止 Debugger Server。 |  (运行)<br>  (停止) |



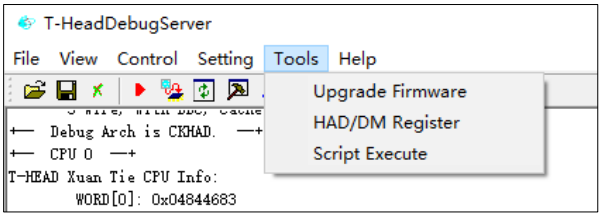

表格 2-5 Setting 菜单栏

| 菜单名称 | 功能介绍 | 工具栏按钮 |
|---|---|---|
| Target Setting | 设置 ICE 配置参数，包括 ICE 工作频率、是否使用 ddc 等，详见图 2-12 其说明。 |  |
| Socket Setting | 设置通信端口（默认 1025），详见图 2-15 及其说明。 |  |
| Cpu Select | 在调试多核开发板时，在 thserver 界面当查看 HAD/CPU 寄存器时需先选择对应的 CPU。 |  |
| TDFile Setting | 为 GDB 指定描述目标板寄存器信息的 xml 文件。 |  |
| Multicore Threads | 设置连接 SMP 多核调试时，是将每个核看作一个独立的个体，还是将所有核看出一个整体。<br><br>为假时，thserver 会为每个核开启一个调试端口，GDB 连接指定端口即为调试该端口对应的核。端口与核一一对应。<br><br>为真时。thserver 只会开启一个调试端口，同时封装多核为多个线程信息发送给 GDB。GDB 调试线程，则在调试线程对应的核。 | 无快捷键 |
| Flash Setting | 配置 Flash 烧录使用的算法文件，Flash 断点和 Simulated 断点配置。 | 无快捷键 |
| Verbose Setting | 设置 thserver 的 log 信息输出：<br><br>usb:记录 thserver 与 ICE 之前交互的协议包。 | 无快捷键 |

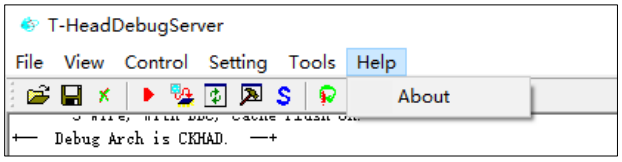
| | | |
|---|---|---|
| | connect:描述连接开发板的细致过程。<br>target:记录 target 抽象层函数调用信息。<br>remote:记录 remote 协议交互信息。<br>djp:记录 djp 协议交互信息。<br>sys:记录 thserver 程序主循环信息。<br>flash:记录 flash 烧录和 flash 断点信息。<br>all:打印以上所有 Log 信息 。<br>选择需要的 log 选项后，点击 select，重新使用 thserver 连接开发板即可。 | |
| |  | |

表格 2-6 Tools 菜单栏

| 菜单名称 | 功能介绍 | 工具栏按钮 |
|---|---|---|
| Upgrade Firmware | 升级 ICE 设备固件（注意：升级文件与 ICE 盒子类型要保持一致,具体对应如下：<br>CKLINK_LITE_V2: cklink_lite.hex<br>CKLINK_PRO_V1: cklink_v1.iic<br>CKLINK_PRO_V2: cklink_pro.iic<br>） |  |
| HAD/DM Register | 操作 HAD 模块的寄存器或 RISCV 调试架构下的 Debug Module 寄存器，具体根据实际 Debug Arch 而定。 |  |

| | | |
|---|---|---|
| Execute Script | 执行 thserver 脚本。 |  |
|  | | |

表格 2-7 Help 菜单栏

| 菜单名称 | 功能介绍 | 工具栏按钮 |
|---|---|---|
| About | 显示 Debugger Server 版本信息。 |  |
|  | | |

## 2.4.3. 启动配置文件说明

启动文件默认为与 thserver 所在目录下的 default.ini 文件，文件内容说明：

【TARGET】标签：

- JTAGTYPE=USBICE; 配置在线调试器类型，当前仅支持 USBICE。
- ICECLK=1200; 配置在线调试器工作频率为 12Mhz。
- DDC=TRUE; 开启 T-Head 硬件快速下载通道。
- OPTIONS：用于指定 2.3.1 中的参数，版本号小于 V5.18.0 的仅支持-onlyserver，大于等于 V5.18.0 的支持所有参数。
- CACHEFLAG=TRUE; 连接，单步，退出调试模式时是否刷 cache。
- MTCRDELAY=10; 执行 mtcr 指令的延时。
- TARGETINIFILE=; 在获取 ICE 后执行的目标初始化脚本，脚本类型为 GPIO 或 JTAG 脚本，执行脚本后继续 Server 启动流程。
- CDITYPE=2/5; 指定获取 ICE 后操作目标板的模式，2 线或 5 线。

- PRERESET=FALSE; 开启在获取 ICE 后向目标板发起 NReset 信号。
- TDESCXMLFILE=; 指定描述目标板寄存器信息的 Xml 文件。
- NRESETDELAY=100; 设置 thserver 发出 NReset 信号的时间长度。
- TRESETDELAY=110; 设置 thserver 发出 TReset 信号的时间长度。
- RESETWAIT=50; 设置等待 Reset 操作完成的延时。
- MULTICORETHREADS=TRUE; 设置调试 SMP 多核的模式。
- DCOMMTYPE=LDCC; 开启检查 DCOMM 输出信息，检查方式为 LDCC。
- LOCALSEMIHOST=FALSE; 设置处理程序半主机请求由 thserver 完成。
- DEBUGARCH=AUTO; 设置当前连接的调试模块，CKHAD/RISCV/AUTO, 分别指代 T-Head HAD，RISCV DM 和自动探测。
- DMSPEEDUP=TRUE; 读写 GPR 寄存器，读写内存操作由 ICE 封装，为 FALSE 时则有 thserver 上层操作 JTAG 完成这些操作。
- CACHEFLUSHDELAY=10; 指定 Cache Flush 的延时*ms。
- TRST=TRUE; 使能 treset。
- IDLEDELAY=; 设置 IDLE Delay 0~255。
- SAMPLINGCPF= ; 使能 CPF 采样 TRUE or FALSE。
- SAMPLINGCPU= ; 指定被采样的 CPU。
- SAMPLINGPORT= ; 指定用于 CPF 采样的端口号。
- SAMPLINGFREQ= ; 设置采样频率。
- SAMPLINGTYPE= ; PCFIFO-HOST/PCFIFO-LINK, 设置采样方式为 host 或 cklink。
- NDMRESETDELAY= ; 设置 DMCONTROL.ndmreset 的延时时间。
- HARTRESETDELAY= ; 设置 DMCONTROL.hartreset 的延时时间。
- FLASHALGORITHMPATH=; 指定 flash 算法文件。
- FLASHTIMEOUT=60; 指定 flash 算法文件执行一次 flash 操作后函数调用后运行至__bkpt_label 的超时时间。
- DISABLEFLASHBP=FALSE ; 关闭 flash 断点。
- DISABLESIMBP=FALSE; 关闭 flash 断点中的指令模拟功能。
- COMPATALGORITHMVER1TO5=; 是否兼容使用 V1~V5 版本的 flash 驱动。

- ENTERDEBUGTIME=; 设置发起同步使 CPU 进调试的超时时间。
- DMIBUSYADJUSTTIMES=10; 设置当发现 DMI.op 为 busy 时再次尝试发起当前 DMI 操作的次数，范围 10~500，每发现 2 次 DMI.op 为 busy 会增加一个 IDLE Delay。
- SETRESETHALTREQALWAYSON=; 配置写 DMCONTROL 时是否写 DMCONTROL.setresethaltreq 为 1。
- NOHWBP=; 是否禁止使用硬件断点资源。

【SOCKETSERVER】标签

- SOCKETPORT=1025; 设置 thserver 开启 Socket 服务的端口。

【VENDOR】标签

- VID=; 指定 LINK 的 Vendor ID。
- PID=; 指定 LINK 的 Product ID。

## 2.4.4. 常用功能介绍

1.打开/保存配置文件：

Thserver 支持使用已有的配置文件来完成 thserver 配置[File->Open Config File]以及将当前配置保存为配置文件[File->Save Config File]，配置信息包括 CPU 频率、是否使用 ddc 等。

2. Target 配置窗口：

除了上述使用已有配置文件，还可以使用手动配置，[Setting->Target Setting]打开配置信息对话框。如图所示：

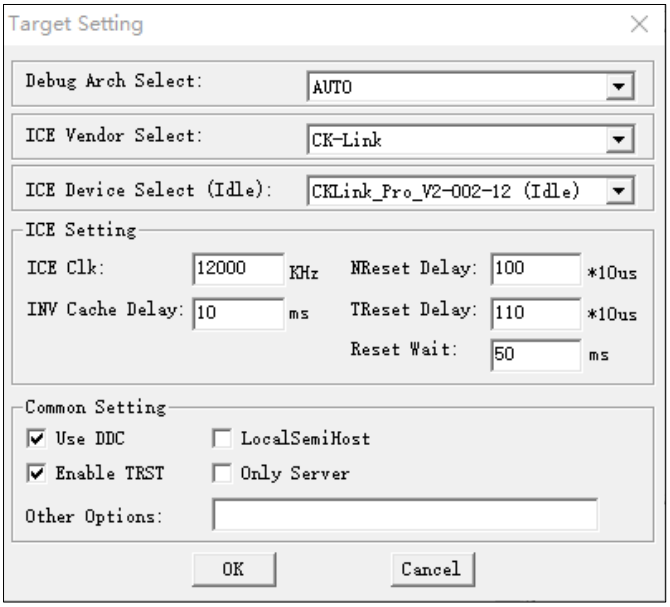



图 2-12 Target Setting 对话框

其中：

- Debug Arch Select: 选择调试架构，可选择 T-Head HAD，RISCV DM 和 AUTO。
- ICE Vendor Select：选择不同厂商的 ICE 设备，默认为 CK-Link。
- ICE Device Select(Free)：指定 ICE 连接。
- ICE Setting: 设置为使用 ICE 作为连接目标时的信息，包括：ICE 频率，mtcr 延时。
- NReset Delay：设置 NReset 延时，确保 ICE 可以生成稳定的硬件复位（nreset）信号，单位 10us，默认为 1ms。
- TReset Delay：设置 TReset 延时，确保 ICE 可以产生稳定的复位信号，用于复位 HAD 状态机，单位 10us，默认为 1.1ms。
- Reset Wait：设置延时，确保在目标板收到复位信号后，目标板复位流程执行结束，默认 50ms。
- Use DDC：选择下载时是否使用 DDC 直通通道下载。
- Enable TRST：执行 Reset Target 时是否执行 TRST。
- LocalSemiHost：指当程序发生 semihosting 请求时是否由 thserver 完成，默认由 GDB 完成。
- Other Options： 用于指定 2.3.1 中的参数，版本号小于 V5.18.0 的仅支持-onlyserver，大于等于 V5.18.0 的支持所有参数。

3．Socket 端口设置：

[Target Setting->Socket Setting]打开 Socket 端口设置对话框，如图所示，手动输入端口号即可，默认值为 1025 。

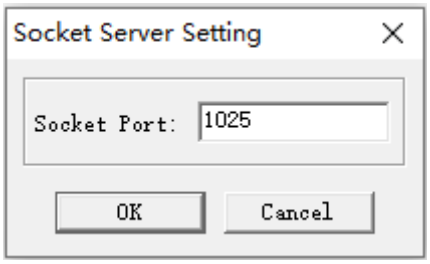



图 2-13 Socket Setting 对话框

4. 固件升级对话框：

[Tools->UpgradeFirmware]打开固件升级对话框，如图所示，

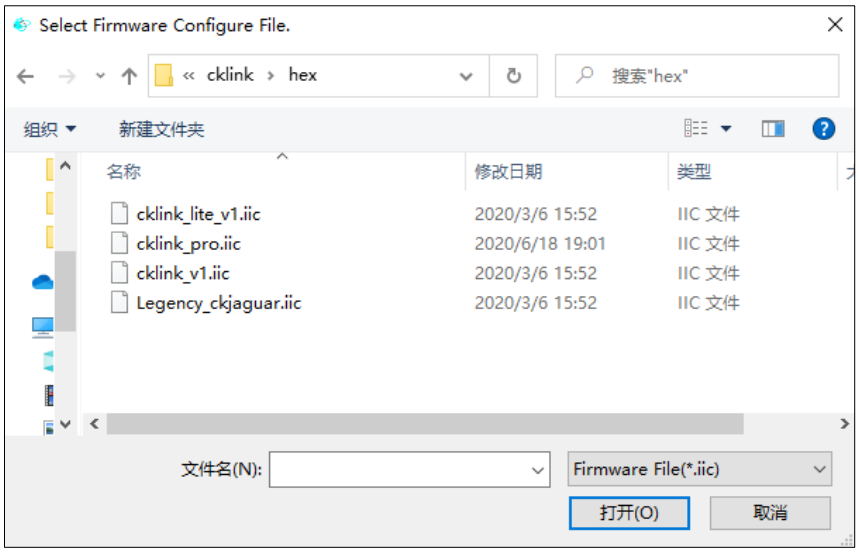



图 2-14 固件升级对话框

选择与你的 ICE 设备对应的的升级文件即可，升级文件与 ICE 盒子类型要保持一致,具体对应如下：

CKLINK_LITE_V2: cklink_lite.hex

CKLINK_PRO_V1: cklink_v1.iic

CKLINK_PRO_V2: cklink_pro.iic

5. Had/DM 寄存器操作窗口：

当当前 Debug Arch 为 T-Head HAD 时：

[Tools->HAD/DM Register]打开 HAD 寄存器的操作对话框，可以进行 HAD 寄存器的读写操作如图所示：

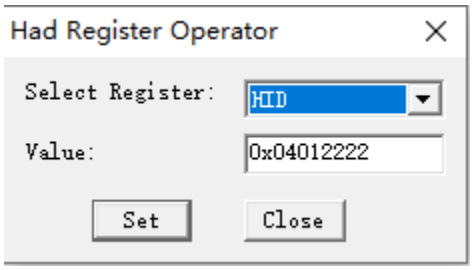



图 2-15 HAD 寄存器操作对话框

读操作：

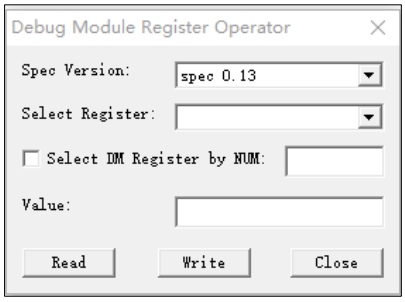
Select Register 选择要读取的 HAD 寄存器；Value 自动会显示为该寄存器的值。

