OpenOCD 和 GDB

简介与使用

版本：*1.0*

www.bouffalolab.com

Contents

1 芯片调试 4
1.1 JTAG 4
1.2 调试环境 5
2 RISC-V 调试部件 6
3 OpenOCD 7
3.1 简介 7
3.2 FT2232 7
4 GDB 8
5 调试实战 9

List of Figures

1.1 JTAG 接口连接 5
1.2 调试环境示意图 5

2.1 RISC-V 调试系统框架 6

5.1 查看寄存器 11
5.2 使用效果 13
5.3 调用栈效果 13
5.4 切换线程 2 效果 14
5.5 打印线程 2 效果 14
5.6 使用 mdw 效果 15

1 芯片调试

芯片调试的目的是为了快速的分析及解决软硬件出现的问题。

调试的手段有：硬件模拟器 (如早期 8031)，硬件仿真器，打 log 等等。微处理器调试的历史参考 A history of uP debug, 1980 – 2016。新的处理器增加了片上调试部件 (OCD/ICE)，可用于实现在线仿真。

调试部件的硬件接口：

- 单线（debugWIRE）：为了降低成本和调试引脚的开销，仅使用一根信号线（RESET），即可完成调试信息的交互，达到控制程序流向，执行指令以及编程熔丝位的功能
- 双线（SWD）：SWDIO–串行数据线；SWDCLK–串行时钟线
- 四线（JTAG）：TCK–测试时钟输入；TDI–测试数据输入，数据通过 TDI 输入 JTAG 口；TDO–测试数据输出，数据通过 TDO 从 JTAG 口输出；TMS–测试模式选择，TMS 用来设置 JTAG 口处于某种特定的测试模式

1.1 JTAG

JTAG(Joint Test Action Group，联合测试工作组) 是一种国际标准测试协议（IEEE 1149.1 兼容），主要用于芯片内部测试。现在多数的高级器件都支持 JTAG 协议，如 DSP、FPGA 器件等。标准的 JTAG 接口是 4 线：TMS、TCK、TDI、TDO，分别为模式选择、时钟、数据输入和数据输出线。还有一个可选引脚 TRST，用来测试复位，是输入引脚，低电平时有效。

JTAG 接口可以一对一的使用，也可以组成菊花链的一对多拓扑结构，拓扑结构如下图所示：

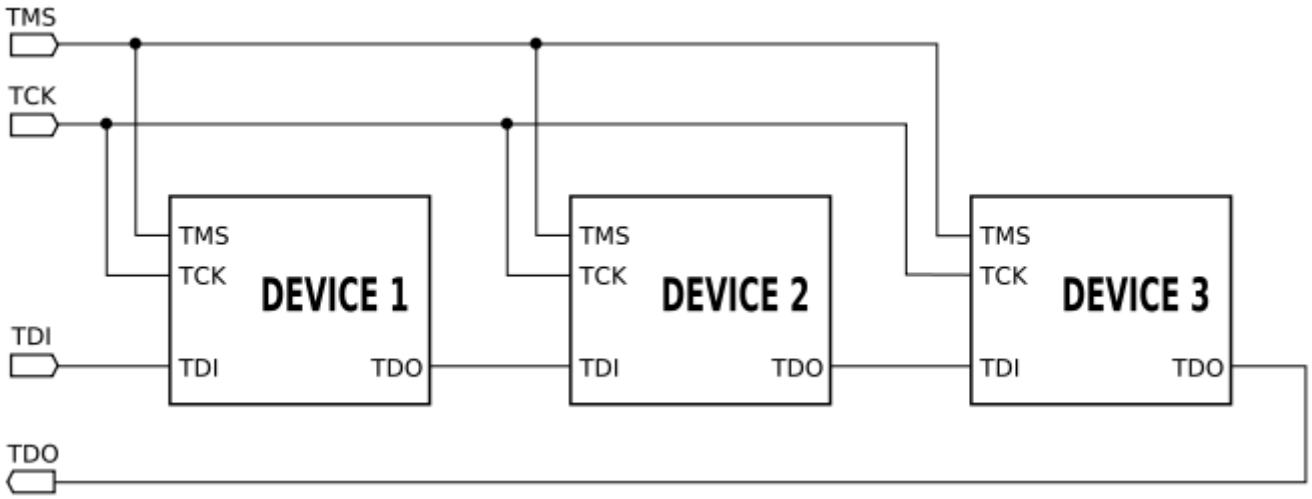

图 1.1: JTAG 接口连接

1.2 调试环境

- PC 上的调试器 (GDB)
- 软件调试的代理 (OpenOCD)
- 硬件调试代理/适配器 (FT2232)
- 调试目标 (RISC-V)