写操作：

Select Register 选择要写入的 HAD 寄存器，在 Value 中写入想要写入的值，点击 Set，即可。

当当前 Debug Arch 为 RISC-V DM 时：

[Tools->HAD/DM Register]打开 DM 寄存器的操作对话框，可以进行 DM 寄存器的读写操作如图所示：




Spec Version 已指明当前探测到的 DM 版本信息。

Select Register 用于选择一个 DM 寄存器，选择后可选择 Read 或 Write 操作。如果 Select Register 中的 DM 寄存器没有包含用户要操作寄存器，则可以使用“Select DM Register by NUM”来指定编号进行读写。

## 2.4.5. 运行说明

1. 步骤

- 点击“开始->所有程序->C-Sky-> T-Head DebugServer-> T-HeadDebugServer，此时会以默认配置打开 thserver，界面如图所示。
- 如果需要更改 thserver 配置，点击 RunDebuggerServer（运行/停止调试 thserver）断开连接，然后按照“2.4.2 菜单栏工具栏介绍”进行配置。然后点击 RunDebuggerServer（运行/停止调试 thserver）重新连接。
- 运行 T-Head GDB 应用程序进行调试：

  1. 使用 T-Head 工具链生成 T-Head elf 程序 a.out
  2. 启动 GDB，比如 csky-*-gdb a.out
  3. 根据 thserver 界面上的提示，在 GDB 的命令行输入连接命令，比如 target remote 172.16.28.158:1025
  4. 当 GDB 连接上 thserver 之后，可以进行 GDB 常用的操作，比如：

     ① load // 下载程序至开发板

     ② break main // 在 main 函数处设置断点

     ③ continue // 运行程序

     ④ info registers r0 // 查看寄存器 r0

     ⑤ print var_a // 查看程序变量 var_a
  5. GDB 的使用方法 GNU GDB 保持一致
- 如果使用 CDK 或 CDS，请参考开发环境所带的用户文档。

2. 运行界面

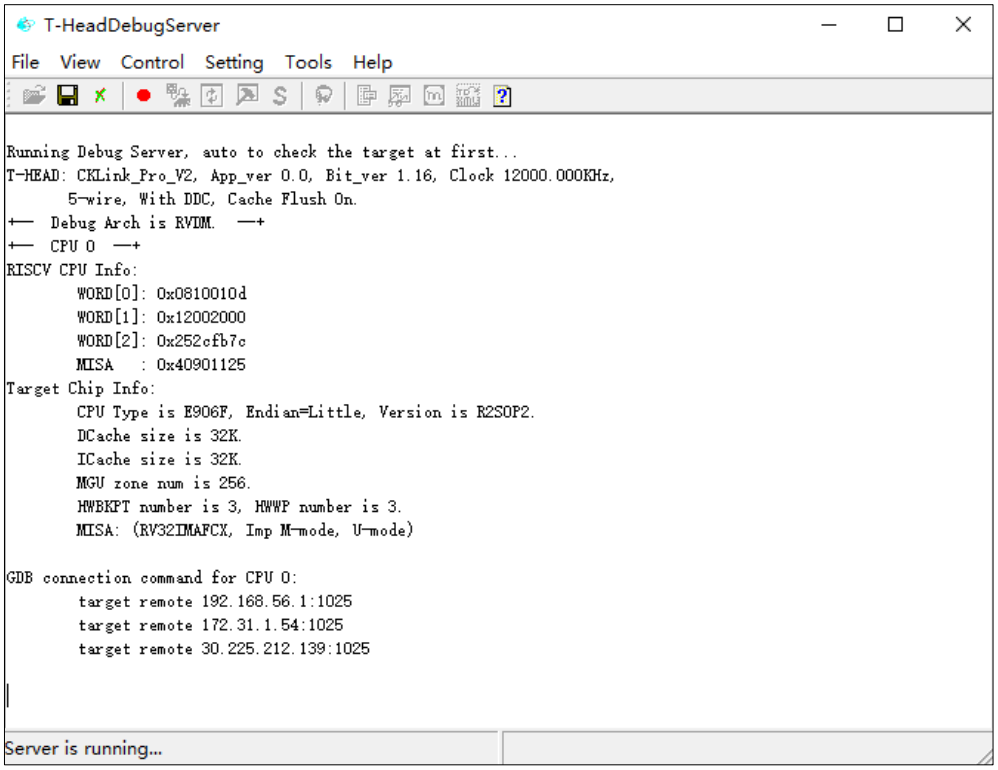



图 2-16 thserver GUI 运行界面

3.注意事项

（1）如果运行 thserver 时提示 ICE Upgrade，选择“是”，进行固件升级，升级完成后需要重新插拔 ICE，使用 thserver 重新连接。

# 3. Linux 篇

Linux 版本的的 thserver 没有 GUI 版本，只支持命令行输入，所以 Linux 版本的操作同 windows 下的 console 版本的操作

## 3.1. thserver 及 ICE 驱动安装

### 3.1.1. 安装包获取

从 T-HEAD 公司的 OCC 平台 https://occ.T-Head.cn/community/download_detail?id=616215132330000384 或技术支持处 获取安装包，安装包中包含 Windows 下的 T-Head-DebugServer-windows*.exe 安装文件和 Linux 平台的两个 T-Head-DebugServer-linux-*.sh 安装文件（分别对应 32 位和 64 位）。

T-Head-DebugServer-linux-i386-*.sh 是 32 位机的安装包，T-Head-DebugServer-linux-x86_64-*.sh 是 64 位机的安装包。请对应主机系统，选择对应的安装包。

### 3.1.2. thserver 安装步骤

1.在进行安装过程中，用户需要获得 sudo 权限。

2.通过命令 chmod+x 增加安装包的执行权限

3.执行 sudo ./T-Head-DebugServer-linux-*.sh –i 开始安装

4.给出提示“Do you agree to install the DebugServer[yes/no]”，输入 yes。

5.系统提示设置安装路径“Set full installing path:”

（1）默认路径安装：直接按回车，那么会给出提示“This software will be installed to the default path: (/usr/bin/)?[yes/no/cancel]:”，用户输入 yes，软件会安装到默认路径“/usr/bin/”目录下。安装完成后提示

“Done ！

You can use command “DebugServerConsole” to start DebugServerConsole！

(NOTE：The full path of ‘DebugServerConsole.elf’ is /usr/bin/T-Head_DebugServer) ”

（2）安装到用户指定目录：用户输入安装路径（绝对路径），安装会给出提示“This software will be installed to the path:(用户输入的路径)？[yes/no/cancel]:”，确认路径无误，输入“yes”并按下回车键。此时安装包将进行安装，安装完成后将提示：

“Done！

You can use command “DebugServerConsole” to start DebugServerConsole！

(NOTE：The full path of ‘DebugServerConsole.elf’ is 用户输入的路径/T-Head_DebugServer) ”

## 3.2. 运行环境

thserver 运行支持 Ubuntu 等大多数 Linux 平台，输入输出设备要求具有 USB 接口。thserver 配合 T-Head GDB 使用，支持所有版本。

## 3.3. 运行参数

同“2.3.1 运行参数”。

## 3.4. Jtag 脚本配置功能说明

同“2.3.2 Jtag 脚本配置功能说明”。

## 3.5. 运行说明

1.步骤

- 安装完成后，在任意目录下通过命令“DebugServerConsole +运行参数”，来打开 thserver，如果不加运行参数则使用默认运行参数运行 thserver。
- 运行 T-Head GDB 应用程序进行调试，运行 T-Head GDB 应用程序进行调试：
  1. 使用 T-Head 工具链生成 T-Head elf 程序 a.out
  2. 启动 GDB，比如 csky-*-gdb a.out
  3. 根据 thserver 界面上的提示，在 GDB 的命令行输入连接命令，比如 target remote 172.16.28.168:1025
  4. 当 GDB 连接上 thserver 之后，可以进行 GDB 常用的操作，比如：

     ⑥ load // 下载程序至开发板

     ⑦ break main // 在 main 函数处设置断点

     ⑧ continue // 运行程序

     ⑨ info registers r0 // 查看寄存器 r0

     ⑩ print var_a // 查看程序变量 var_a

     (1) GDB 的使用方法 GNU GDB 保持一致
- 如果使用 CDK 或 CDS，请参考开发环境所带的用户文档。

2.运行界面与【2.3.3 运行说明】windows console 版本同。

3.注意事项

（1）在运行 thserver 之前，查看/etc/hosts 文件，查看是否已经存在主机 IP 和主机名，若不存在，则用户在首行添加。

（2）如果运行 thserver 时提示 ICE Upgrade，选择“是”，进行固件升级，升级完成后需要重新插拔 ICE，使用 thserver 重新连接。

# 4. 半主机功能说明

Semihosting(半主机)是指在调试过程中，使用 host 的输入输出替换设备端的输入输出的一种调试方式。例如 host 机器的键盘、磁盘读写、屏幕输出等输入输出设备都可以作为目标板的输入输出来使用。

通过 Semihosting 可以减少在开发过程中很多硬件设备的依赖，仅需要通过一种调试方式接入目标板即可。

semihosting 支持的常见的操作有

1.open

2.close

3.read

4.write

5.lseek

6.rename

7.unlink

8.stat

9.fstat

10.gettimeofday

11.isatty

12.system

T-Head 800 系列 CPU 半主机使用步骤：

1. 编写 T-Head 裸程序，其中包含有文件操作，或输入输出。
2. 编译工程时，编译器添加编译选项“-lsemi”，工具链版本要求大于等于 V3.8.x。
3. 使用 T-Head 调试器下载程序并调试，thserver 要求大于等于 V5.2.0。
4. 运行程序，程序中涉及 Semihosting 操作将会有 GDB 完成。比如 printf 输出将在 gdb 界面输出，fopen 可以打开 GDB 所在主机的文件等。

Semihosting 操作默认由 GDB 完成。如果由 thserver 完成，方法为在 thserver 启动时添加【-local-semi/-ls】参数（UI 版本启动方式请查阅 2.4 章节），则程序中的 Semihosting 请求将由 thserver 完成。比如 printf 输出将在 thserver 的界面中输出，fopen 可以打开 thserver 所在主机的文件等[默认情况下，Semihosting 的执行在 GDB 中完成，可通过[-local-semi/-ls]选项重设为在 thserver 中执行。

RISC-V 系列 CPU 半主机使用步骤：

1. C 文件需要 在 main 入口增加 init_semihosting()。
2. 使用版本大于等于 V2.2.0 的工具链编译，链接选项增加 --specs=semihost.specs。
3. DebugServerConsole -ls。
4. 使用 GDB 调试，elf 中出现的半主机请求将有 thserver 完成。

注意：

1. GDB 暂不支持 RISC-V 的半主机请求，目前只能由 thserver 完成。
2. 在 Windows 中加-local-semi/-ls 时，不支持 isatty 和 system 操作。

# 5. 调试输出功能说明

T-Head DCOM(Debug Communication)是基于 T-Head JTAG 通路支持调试过程中输出信息的通道，目前仅支持含有 LDCC（T-Head Light Debug Communication Channel）功能的 CPU。

使用步骤，以包含 LDCC 的 CK802 为例：

1. 编写 T-Head 裸程序代码，实现 fputc 如下：

```
#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. 使用 T-Head ELF 工具链编译代码，生成 Elf 文件。
3. 启动 thserver，执行”DebugServerConsole.exe -dcomm=ldcc”，UI 版本参考 2.4.3 章节说明。
4. 使用 T-Head GDB 连接 thserver 并全速运行程序。
5. 则程序中使用 printf 输出的信息将通过 LDCC 传输至 thserver 窗口显示。

# 6. 命令行功能说明

T-Head 提供了命令行进行 target 调试手段，用户可以在 thserver 启动后，通过相关命令实现读写 HAD 寄存器，读写 CPU 寄存器，读写内存等操作。支持的操作如下：

| 命令 | 描述 | 例子 |
|---|---|---|
| flash xxx | 指定 flash 算法文件或执行 flash 操作，具体查看【Flash 烧录及 Flash 断点】。 | |
| cacheflush [l2 start_addr end_addr] | 默认为刷新 L1 Cache，可加参数来刷新 L2 cache。 | cacheflush<br>cacheflush l2 0x0 0x10000 |
| setclk clock | 修改 CKLINK 的的频率，默认单位为 Mz，可指定使用 Khz。 | setclk 3<br>setclk 3KHz |
| singlestep/si | 执行指令单步。 | singlestep |
| sreset -c value [halt] | 执行软复位命令，-c 之后的数字根据具体实现而定。不加 halt，表示执行软复位后 CPU 全速运行;加 halt，表示执行软复位后进入调试模式。 | sreset -c 0x1234abcd<br>sreset -c 0x1234abcd halt |
| nreset [halt] | 执行 JTAG 接口的 NReset 操作。不加 halt，表示执行软复位后 CPU 全速运行;加 halt，表示执行软复位后进入调试模式。 | nreset<br>nreset halt |
| reset [halt] | 执行硬复位操作。不加 halt，表示执行软复位后 CPU 全速运行;加 halt，表示执行软复位后进入调试模式。 | reset<br>reset halt |

| | | |
|---|---|---|
| pctrace [count] [cpu_n] | 输出 pcfifo 的数据，可选择打印的数量和指定具体的核。 | pctrace<br>pctrace 16<br>pctrace 16 2 |
| p/print | 1.打印寄存器($)/内存(*)值，内存打印仅支持 Word 大小。已知位域含义的寄存器将会按照位域显示该寄存器的值。寄存器包含 HAD，RVDM 和 CPU 寄存器。<br>2. target， 打印 target 信息。<br>3. 打印指定位置的 DM 寄存器。<br>4. cpu，打印当前选择的 CPU 编号(仅在多核下有意义)。 | p $psr<br>p $hsr<br>p $dmstatus<br>p $pc<br>p $dm-reg-0x400<br>p *0x10000000<br>p target<br>p cpu |
| p/print sd_info start_addr length | 打印 system description 数。 | p sd_info 0x400 0x20 |
| p/print virtual-mem-access | 由于 RISCV DM 架构下 CPU 进调试为 M-mode，但进调试之前可能是 S/U-mode，且已经开启 MMU。为了保证调试器看到的内存值与运行在 CPU 上的程序看到的内存值保持一致，thserver 默认会在 DCSR.prv!=3 的情况下设置 DCSR.mprven 为 1, MSTATUS.mpp 为 DCSR.prv, MSTATUS.mprv 为 1。<br><br>上述功能默认开启，但也可以通过 set virtual-mem-access on/off 来修改，通过 p virtual-mem-access 来查看当前状态。 | p virtual-mem-access |
| p/print had-reg-list | 当前调试架构为 T-HEAD HAD 是，可使用该命令打印当前可查看的所有 HAD 寄存器名称。 | p had-reg-list |
| p/print dm-reg-list | 当前调试架构为 RISCV DM 时，可使用该命令打印当前可查看的所有 DM 寄存器名称。 | p dm-reg-list |
| p/print target | 打印当前已连接的目标板信息。 | p target |
| set | 1. 设置寄存器($)/内存(*)值，修改的内存地址需为 ram 空间。寄存器包含 HAD，RVDM 和 CPU 寄存器。<br>2. 设置释放程序运行时是否设置 CSR.fdb (仅针对 T-Head HAD 调试架构下使用)，即可选择控制在释放 CPU 运行后，CPU 遇到 bkpt 软件断点指令后进调试或进异常。 | set $r0=0x10000000<br>set $hcr=0x8000<br>set $dmcontrol=0x80000001<br>set $dm-reg-0x400=0x112233<br>set *0x10000000=0x1 |

| | | |
|---|---|---|
| | 3. 设置读取内存的方式，可选择 progbuf/abscmd/sysbus。执行后可根据提示确认是否切换成功。<br>4. 切换当前 CPU 编号。一般需要先使用 p cpu 查看当前 CPU，切换 CPU 查看信息后再使用 set cpu 来恢复当前 CPU。 | set resume-with-fdb on<br>set resume-with-fdb off<br>set mem-access progbuf<br>set mem-access abscmd<br>set mem-access sysbus<br>set cpu=1 |
| set resume-bkpt-exception on/off | 修改 CPU 执行软件断点指令之后的行为：on，进入断点异常；off，进入调试模式。 | set resume-bkpt-exception on set resume-bkpt-exception off |
| set mem-access progbuf/abscmd/sysbus | 当前架构为 RISCV DM 时，修改当前调试访问内存的方式。 | set mem-access progbuf<br>set mem-access abscmd<br>set mem-access sysbus |
| set virtual-mem-access on/off | 设置 thserver 访问内存时是否与当前程序保持一致，on 即与程序保持一致，off 则为调试器读写内存时的地址均为物理地址。默认为 on。 | |
| set mem-access-max-mode auto/dword/word/hword/byte | 设置 thserver 访问内存时，单次访问数据的最大位宽。auto 即使用调试架构支持的最大位宽。如果设置的位宽大于当前调试架构支持的最大为宽，则使用调试架构支持的最大为宽。<br><br>比如 E906，调试架构最大支持但此访问为 word，在如果被配置为 word，则操作大块内存时，与 word 对齐的部分会使用 word 访问，不对齐部分与 hword 对齐的使用 hword 访问，剩余的使用 byte 访问。<br><br>但如果被设置为 hword，则于 hword 对齐的部分使用 hword 访问，剩余的使用 byte 访问。 | set mem-access-max-mode auto<br>set mem-access-max-mode dword<br>set mem-access-max-mode word<br>set mem-access-max-mode hword<br>set mem-access-max-mode byte |
| q/quit | 退出 thserver | q |
| help | 打印 help 信息 | help |

该功能仅在 DebugServerConsole 版本提供，用户启动 DebugServerConsole 后即可使用上述命令进行需要的操作，UI 版本 DebugServer 已经在界面上实现了上述操作。

附寄存器列表：

T-Head HAD 调试架构（可以通过 p had-reg-list 查看）：

HAD 寄存器：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, hacr。

RISCV DM 调试架构（可以通 p dm-reg-list 查看）：

Debug Module 0.13 寄存器: 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, haltsum0。T-Head 添加 itr，customcs，customcmd，compid。

CPU 寄存器列表：CPU 寄存器列表请查看 DebugServer 安装目录 tdescriptions 文件夹中对应的寄存器描述。

# 7. XML 使用说明

## 7.1. 简介

基于 GDB 本身对 XML 文件描述寄存器的支持，T-Head-GDB 实现了一套 XML 文件描述调试目标寄存器的扩展机制。T-Head-GDB 根据 XML 描的寄存器名称，个数，寄存器组信息等来组建内部寄存器运行需求。在此机制下，用户可以根据接口，灵活方便的使用 XML 文件描述目标的寄存器信息。需要注意的是，该部分的直至仅限于支持 T-HEAD 800 系列 CPU。

## 7.2. XMl 文件格式

### 7.2.1. 编写规则

GDB 内部对描述文件规则有：

- 描述文件遵守 XML 文件规则。
- XML 文件中 target 为根元素，具有 version 属性，默认为 1.0。
- Target 下同级的元素有 architecture，feature。
- Architecture 的值表明该文件描述的体系结构，比如 arm，mips，csky。

- Feature 有 name 属性，该属性表明了对寄存器的划分。
- Feature 下主要元素为 reg，reg 的属性有 name（寄存器名称），bitsize（位宽），regnum（GDB 内部编号），type（数据类型），group（该寄存器所属组名），save-restore（GDB 内部使用的属性）；这些属性中 name 和 bitesize 为必要属性，其他的为可选属性。
- Feature 下还有 vector，struct，union，flags 元素，这些元素是作为对目标寄存器类型的描述的扩展，在附录中描述。

T-Head-GDB 在上述规则的基础上补充如下规则：

- Target 具有 version 属性，默认为“1.0”，而当前 T-Head-GDB 只支持“1.0”。
- Architecture 中必须为 csky。
- Feature 名称为 org.gnu.csky.abiv1.xxx 或 org.gnu.csky.abiv2.xxx 。
- 每个 Feature 中的第一个 reg 需要有 regnum 属性，之后 regnum 连续的 reg 可没有该属性，自动根据上一个 reg 的 regnum 加 1；如过出现不连续的，则需要有 regnum 属性。
- 如果 reg 需要在 group 中显示，则需有 group 属性。
- Reg 的属性值不应出现特殊字符。
- Pseudo 寄存器的 reg 描述编写在特定名称为“org.gnu.csky.pseudo”的 feature 下。
- Name, group 属性的合法字符范围为“a/A-z/Z, 0-9, _”，且 name 的长度不应大于 15 个字符。

T-Head Abiv2 GDB 对 feature 名称限定于以下字符串，如果不是以下字符串，则属于该 feature 的寄存器将不会被 GDB 接收：

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 Abiv2 GDB 对寄存器名称补充：

对于控制寄存器，bank0 可以使用 cr0~cr31 进行描述，bank1~bank31 的 GDB 默认支持 cpxcry 的名称。

cpxcry 的解释为：x 指代 bank 编号从 1~31，y 指代该控制寄存器在组内的编号，从 0~31。比如 bank4 的 21 号寄存器，其名称则被描述为 cp4cr21。

其他名称如果需要支持，则需要先在 GDB 添加支持。

### 7.2.2. 举例描述

T-Head 描述目标寄存器信息的 xml 文件样本如下：

```
<?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>
```

注释：

- XML 文件中 target 为根元素，version 为 1.0。
- architecture 中表明该文件描述的是 T-Head 体系结构。

- feature 的 name 为 org.gnu.csky.abiv1.gpr 表明该 feature 描述的是 T-Head 体系结构下 abiv1 的通用寄存器信息。
- 描述的第一个寄存器的名称为 r0，位宽为 32，在 GDB 内部编号为 0，类型为 uint32，属于寄存器组 gpr。
- 描述的第二个寄存器的名称为 r1，位宽为 32，在 GDB 内部编号为 1。
- 描述的第三个寄存器的名称为 r2，位宽为 32，在 GDB 内部编号为 2。
- 第二个 feature 的 name 为“org.gnu.csky.pseudo”表明该 feature 是作为描述 pseudo 寄存器的特定 feature。
- 第二个 feature 的第一个寄存器名称为 r01，位宽 64 位，类型为 uint64，寄存器编号。

在 XML 文件中使用到的寄存器编号，即 regnum 的内容指表格【表格 7-1 abiv1 寄存器编号】和【表格 7-2 abiv2 寄存器编号】中的 remote 编号项内容。下列表格中对目前支持的和可预期支持的寄存器做了编号，对寄存器名称并未一一列出。在 T-Head-GDB 支持 XML 后，寄存器的名称可以根据需求修改。

#### 7.2.2.1. Abiv1 寄存器

Abiv1 目前的寄存器有：

- General registers r0~r15
- Optional registers r0~r15
- Cr0~cr31 控制寄存器
- Hi,Lo
- FPU
- PC
- FPC-Control
- MMU

寄存器编号为：

表格 7-1 abiv1 寄存器编号

| 寄存器名称 | GDB 编号 | 备注 |
| --- | --- | --- |
| R0~r15 | 0~15 | 列表中的 remote 编号均依照当 |

| | | |
|---|---|---|
| | | 前 GDB 对寄存的编号处理编写 |
| Hi,Lo | 20，21 | |
| FPU | 24~55 | |
| PC | 72 | |
| Optional registers r0~r15 | 73~88 | |
| Cr | 89~119 | 没有 cr31 |
| FPC-Control (命名为 cp1cr0~cp1cr6) | 121~127 | |
| MMU (命名为 cp15cr0~cp15cr16 和 cp15cr29~cp15cr31) | 128~147 | 总共 20 个 MMU 寄存器，remote 编号连续 |

### 7.2.2.2. Abiv2 寄存器编号

Abiv1 目前的寄存器有：

- General registers r0~r31
- Optional registers r0~r15
- Hi,lo
- FPU/VPU
- Profiling
- PC
- Cr_Bank0
- Cr_Bank1
- Cr_Bank3
- Cr_Bank15

寄存器编号为：

表格 7-2 abiv2 寄存器编号

| 寄存器名称 | GDB 编号 | 备注 |
|---|---|---|
| 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 | |
| Bank15 剩余 | 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. 扩展的 TEE 寄存器描述

### 7.2.3.1.扩展简述

在 T-Head TEE（安全）编程模型中，存在同一个寄存器在 TEE 和 REE 两个世界中各有一份。这些寄存器均需要切换世界或通过寄存器映射来读写，具体可查看 T-Head TEE 的相关手册。GDB 配合 DebugServer 的 xml 文件可辅助用户在任一世界时，查看当前以及另一个世界的寄存器信息（版本需求 T-Head GDB >= V3.10.0, DebugServer >= V4.5.0），由 DebugServer 辅助切换世界并读取对应世界的寄存器。

### 7.2.3.2.编写规则

1. TEE 寄存器描述功能添加在 pseudo 寄存器内部中，故其遵循【7.2.1】中 pseudo 寄存器相关的所有规则。

2. TEE 寄存器使用”env”属性表明该寄存器属于 TEE 还是 REE 的寄存器，属性值只有”ree”，”tee”可选。

3. 使用”regs” 属性来表明对应的物理寄存器编号。

4. bytesize，type 根据实际属性填写。

### 7.2.3.3.举例说明

下面是以 CK810T 为例的寄存器描述文件示例(用户可在 DebugServer V5.4.0 及之后的版本安装目录中，查看 tdescriptions 文件夹中的文件做参考)：