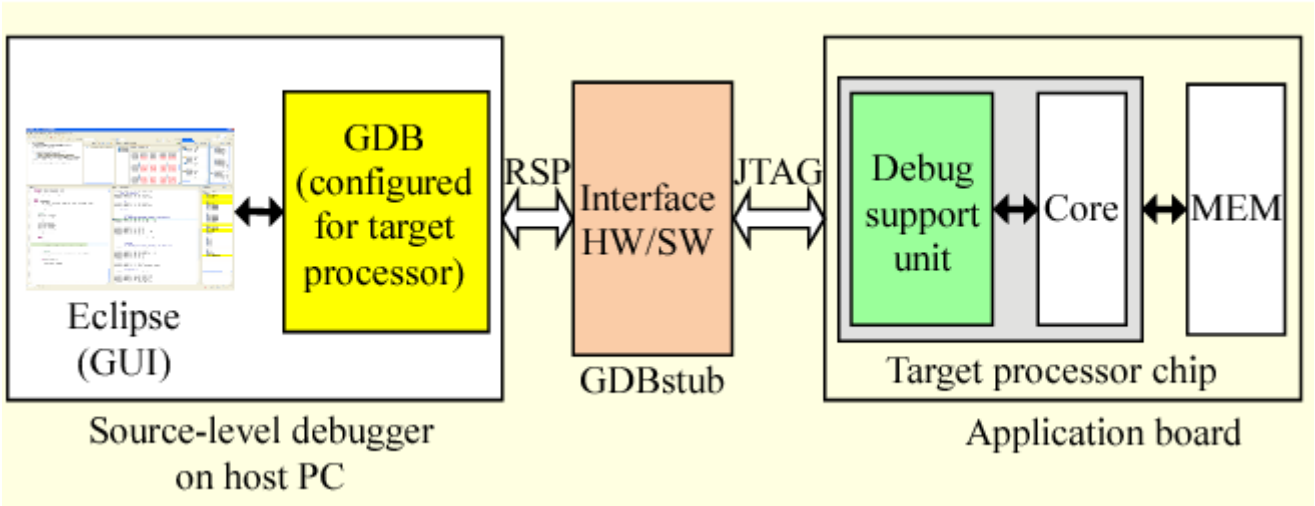

图 1.2: 调试环境示意图

2 RISC-V 调试部件

框架一共分为 3 个部分，分别是 Debug Host，例如 PC；Debug Transport Hardware，例如 JLink 或者 CMSIS-DAP 等的调试工具；第三部分是嵌入在芯片内部的调试模块。在调试模块内部，DTM 模块与调试工具直接进行交互，它通过 DMI 接口与 DM 模块进行交互。