```
<?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>
```

根据上面的 xml 文件描述，用户在 T-Head GDB 层面查看 psr 即为当前世界的 psr，查看 t_psr

则为安全世界的 psr，查看 nt_psr 则为非安全世界的 psr。也就是说 psr 可能等于 t_psr，也可能等于 nt_psr，依据当前所处的世界而定。

### 7.2.4. UI 版

UI 版 thserver 提供了在启动配置文件（default.ini）中指定 XML 文件和界面窗口指定 XML 文件的方式。

**在 default.ini 中：**

Default.ini 作为 UI 版 thserver 的默认配置启动文件，在该文件中提供了用户设置 xml 文件路径的变量“ TDESCXMLFILE ”。用户可将编写好的 xml 文件路径赋值于该变量，当 GDB 发起获取描述目标描述信息的 xml 文件时，thserver 将打开用户指定的 xml 文件，并与 GDB 进行交互。

当用户使用 UI 窗口修改过 xml 文件路径后，thserver 关闭时，将提示用户是否修改默认的配置。用户选择是，则按照当前 thserver 的相关配置保存成配置文件。

**在 UI 窗口中：**

UI 版设置窗口指在 UI 版 thserver 的窗口界面上提供用户选择 xml 文件的接口。目前，在 setting 菜单栏下添加 TDFile setting 选项，如下图：

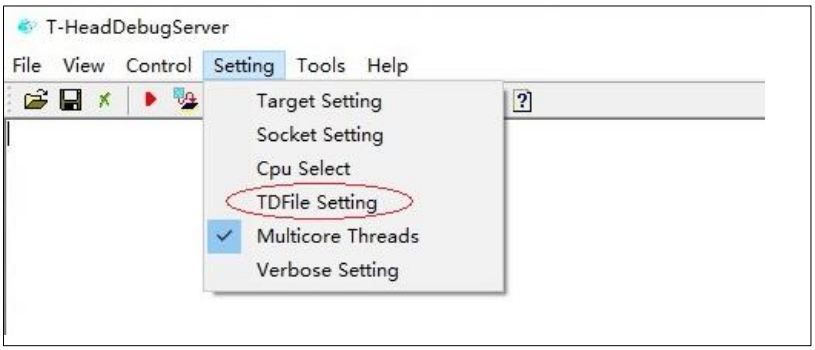



图 7-1 TDFile Setting 菜单选项

在工具条中添加快捷启动选择 xml 文件选择按钮，如下图：


图 7-2 TDFile Setting 快捷按钮

以上两种方式点击后均可弹出 xml 选则对话框，如下图：

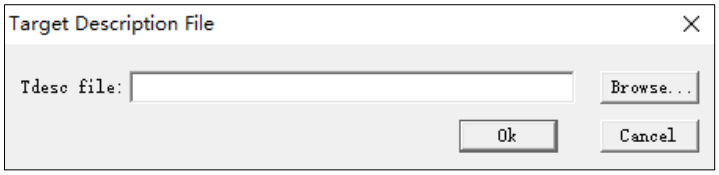



图 7-3 Target Description File 对话框

之后，点击 Browse 按钮选择本地描述目标的 xml 文件，确认后点击 Ok 按钮。

### 7.2.5. Console 版

Thserver Console 版添加启动参数“-tdescfile +filepath”选项来指定描述目标寄存器的 xml 文件。

**Windows 环境中：**

1) 建立 Console 版 thserver 的快捷方式


2) 右击打开快捷方式并点击属性

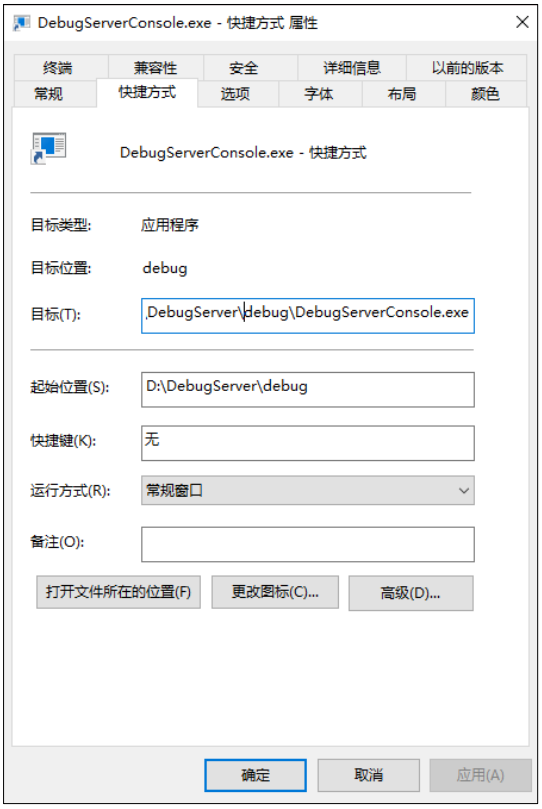



图 7-4 DebugServer 快捷方式属性

3) 在快捷方式页面的目标栏程序后面添加 -tdescfile xmlpath，xmlpath 为用户指定的 xml 文件路径


图 7-5 为快捷方式添加启动参数

4) 确认后点击确定按钮，启动该快捷方式，使用 T-Head-gdb 连接后进行正常调试。


图 7-6 修改启动参数后点击确认按钮

**Linux 环境中：**

使用 sudo ./Debugserver.elf -tdescfile /home/xxx/gdbxml/csky-abiv2.xml 启动 linux 版 thserver，之后使用 T-Head GDB 连接调试。

# 8. 多核调试操作使用说明

## 8.1. 简介

T-Head CPU 的同构多核有 2 个系列: 以 CSKY 指令集构建的 CK860MP 和以 RISC-V 指令构建的 C910MP，C920MP，C910v2MP，C920v2MP，C908MP 等。其中 CK860MP，C910MP 和 C920MP 使用的都是 T-Head HAD 硬件调试架构，其他的多核使用的都是 RISC-V DTM&DM 硬件调试架构。Thserver 已支持两种架构的自动探测，连接成功后，在连接的界面打印信息中会有类似“Debug Arch is CKHAD.”或“Debug Arch is RVDM.”的提示。

虽然两种硬件调试架构不同，但对于软件用户而言，thserver 为 SMP 多核抽象出两种模式：多核单端口和多核多端口模式。下面我们将对这些信息和不同模式的使用进行介绍。

## 8.2. HAD 模块构建的多核

T-Head CK860MP 是面向嵌入式系统和 SoC 应用领域的 32 位超高性能嵌入式多核心处理器，调试采用多个核单个 JTAG 接口的调试框架，通过一个共享的 JTAG 接口访问各个 CORE 的辅助调试单元（HAD），触发 CORE 进出调试模式和访问处理器资源。

同时 CK860MP 设置了一个集中的事件传输模块（ETM）用于支持多核之间调试事件（进入调试和退出调试）的传输。当 CK860 核心在接收到 ICE 发送的调试命令产生调试进入或者调试退出事件时，同步将该事件发送到 ETM，由 ETM 决定是否将该事件转发给其他的 CORE，实现多个核之间进行进入和退出调试信号交互，响应的目的。

即当一个核进入或退出调试模式时，可以选择将该信号发送给其他核。其他核可以选择是否响应该信号，同步进入或退出调试模式。硬件模型如下图：

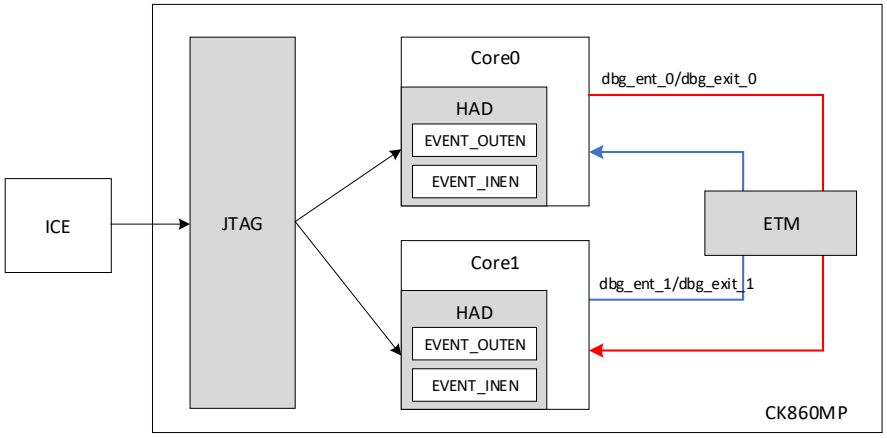



图 8-1 多核调试整体框架

同时 T-Head 实现 RISCV 多核 C910MP 的 ETM 模块与 C860MP 的实现一致，调试方式也一致。

## 8.3. DM 模块构建的多核

C908，C910v2 等后续产品将使用标准的 RISC-V DTM&DM 来实现多核，这里我们对 RISC-V DTM 与 DM 的信息进行一些介绍。在《RISC-V-External-Debug-Support_0.13.2.pdf》中，我们可以抽象出简单的模型：

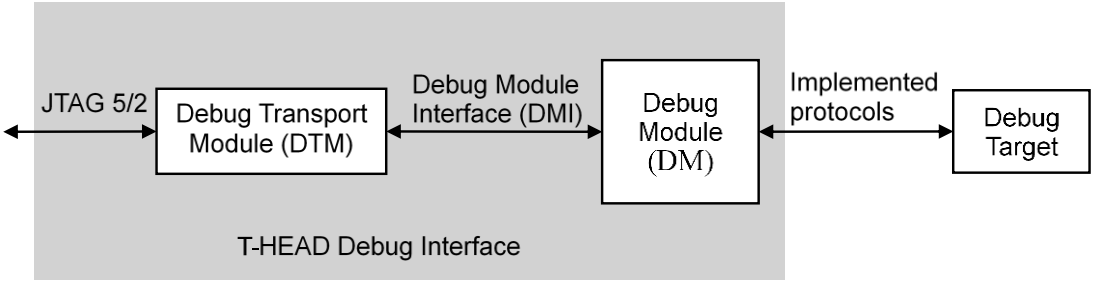



图 8-2 多核调试整体框架

外部通过 JTAG 驱动访问 DTM，DTM 通过 DMI 来访问 DM。DM 负责与 CPU 进行交互，来进行数据读写和功能控制。启动 DTM 只有一份，DM 可以通过链表的方式串起来：

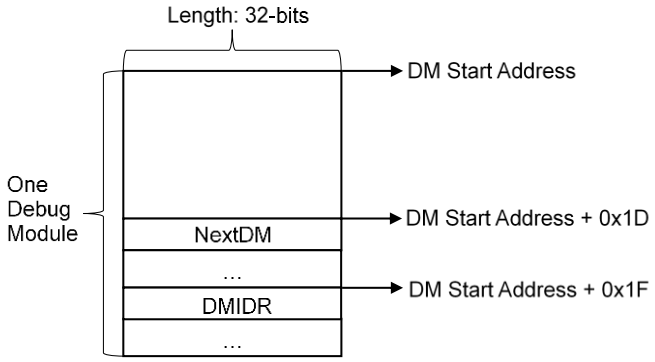



图 8-3 单个 DM 布局

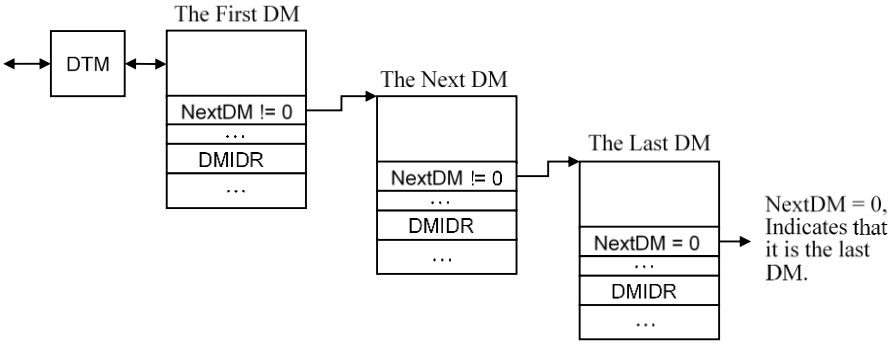



图 8-4 DM 链表布局

Spec 中，在单个 DM 的设计上最多可以支持 $2^{20}$ 个 CPU。那么通过我们会出现以下几种方式的多核：

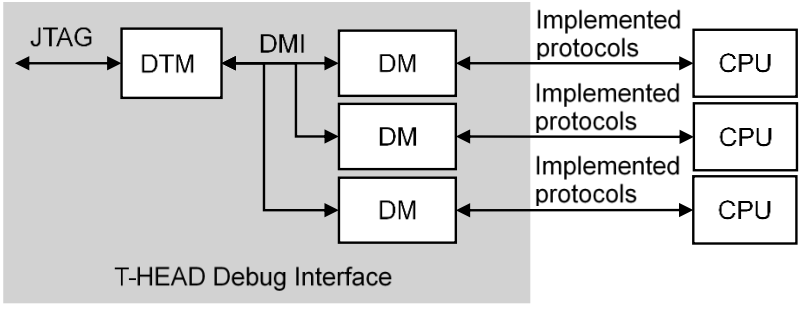



图 8-5 多个 DM，每个 DM 上挂一个 CPU

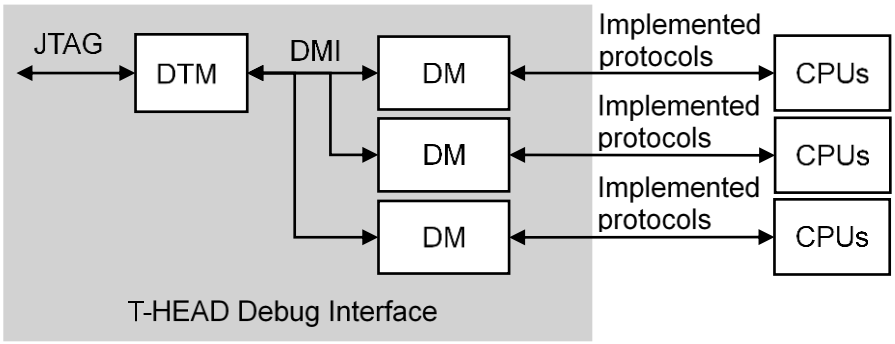



图 8-6 多个 DMs，每个 DM 上挂多个核

在 T-Head 的使用设计中，一个 DM 内的所有核默认为 SMP。同时在 T-Head 的实现中，对 SMP 多 clusters 多核在 DM 上实现 DMCS2 的实现，以辅助 DM 之间的调试信号互通。

## 8.4. 调试环境需求

**硬件需求：**

1. Ck860MP 开发板/C908MP 开发板
2. T-Head ICE CKLINK_PRO_V2 盒子一个，配套排线，USB 线
3. PC 机，windows 系统或系统为 linux 系统

**软件需求：**

1. csky-*abiv2*-gdb，来自 V3.6.x 及之后工具链
2. riscv64-unknown-elf-gdb 来自 V2.2.8 及之后的工具链
3. DebugServer V5.16.10 软件（根据环境获取 windows 版本或 linux 版本）

## 8.5. 多核单端口模式

在该模式下，DebugServer 将为可识别打的 SMP 多核只开启一个服务端口供 T-Head-GDB 连接，识别到的单核也为其开启一个复位端口。T-Head-GDB 通过”target remote ip:port” 方式连接上 DebugServer 后，DebugServer 将多个核的信息封装为线程信息反馈给 T-Head-GDB。用户在 T-Head-GDB 端可以以 Thread 的方式查看，调试多个 CPU。

该模式下，多个 CPU 将互相发送并响应其他核的调试信号，意为其中一个 CPU 进入或退出调试模式时，其他 CPU 同步进入或退出调试模式。

### 8.5.1. CK860MP 多核调试模型

模型如下：

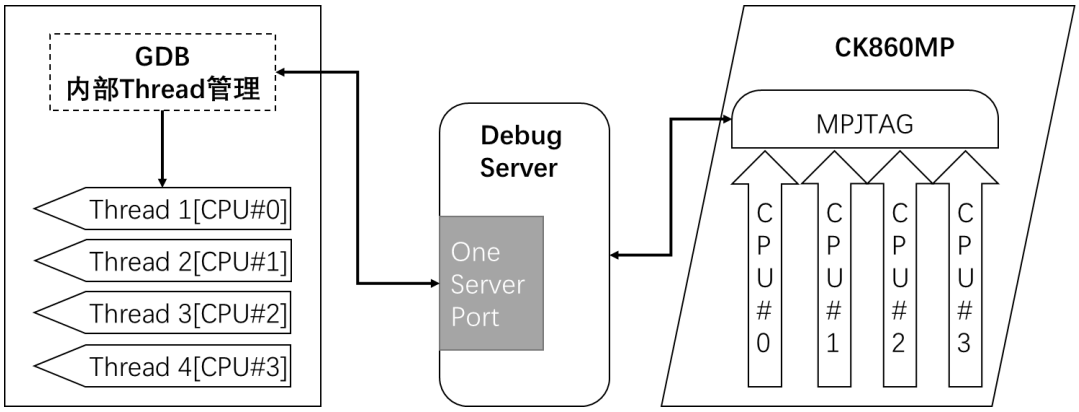



图 8-7 CK860MP 多核单端口模型图

### 8.5.2. C908MP 多核调试模式

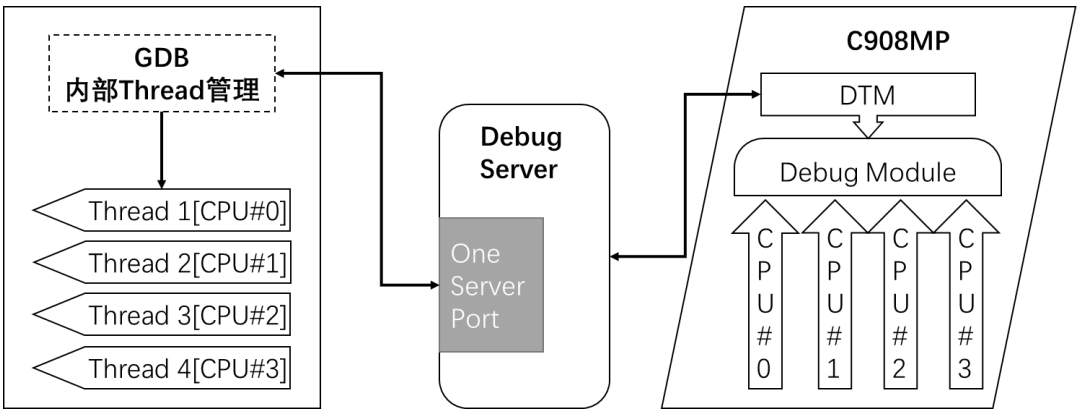



图 8-8 C908MP 多核单端口模型图

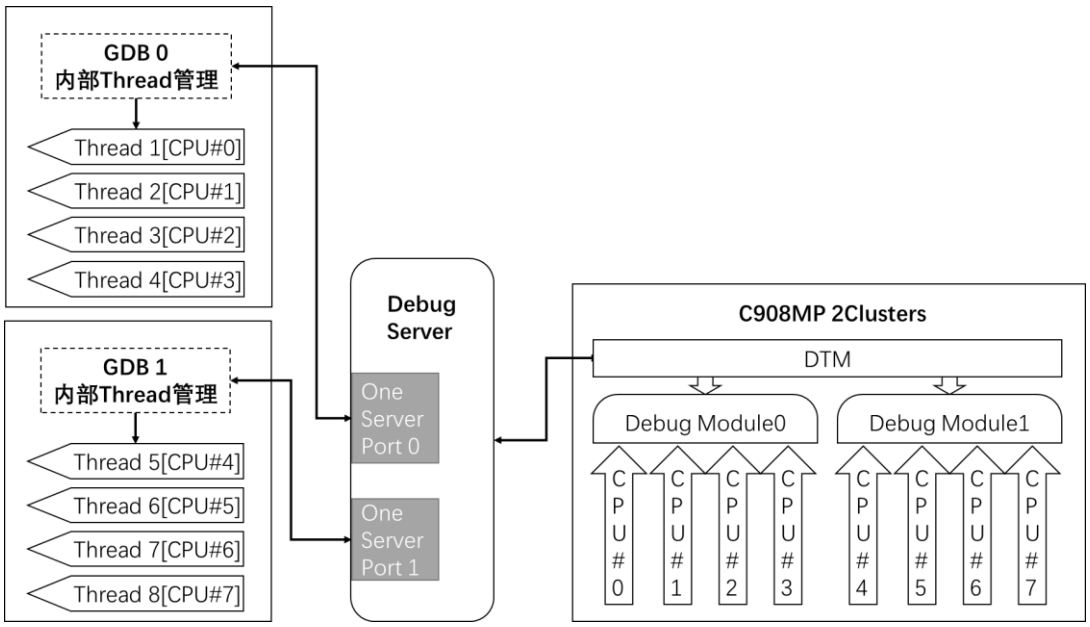



图 8-9 C908MP 多核多 Clusters 默认模型图

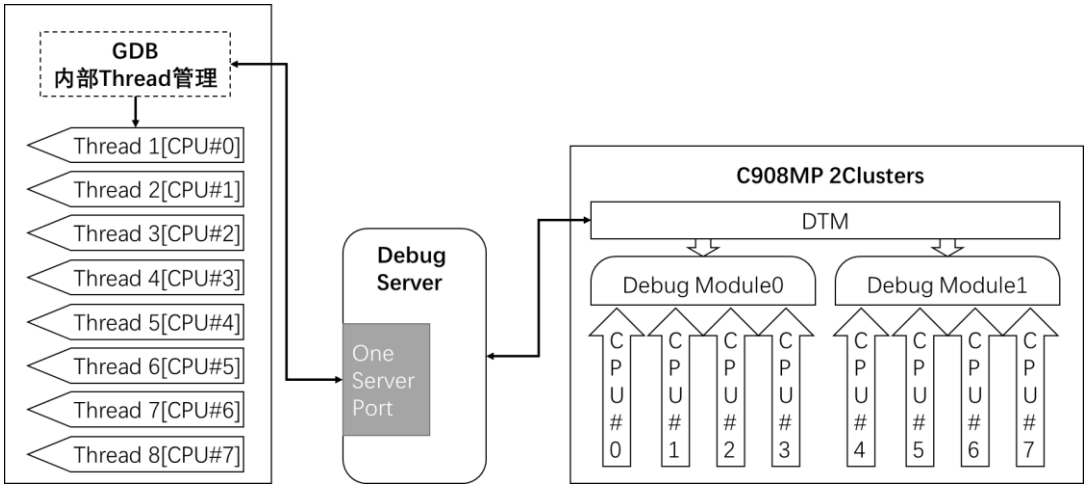



图 8-10 C908MP 多核多 Clusters -SMP 模型图

## 8.5.3. 操作步骤

**此处以 2 个核的 CK860MP 为例：**

1. 给开发板上电，连接 ICE，连接 ICE 的 USB 线连接至 PC 机。

2. 运行 DebugServer，显示界面如下：

(1) UI 版本界面

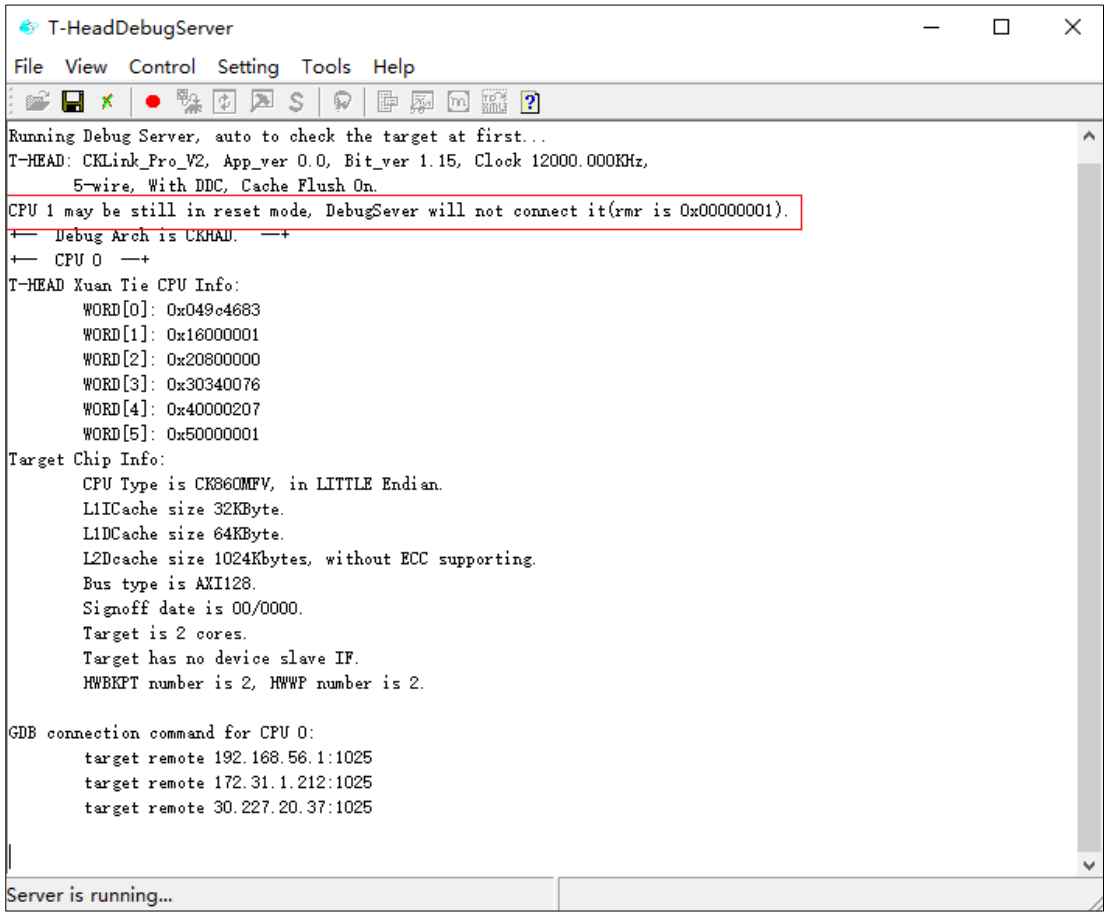



图 8-11 UI 版本 DebugServer 首次连接刚上电的 CK860MP

(2) Console 版本界面（linux 版本同）与 Windows UI 版本一致。

此时 DebugServer 正常打印 CPU 0 的 CPUID 信息，并打印 CPU 1 无法正常操作，该提示信息正常。原因是第一次上电的状态下，除过 CPU 0 之外的其他核均处于复位状态，故 JTAG 无法获取到 CPU 1 的信息，打印无法操作。

3. 启动 GDB 唤醒 CPU 1：

(1) 启动 csky-*abiv2*-gdb。

(2) 在 gdb 的命令行输入”target remote ip:port”(DebugServer 界面显示的连接命令)。

(3) 连接成功后，命令行输入：set $cr29 = 3, 唤醒 CPU 1 （cr29 的描述还请查阅 CK860MP 用户手册）。

```
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 ~]$
```

图 8-12 启动 GDB 并唤醒 CPU 1

(4) 命令行输入: quit，退出 gdb。

4. 重新启动 DebugServer，识别多核

(1) UI 版 DebugServer 操作：

① 点击暂停按钮：

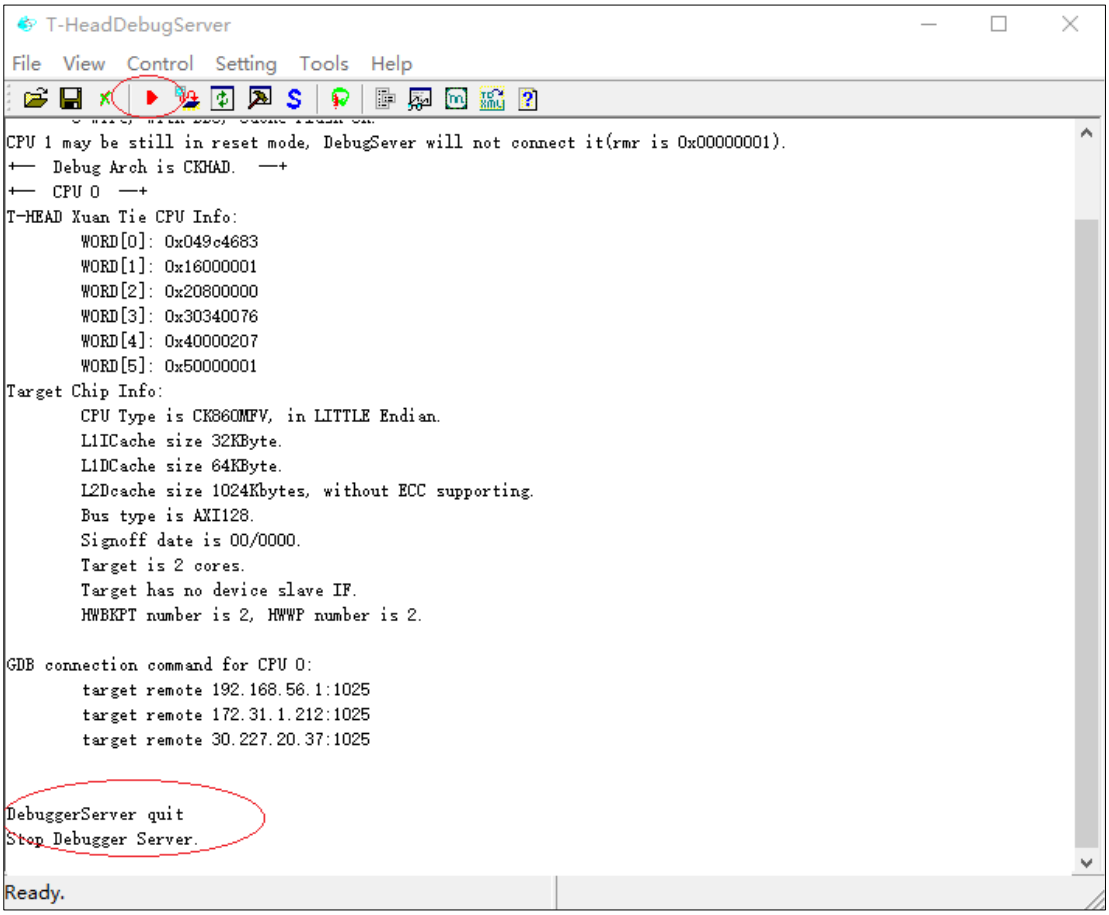



图 8-13 断开 UI 版本 DebugServer

② 重新连接，界面显示如下：

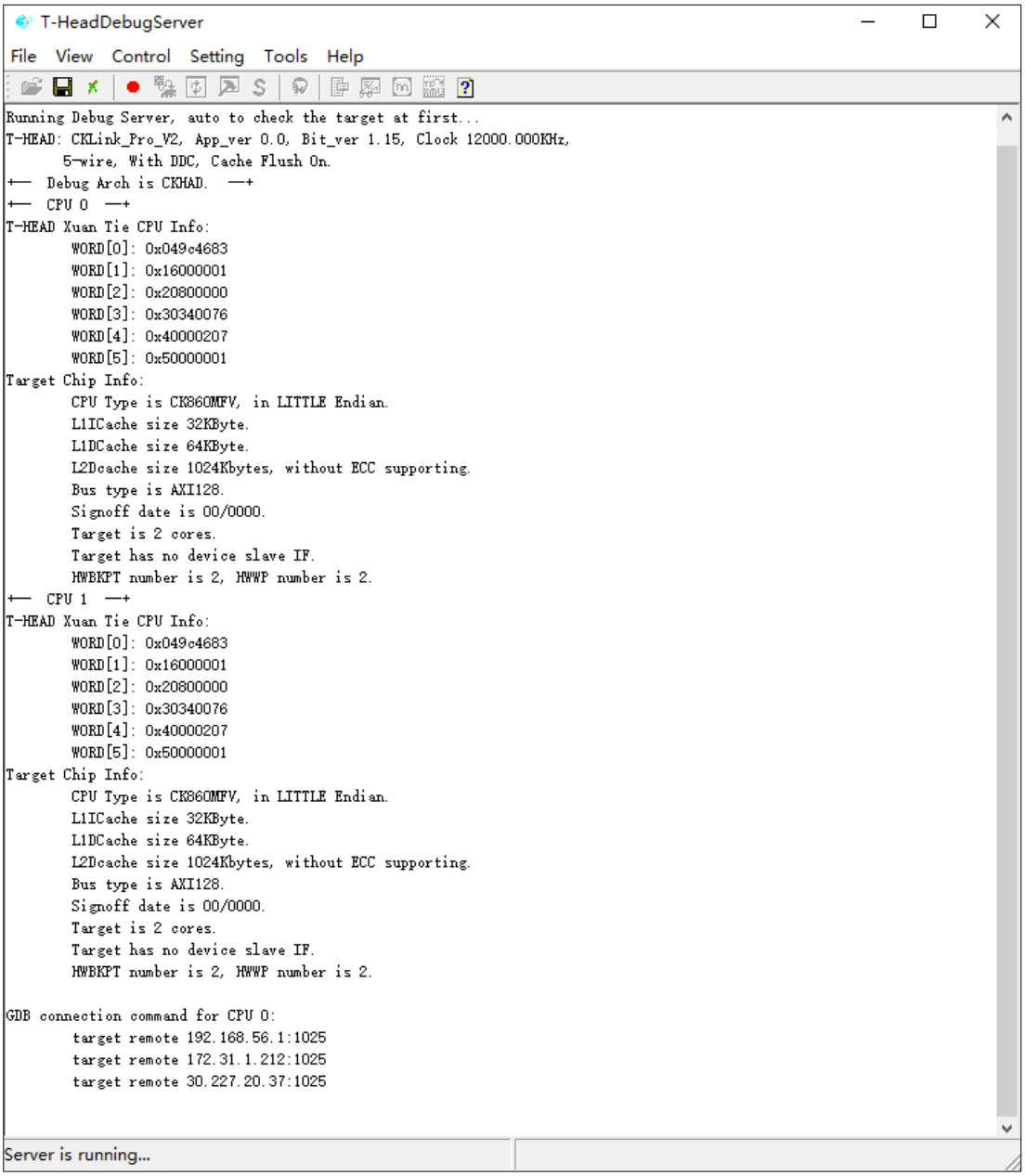



图 8-14 UI 版本 DebugServer 以多核单端口连接 CK860MP 显示

此时 DebugServer 将正常打印 CPU 0 和 CPU 1 的 CPUID 信息。

(2) Console 版 DebugServer 操作（linux 版本同）：

① 使用 Ctrl+c 结束 DebugServerConsole。

② 重新运行 DebugServerConsole。

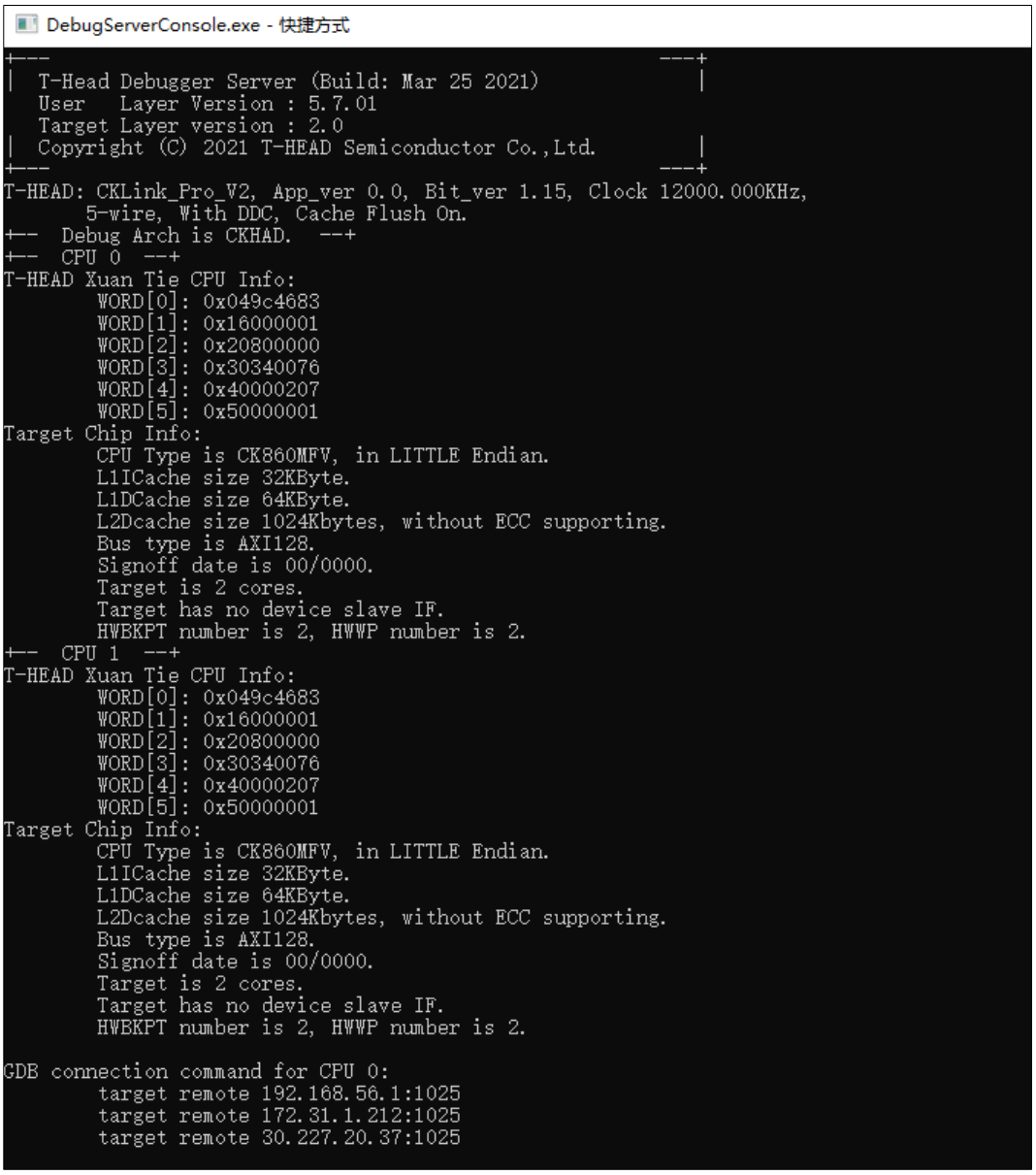



图 8-15 Console 版本 DebugServer 以多核单端口连接 CK860MP 显示（linux 同）

此时 DebugServer 将正常打印 CPU 0 和 CPU 1 的 CPUID 信息。

5. 启动 GDB，调试多核：

(1) 启动 csky-*abiv2*-gdb

(2) 在 gdb 的命令行输入”target remote ip:port”(DebugServer 界面显示的连接命令)

(3) 在 gdb 的命令行输入：info threads，显示

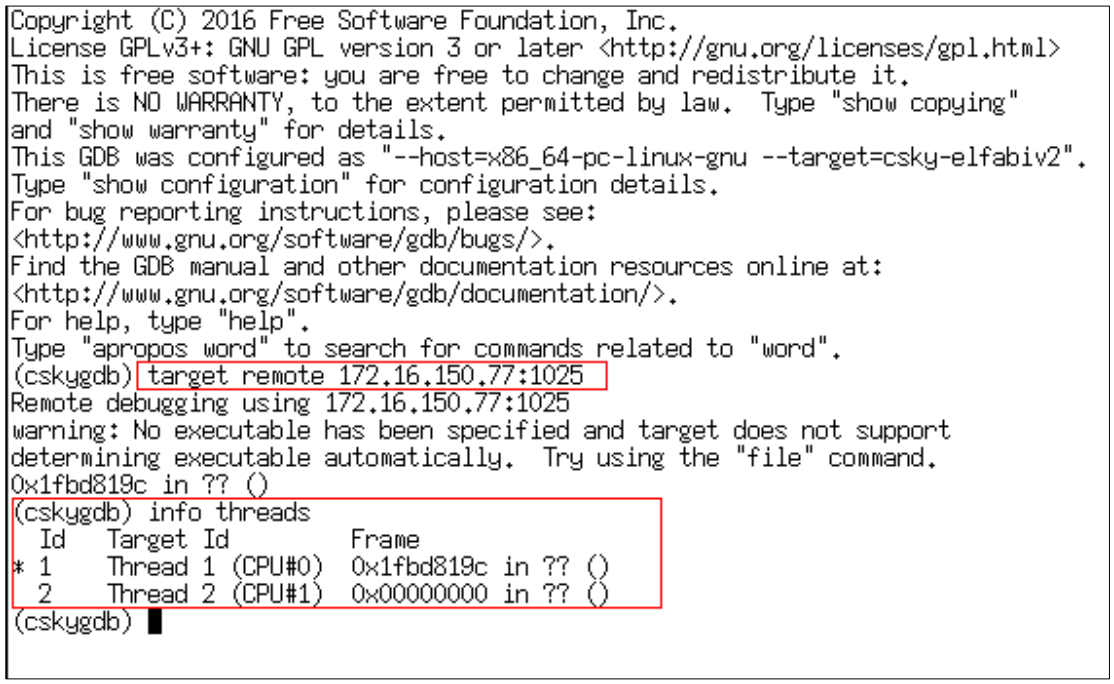



图 8-16 GDB info thread 查看线程

如上图所示，GDB 内部显示了两条线程 Thread 1 和 Thread 2，分别对应 CPU 0 和 CPU 1。

## 8.5.4. 线程操作

在【8.3.1】 操作的基础上，GDB 端 Thread 将与 CPU 一一对应，那么如果用户需要：

**1. 查看 CPU 1 的寄存器信息（以下为 gdb 命令行输入信息）**

(1) thread 2 (将 gdb 内部的线程切换至 Thread 2)。

(2) info registers (此时 gdb 将显示 CPU 1 的 GPR，PC，PSR 等寄存器信息)。

```
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)
```

图 8-17 GDB 端切换线程查看 CPU 1 的寄存器信息

**2. 查看 CPU 0 的寄存器信息**

(1) thread 1 (将 gdb 内部的线程切换至 Thread 1)。

(2) info registers (此时 gdb 将显示 CPU 1 的 GPR，PC，PSR 等寄存器信息)。

```
(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)
```

图 8-18 GDB 端切换线程查看 CPU 0 的寄存器信息

**3. 设置 CPU 0 的 PC 到 0x10000**

(1) 如果已经在 Thread 1，则用再输入 Thread 1 来切换（线程编号为 CPU 编号+1）。可以在 gdb 的命令行输入”thread”，或”info threads” gdb 将显示当前线程：

```
(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)
```

图 8-19 GDB 端查看当前线程

打印信息中，最前面带“*”的即为当前线程。

(2) set $pc = 0x10000 (设置 CPU 0 的 PC 为 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
```

图 8-20 GDB 端切换线程并设置 CPU 0 的 PC 至 0x10000

4. **所有的查看和修改操作需要对指定 CPU 进行查看寄存器，设置寄存器，查看 backtrace，内存等均需要在查看前确认是否为对应线程。**

5. **运行程序**: 由于该模式下调试信号相互响应，即用户在 gdb 的命令行输入 stepi，step，next，continue 等让程序运行的时候，该命令虽然是发给当前 CPU，但该退出调试模式的信号会被其他核响应。

(1) Si 时，所有 CPU 均会执行 si

(2) Step，continue 等命令时，当前线程对应的 CPU 退出调试模式，其他 CPU 均跟随退出调试模式。直到所有 CPU 中有一个 CPU 因为断点或其他原因进入调试模式后，其他的 CPU 将被第一个进入调试模式的 CPU 拉入调试模式。GDB 获取到 CPU 进入调试模式后，打印相关提示信息，并将当前线程切换至第一个进入调试模式的 CPU 对应的线程。

**6. 设置断点**

(1) 软件断点，该断点对所有 CPU 有效。

(2) 硬件断点，该断点对所有 CPU 有效。

(3) 如果只想某个 CPU 停止遇到断点停止时，可在设置断点时加入 Thread 信息，比如：

“break *0x10000 thread 2”。

即当 Thread 2（CPU 1）遇到该断点时，GDB 停止。

### 8.5.5. RISC-V 区别操作

对于类似使用 RISC-V DTM&DM 的 CPU，比如 C908MP，在操作步骤和线程操作是有一些区别的。

C908MP 单 DM 多核：在该模型中，只有一个 DM，一个 DM 内存在多个核，thserver 默认这些核为通过多核，会将一个 DM 内的多核看出七个整体。其他不同点：

1. 由于 Soc 的设计不同，唤醒多核的方式存在差异，请使用具体 Soc 设定来操作。

2. GDB 需要使用 riscv64-unknown-elf-gdb。

3. 其他操作类似。

C908MP 多 Clusters 多核：在该模型中，一个 DM 下的所有核为一个 Cluster，多个 DMs 即多个 Clusters。实际上，多个 Clusters 的多核在运行过程中是作为一个整体来运行系统的，但在调试接口上并没有信息来指明多个 Clusters 是作为一个整体的。所以：

默认处理方式：thserver 将会以 DM 为单位，将 DM 内得所有核作为一个整体，为其配置一个 GDB 调试的端口，GDB 连接上该端口后以线程的放 hi 来调试该 DM 内的所有核。DM 与端口绑定，一一对应。

-SMP 的处理方式：该方式即为指定多个 DM 为一个整体：

- UI 版本添加参数方式：
  - 修改 default.ini 中 OPTIONS=-smp cluster0:cluster1;
  - 在 Setting->Target Setting->Other flags 中输入-smp cluster0:cluster1
- Console 版本添加参数方式：-smp cluster0:cluster1

## 8.6. 多核多端口模式

在该模式下，DebugServer 将为每个 CPU 开启一个服务端口供 T-Head-GDB 连接， 以 port 作为区分。 T-Head-GDB 通过”target remote ip:port” 方式连接上 DebugServer 后，命令中 port 指定后即指定连接该 port 对应的 CPU。对应关系在 DebugServer 的界面上可以看到。

该模式下，多个 CPU 将不互相发送并不响应其他核的调试信号，意为被调试的核独立，其中一个 CPU 进入或退出调试模式，都不会影响其他核的运行。可以认为用户在调试多个开发板，只是多个开发板的内存是一份。

### 8.6.1. CK860MP 多核调试模型

模型如下：

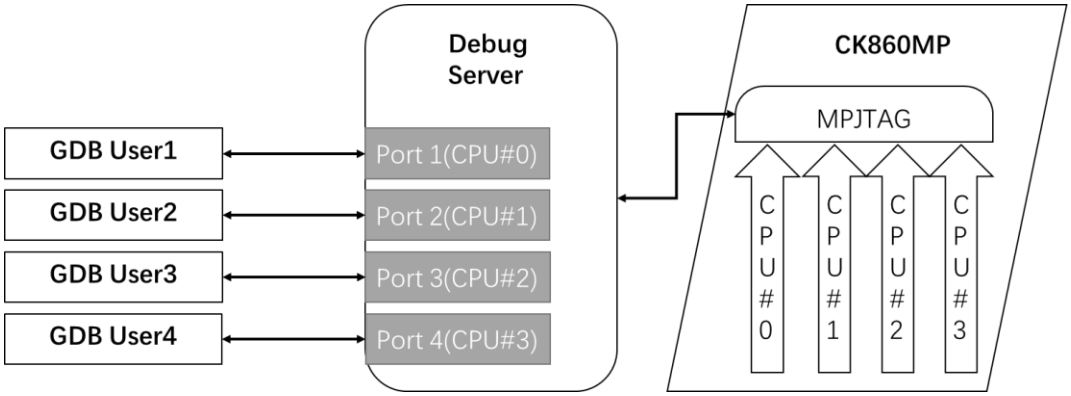



图 8-21 CK860MP 多核多端口模型图

### 8.6.2. C908MP 多核调试模型

模型如下：

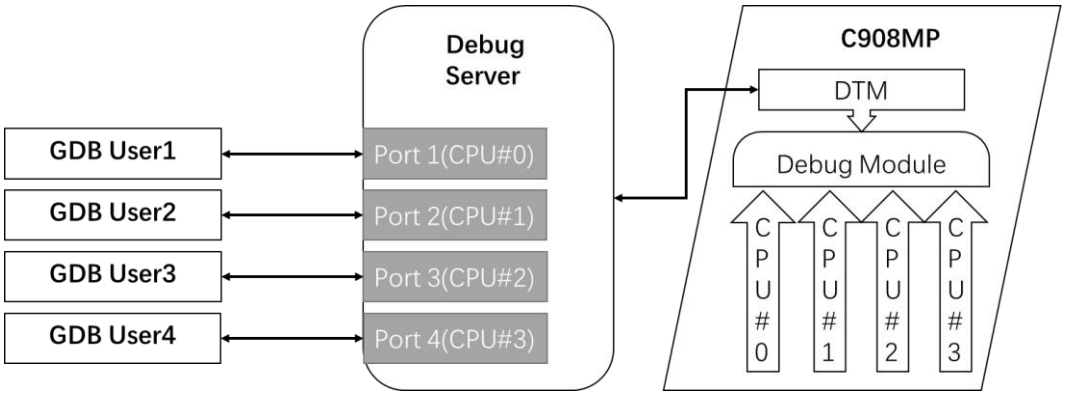



图 8-22 C908MP 多核多端口模型图

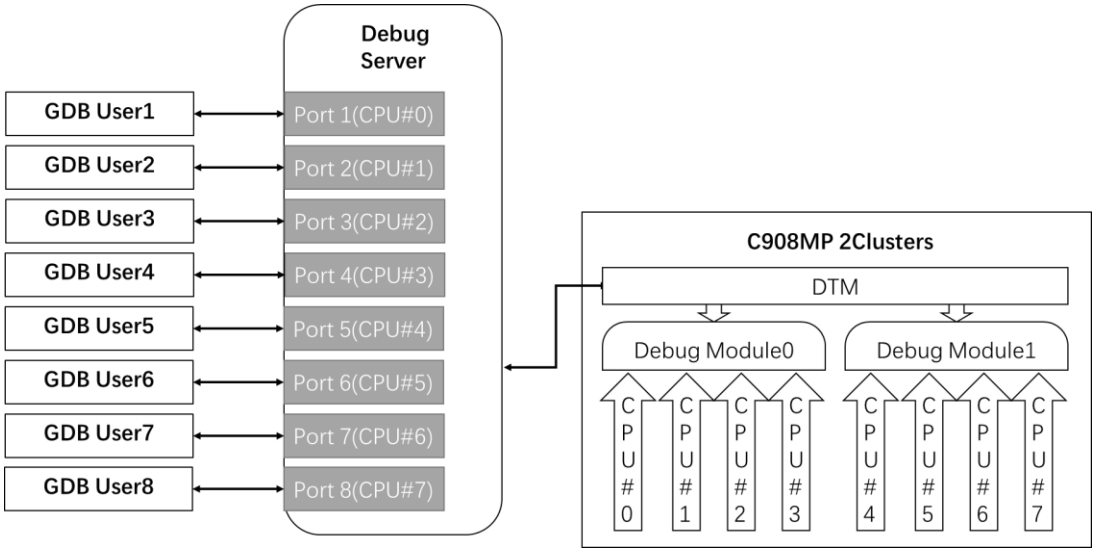



图 8-23 C908MP 多核多 Clusters 多端口模型图

## 8.6.3. 操作步骤

此处以 2 个核的 CK860MP 为例：

1. 与【操作步骤】的 1 操作一致。

2. 与【操作步骤】的 2 操作一致。

3. 与【操作步骤】的 3 操作一致。

4. 重新启动 DebugServer，识别多核：

(1) UI 版本 DebugServer

① 可选择先启动 DebugServer，运行起来后暂停 server。点击菜单中的”Setting->Multicore Threads”，当该选择如下图时，再次启动 Server：

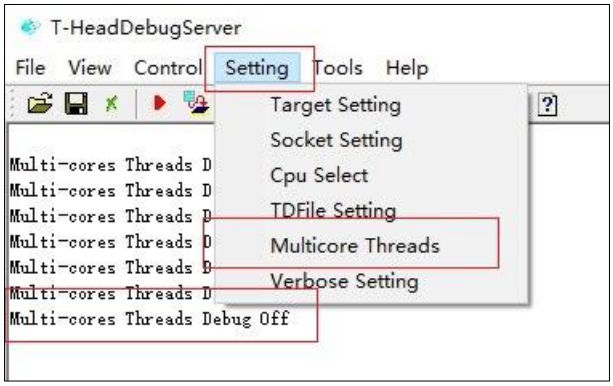



图 8-24 UI 界面设置 -no-multicore-threads 模式

② 也可选择修改 DebugServer 程序所在目录的 default.ini 中的“MULTICORETHREADS=TRUE”为“MULTICORETHREADS=FALSE”，然后启动 DebugServer。

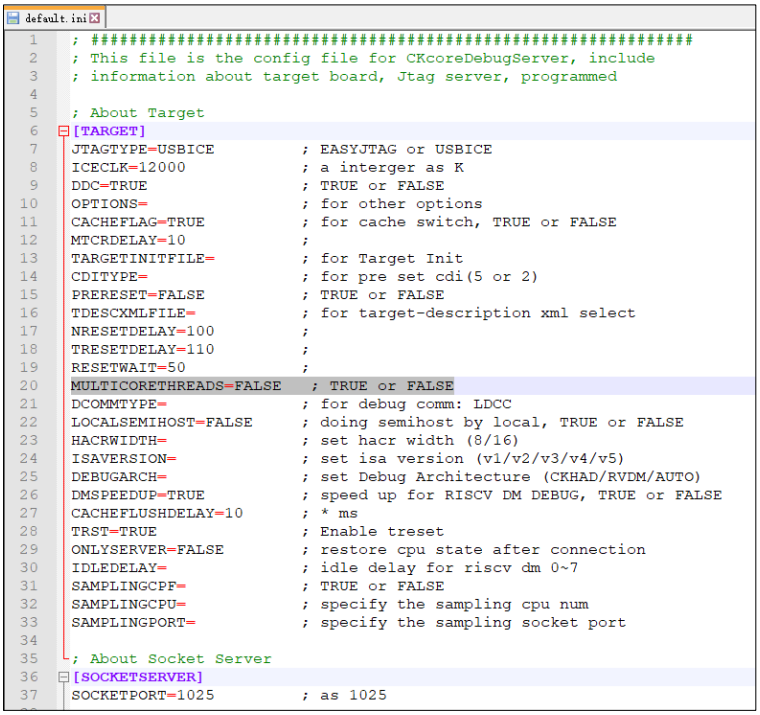



图 8-25 UI 通过 default.ini 设置 -no-multicore-threads 模式

③ 启动 DebugServer 后如下图：

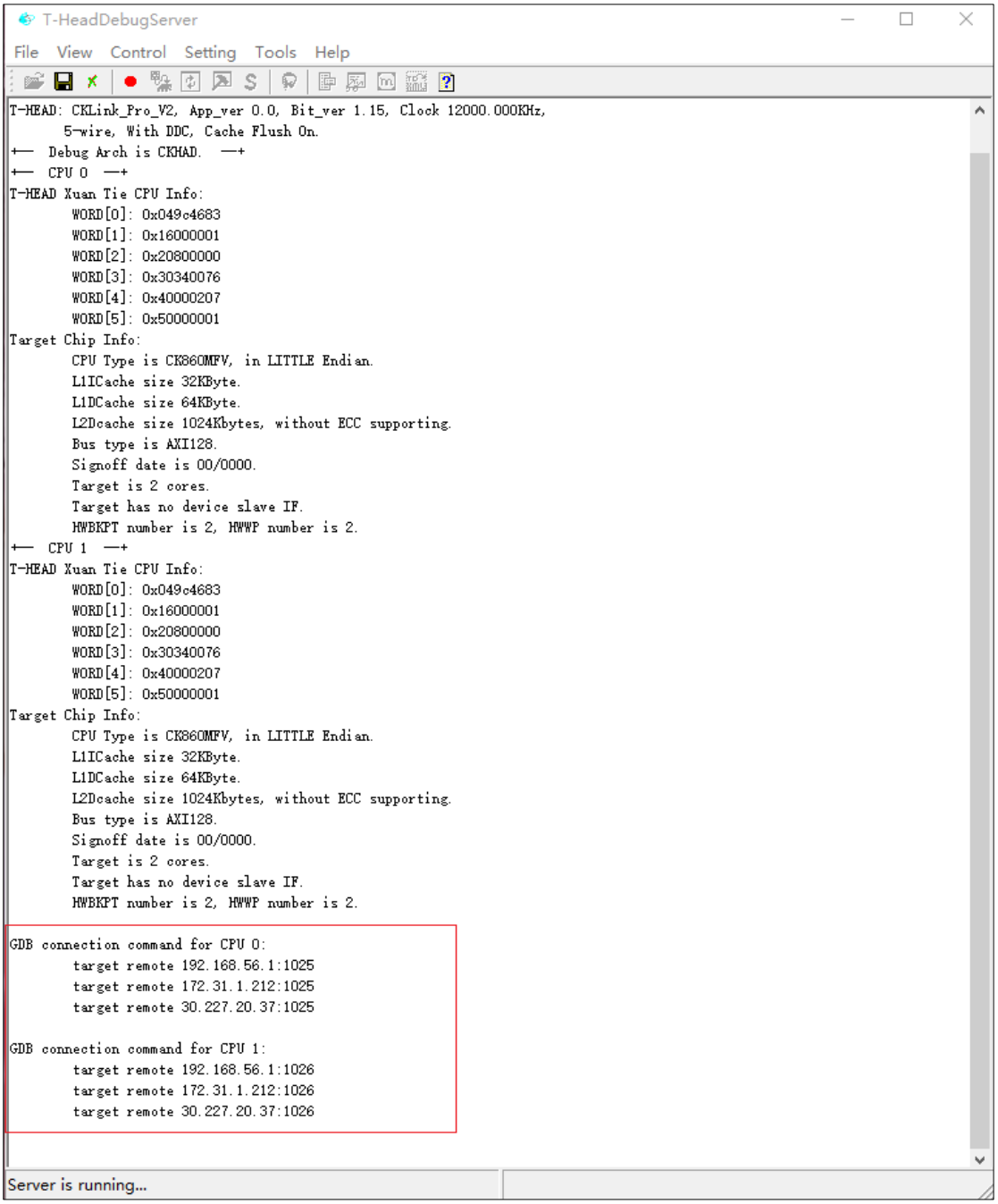



图 8-26 UI 版本 DebugServer 已多核多单端口连接 CK860MP 显示

(2) Console 版本 DebugServer（linux 版本同）：

启动 DebugServerConsole 加启动参数： -no-multicore-threads。

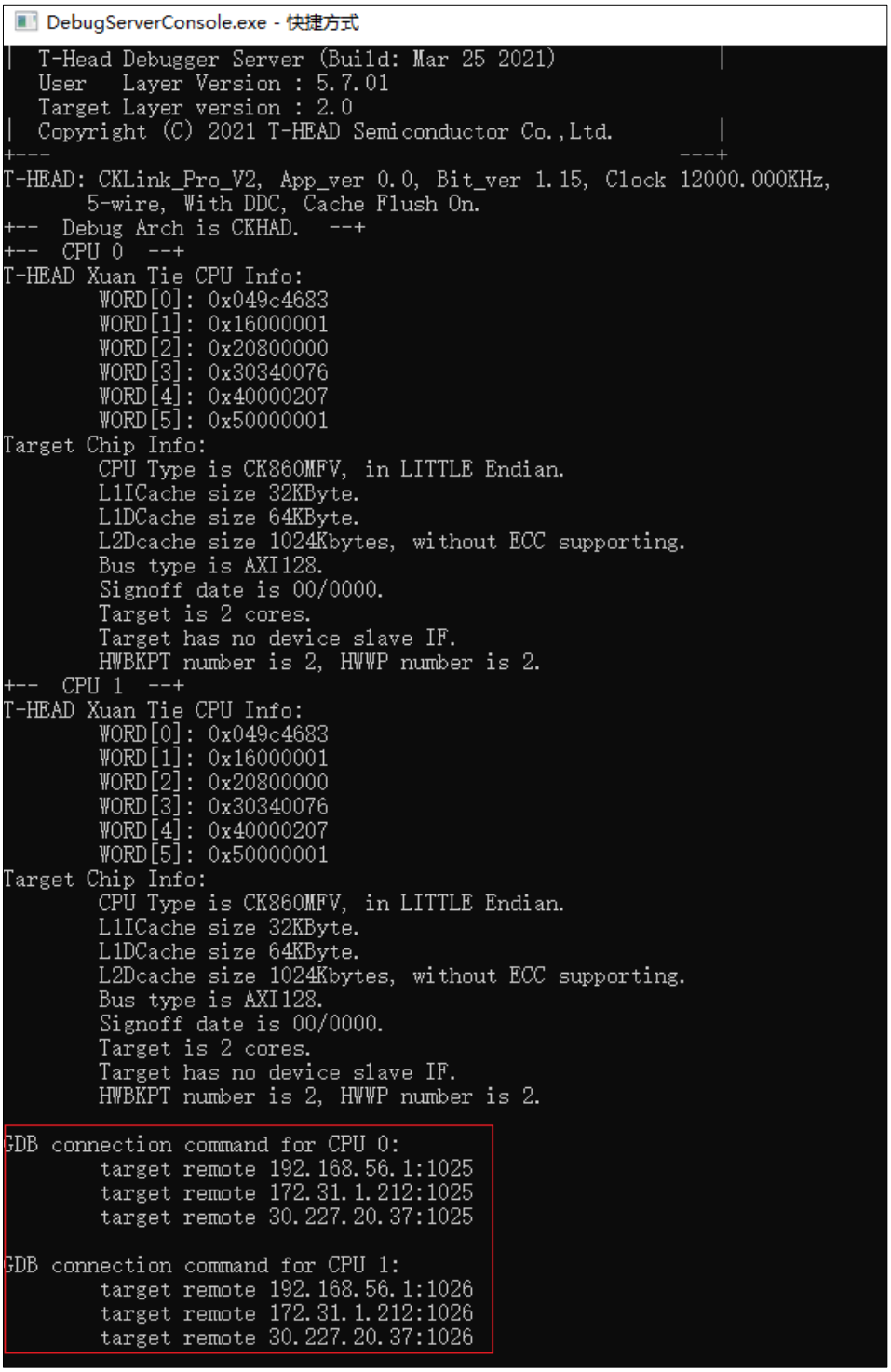



图 8-27 Console 版本 DebugServer 以多核多单端口连接 CK860MP 显示（linux 同）

5. GDB 连接，调试：

(1) 此时，DebugServer 界面上已经显示了连接 CPU 0 和 CPU 1 的连接命令，IP 相同，

Port 做区分。

(2) 此时 GDB 调试可认为在调试单核，与单核调试无异（调试者需要知道此时各个核是共享内存的即可）。

### 8.6.4. RISC-V 区别操作

多核多端口模式是以核的个数为单位的，此处主要的区别是：

1. 由于 Soc 的设计不同，唤醒多核的方式存在差异，请使用具体 Soc 设定来操作。

2. GDB 需要使用 riscv64-unknown-elf-gdb。

3. 其他操作类似。

## 8.7. 多核调试通用规则

针对于 HAD 调试架构：

1. 在默认或 multicore-threads 模式下，thserver 可以识别到的 SMP 默认开一个端口，异构的部分则开启独立的 GDB 调试端口。

2. 在-no-multicore-threads 模式下，以核的个数为单位，开启对应个 GDB 调试端口，核与端口绑定，GDB 连接指定端口，即为调试指定核。

针对于 DTM&DM 调试架构：

1. 在默认或 multicore-threads 模式下，thserver 以 DM 为单位，为每一个 DM 开启一个 GDB 调试端口。DM 内的所有核默认为 SMP，即在 T-HEAD 的设计中，一个 DM 内不会出现多种型号的 CPU。

2. 在默认或 multicore-threads 模式下，多 Clusters 的多核（每个 Cluster 分布在不同的 DM 内），可以通过-smp 参数将多个 clusters 看成一个整体。被看作一个整体的多个 DMs 会只开启一个 GDB 调试端口，剩余的 DM 被单独看作一个整体，拥有对应的 GDB 调试端口。

3. 在-no-multicore-threads 模式下，以核的个数为单位，开启对应个 GDB 调试端口，核与端口绑定，GDB 连接指定端口，即为调试指定核。

# 9. Flash 烧录及 Flash 断点

Flash 烧录在 MCU 的调试中极为常见。大部分 MCU 的 flash 对于 CPU 来说都是存在地址映

射的，因此其 code 也可以在 flash 上运行。因此 DebugServer 将引入 Flash 烧录的功能。

## 9.1. Flash 烧录原理

Thserver 中 flash 烧录和 CDK 中的 flash 烧录原理保持一致。

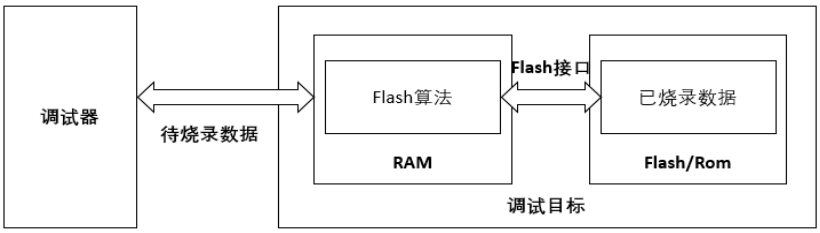



图 9-1 调试器烧录 Flash 图示

如上图所示，调试器通过 RAM 区域的 Flash 算法文件，对 Flash/ROM 区域进行擦除和烧录功能。详细信息参考以下链接中的 5.2， 5.3， 5.4 章节：

https://occ.t-head.cn/development/series/video?spm=a2cl5.25410618.0.0.4a53180f137C4G&id=3864775351511420928&type=kind&softPlatformType=4#sticky

### 9.1.1. 算法文件要求

由于在 thserver 进行下载和设置 Flash 断点时不可破坏硬件现场，故需要知道：

1. 算法文件会破坏的寄存器。寄存器涉及 GPRS，PC 和包含中断使能控制位的控制寄存器，这个对 T-HEAD 的算法文件无要求。

2. 算法文件会破坏的内存区域。内存涉及 Flash Algorithm 程序代码执行内存和堆栈使用内存，代码执行内存可获取，但堆栈使用内存将对算法文件提出要求，需要算法文件的版本大于等于 6。

要求算法文件使用 CDK 模板，且版本大于等于 6。 基本要求：

1. 堆栈使用区间被包含在 .bss 段内
2. __bkpt_label 处为软件断点指令
   a. 900 系列 CPU 的为：

```
__bkpt_label:
ebreak
ret
```

   b. 800 系列 CPU 的为：

```
__bkpt_label:
bkpt
rts
```

3. 存在__continue_label ,

a. 900 系列 CPU 的为：

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

b. 800 系列 CPU 的为：

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

更多信息参考 CDK Flash 模板工程。

## 9.1.2. 命令行支持的 Flash 操作命令

Thserver 的命令行中实现对 Flash 操作的相关功能，集成命令如下：

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

flash：指定 flash 算法文件，参数：

- -al/--algorithm file，指定 flash 算法文件。
- -cov/--compat-algorithm-ver1to5，支持 v1~v5 版本的 flash 算法文件。

flash info：查看当前 Flash 算法文件的相关信息，无其他参数。

flash program：执行 flash 烧录操作，参数：

- -al/--algorithm file， 指定烧录时使用的 flash 算法文件，默认使用 thserver 启动时指定 flash 算法文件，或通过 flash -al 指定的算法文件。
- -f/--file，指定烧录的文件。
- -b/--binary，如果烧录的文件是 bin 文件，需要该参数指明。
- -a/--address ADDR，如果烧录的文件是 bin 文件，该参数指令烧录的起始位置。
- -v/--verify，烧录后验证 flash 是否烧录成功。
- -cov/--compat-algorithm-ver1to5，支持 v1~v5 版本的 flash 算法文件。
- -fi/--flash-index，如果存在多块 flash 区域拥有相同的地址段，可配合使用该参数。

flash erase：对 flash 进行擦除操作，参数：

- -al/--algorithm file， 指定擦除使用的 flash 算法文件，默认使用 thserver 启动时指定 flash 算法文件，或通过 flash -al 指定的算法文件。
- -c/--chip，指明进行片擦。
- -a/--address ADDR，不加-c/--chip，即不是片擦时指明擦写的起始位置。
- -s/--size LENGTH，不加-c/--chip，即不是片擦时指明擦写的长度。
- -cov/--compat-algorithm-ver1to5，支持 v1~v5 版本的 flash 算法文件。

flash dump：转储 flash 内容到文件，参数：

- -al/--algorithm file， 指定转储使用的 flash 算法文件，默认使用 thserver 启动时指定 flash 算法文件，或通过 flash -al 指定的算法文件。
- -o/--output，指明转储到的文件名。
- -b/--binary，指明转储的文件类型为 bin 文件。
- -h/--hex，指明转储的文件类型为 hex 文件。
- -a/--address ADDR，指明转储的起始位置。
- -s/--size LENGTH，指明转储的长度。
- -cov/--compat-algorithm-ver1to5，支持 v1~v5 版本的 flash 算法文件。

flashbp-clear: 恢复 Flash 内存中已插入或还未删除的 flash 断点。因为调试过程中为了优化加速会使用指令模拟，或仅在适当的时机才删除 flash 断点，以减少对 flash 的操作。这样在调试过程中就会隐含当前 flash 内存中存在已插入或还未删除的 flash 断点。 该命令将会删除这些断点，恢复 flash 中原有的值。

**示例：**

*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 断点

在了解 Flash 断点之前，先介绍以下断点的分类：

- 软件断点
- 硬件断点
- 数据观察点

软件断点：CPU 的指令集中包含一条断点指令，当 CPU 执行到该指令时，产生调试异常，CPU 会进入调试模式。通俗的讲就是 CPU 运行会停住。软件断点就是利用该指令的特性，调试器通过 load/store 替换断点位置的原有指令，从而使断点生效。

硬件断点：CPU 内部包含地址比较器单元，当设置硬件断点时，配置比较器的取址 PC 和断点地址进行比较。当取址 PC 和断点地址匹配时，则使 CPU 进入调试模式，即硬件断点生效。由于比较器的资源占用比较大，一般 MCU 中的硬件断点比较器都有个数限制。

数据观察点：CPU 内部的数据地址比较器，和硬件断点的比较器类似。数据观察点比较器比较的对象是 load/store 的地址和数据观察点地址。当比较器匹配时，则使 CPU 进入调试模式，即硬件断点生效。一般来说，硬件上的数据地址比较器和取址比较器为同一单元。

在 MCU 领域，往往 CODE 都是在 flash 上，当对 CODE 打软件断点时，由于软件断点是通过 load/store 指令改写指令码，由于 CODE 是 flash，无法通过 load/store 改写指令码。从而只能使用硬件断点。而硬件断点由于个数限制，限制了开发者的调试工作。

Flash 断点由此而来。Flash 断点的原理和软件断点类似，同样采用断点指令使 CPU 进入调试模式，不一样的地方在于软件断点采用 load/store 改写指令码，而 Flash 断点是擦写 flash 改写指令码。

### 9.2.1. 工作原理

前面说到 Flash 断点是通过擦写 flash，改写指令码的方式。因此 thserver 会记录用户使用的断点信息，在需要新插入断点的需求的时候，对 Flash 进行擦除和烧录，替换原指令为 bkpt/ebreak 软件断点指令。烧录原理遵从 9.1Flash 烧录原理章节描述的 Flash 烧录原理。

当用户需要执行断点处命令的时候，thserver 集成了模拟执行单元，模拟其中部分指令。若无法进行模拟的指令，会通过 sp 位置写指令码由 CPU 来执行。

### 9.2.2. 断点的效率

在调试过程中，由于 Flash 断点需要对片上 flash 进行擦写操作，相比于软件断点，增加了时间的成本。为了减少 flash 操作，thserver 中进行一系列的优化，包括但不限于硬件断点的联合使用、指令模拟等。

为了保持数据的一致，在 Flash 断点的烧录前，DebugServer 会保存算法文件使用的内存区域。为了提高效率，我们建议：Flash 算法文件 **code size** 和 **data size** 应尽量的小。

### 9.2.3. 使用方法

准备内容：

1. 一块拥有 Flash 的开发板。
2. 使用 CDK 的 Flash 驱动模板工程生成的 Flash 算法文件，要求 Flash 驱动模板版本号>=6。
3. 其他软硬件工具。

操作过程：

1. 开发板上电，连接 CKLINK。
2. 如果使用 Console 版本 thserver，启动时加参数：-flash-algorithm flash_algorithm_path。
3. 如果使用 GUI 版本 thserver, 使用 Setting->Flash Setting 来指定 Flash 算法文件。
4. 启动 GDB 进行连接 thserver 进行调试。

使用上述功能时仅可能的使用较新版本 thserver，可从 https://xuantie.t-head.cn/community/download?spm=a2cl5.14290816.0.0.5911UscjUscjDl 下载。

使用的 CDS 或 CDK，请参考 CDS 或 CDK 的用户手册说明。

预期结果：用户可以使用 GDB 在 flash 区域内设置断点（不区分软件断点与硬件断点）进行调试。此时，如果在 flash 区域的断点个数未超出 CPU 已有的硬件断点个数，则 thserver 将硬件断点来进行调试；如果已超出，thserver 将会自动选择，将超出的部分断点使用

Flash 断点进行插入。

# 10. Vendor ICE 支持

在业务的发展过程中，我们的客户具备了开发设计 Link 的能力。在 DebugServer 应对客户自己设计开发的 Link。因此 DebugServer 增加了客户设计开发的 Link 功能。

该功能的实现方式是：

安装目录下的 links 目录中存放着多个 Vendor 目录，每个目录中保存的客户实现的 Link 库文件。库文件需要实现一些接口用于 porting DebugServer 完成调试功能。

porting 接口见表格 10-1 Link Porting 接口列表，详细也可见示例程序的 Include/link.h

| 接口 | 接口原型 | 描述 |
|---|---|---|
| link_init | Int link_init (dbg_server_cfg_t *cfg); | 初始化 link<br>option |
| link_open | void*link_open(dbg_server_cfg_t *cfg, void *unique); | 打开 link 设备 |
| link_close | void link_close (void *handle); | 关闭 link 设备 |
| link_config | int link_config (void *handle, enum LINK_CONFIG_KEY key, unsigned int value); | 配置 link，包括频率、复位时间等<br>具体配置项可 option |
| link_upgrade | int link_upgrade (void *handle, const char *path); | 固件升级<br>option |
| link_memory_read | int link_memory_read (void *handle, uint64_t addr, int xlen, uint8_t *buff, int length, int mode); | 读 target 内存 |
| link_memory_write | int link_memory_write (void *handle, uint64_t addr, int xlen, uint8_t *buff, int length, int mode); | 写 target 内存 |
| link_register_read | int link_register_read (void *handle, int regno, uint8_t *buff, int nbyte); | 读 target 通用寄存器 |
| link_register_write | int link_register_write (void *, int regno, uint8_t *buff, int nbyte); | 写 target 通用寄存器 |
| link_jtag_operator | int link_jtag_operator (void | 执行一次 JTAG 操作 |

| | | |
|---|---|---|
| | *handle, int ir_len, unsigned char *ir, int dr_len, unsigned char *dr_r, unsigned char *dr_w, int read); | option |
| link_gpio_operator | int link_gpio_operator (void *, int gpio_out, int *gpio_in, int gpio_eo, int gpio_mode); | 执行一次 GPIO 操作<br>option |
| link_show_info | int link_show_info (void *, dbg_server_cfg_t *cfg, void (*func)(const char *, ...)); | 显示 link 信息。<br>option |
| link_reset | int link_reset (void *handle, int hard); | 复位 link |
| link_get_device_list | int link_get_device_list (struct link_dev *dev, int *count); | 获取 link 列表 |
| THE_NAME_OF_LINK | const char * THE_NAME_OF_LINK (void); | LINK 名字，该接口作为 DLL 识别接口，必须实现。 |

表格 10-1 Link Porting 接口列表

# 11. Example 工程

针对不同的应用场景，客户某些时候需要自己实现一些操作去控制或者读写芯片信息。因此我们提供一个示例工程，告知客户如何通过 target 接口去操作芯片。

在 T-Head DebugServer 工具的安装目录下存在一个 Example 目录，在 linux 和 windows 下分别有不同的使用方式。

linux 下，我们提供提供 Makefile 编译工程。依赖工具：make, gcc, g++。

windows 下，我们提供 VS 工程。依赖工具： Virtual Studio。

打开工程，源文件中我们可以看到一些操作。包含：

1) 初始化过程及连接目标板
2) 寄存器读写操作
3) 内存读写操作
4) 运行/停止等
5) 断开连接

详细信息请打开工程查看。

# 12. 常见问题及解决方法

表格 12-1 常见问题及解决方法

| 问题类型 | 错误提示 | 解决方法 |
| --- | --- | --- |
| ICE 连接失败 | ERROR: No C-SKY ICE connected to Your PC or your C-SKY ICE driver not installed correctly!<br>Input enter to exit... | 确保 ICE 重新连接并且 ICE 驱动成功安装 |
| Target 连接失败 | ERROR: Fail to enter debug mode!<br>Error: Can't enter debug mode, please check the target board physical link.<br>Input enter to exit... | ICE 连接好目标板，并且保证目标板上电，重新连接 |
| 端口绑定失败 | ERROR: Fail to bind socke port 1025, please change another one.<br>ERROR: Fail to create socket server.<br>Input enter to exit... | 打开端口设置菜单选项卡，重新选择端口号，再次连接 |
| 版本升级问题 | WARNING: ICE Upgrading ignored, it may cause function lose or error!!!<br>ERROR: DebugServer can't implement ICE config!<br>Input enter to exit... | 重新打开 thserver 连接 ICE，出现 ICE 升级提示时，选择‘y’，升级成功后，重新插拔 ICE，再次使用 thserver 进行连接即可。 |
| ICE 与 thserver 版本兼容问题 | ERROR: You ICE's version is newer than Your DebugServer, Please update Your DebugServer!<br>ERROR: DebugServer can't implement ICE config!<br>Input enter to exit... | 此问题针对 cklink_lite 型号 ICE，thserver 版本过老会出现此问题，应该从 T-Head 官网下载更新最新版本的 thserver |