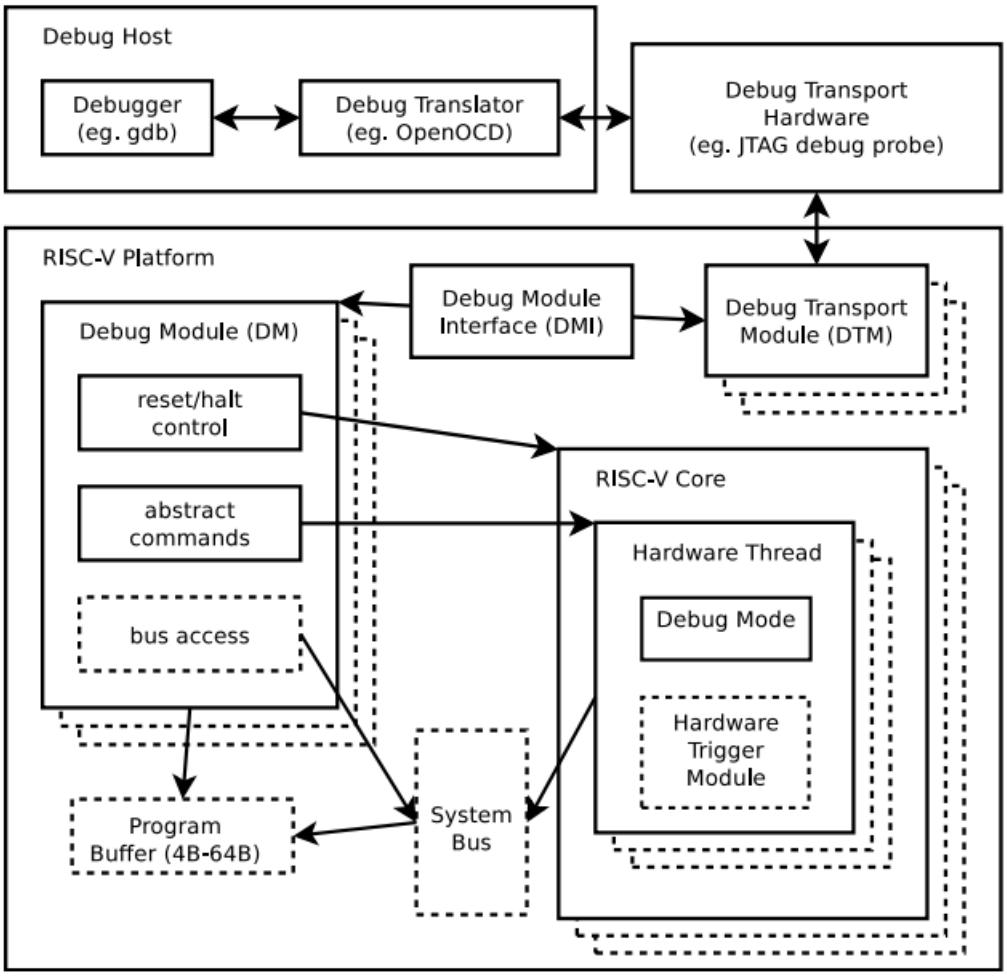

图 2.1: RISC-V 调试系统框架

3 OpenOCD

3.1 简介

OpenOCD 是一个开源的软件调试代理，可以支持各种架构处理器的调试和烧写；支持各种调试硬件代理/适配器：FTDI, J-link 等。OpenOCD 接收来自 GDB 的命令 (socket)，再通过硬件代理与芯片上的调试部件通信以完成各种调试任务。OpenOCD 还有一个 telnet 服务器，可以进行一些独立的操作。具体可参考 https://github.com/ntfreak/openocd

3.2 FT2232

FT2232 是 FTDI 生产的一系列 USB 转串口的桥接芯片中的一颗。其中的第一个数字 2，表示芯片有 2 个转串口组件；尾缀字母代表规格 (速度等)。其特殊之处在于其内部每个串口组件都实现了 MPSSE，可以实现 USB 转 JTAG，因此被广泛应用在了 JTAG 仿真器的硬件上。默认情况下，芯片被识别为 2 个 (USB 转接出来的) 串口，OpenOCD 对 FTDI 系列的支持：通过特定驱动使用 MPSSE 做为 JTAG 硬件接口。在 Windows 下使用需要调整驱动 (Zadig, WinUSB)，在 Linux 下使用需要注意权限。

因为 FT2xxx 内部有两个转串口部件 (interface)，因此在 OpenOCD 的配置中需要选择 JTAG 口位于哪个 interface。

- Sipeed USB 调试器：基于 FT2232d，速度低，JTAG 口位于 interface 0，无 EEPROM ，需要使用 Zadig 进行驱动替换。更换 USB 口可能会丢失驱动配置。
- Bouffalolab debugger 适配板：基于 FT2232HL，速度快，JTAG 口位于 interface 1，且有 EEPROM。在使用 FT_Prog 编程后，直接使用 UsbDriverTool 切换驱动即可。更换 USB 口不会丢失配置。
- 除了 MPSSE 转 JTAG 的引脚是固定的之外，其它引脚可以被自定义成不同的功能，例如但不限于：SRST，TRST，LED
- 可以支持两线的调试接口，比如 SWD 及 cJTAG(目前 602 并不支持)

4 GDB

GDB — GNU Debugger，是 GNU 软件系统中的标准调试器。它的功能如下：

- 可移植到不同平台运行，功能可配置
- GDB 支持各种体系架构的调试：ARM,RISC-V,MIPS,x86, …
- GDB 支持各种编程语言的调试
- GDB 默认是命令行工具，但也有图形化的前端，例如 eclipse 里的 debug GUI
- 目前的 RISC-V GDB 在 Linux 下使用更方便：tab 补全，路径补全，命令历史，…

5 调试实战

1. 从命令行启动 OpenOCD 调试代理，其默认在 3333 端口等待 GDB 连接
    - openocd.exe -f if_bflb_dbg.cfg -f tgt_602.cfg
    - 启动 openocd 时需要提供配置脚本
    - 为了分离调试器硬件及目标芯片，这里将其分开成两个脚本
    - 其中 if 开头的是调试器硬件配置，而 tgt 开头的是目标定义，例如 tgt_702.cfg 就是 702 的目标配置
    - 如果希望连接时，只是 attach 而不对 target 进行任何初始化/改动，需要使用_attach 脚本，比如 tgt_602_attach.cfg。这通常在脱机运行出现异常后需要进行连接调试时使用
2. 如果是 XIP 的代码，需要先烧写再进行 debug
    - 目前 OpenOCD + GDB 不能实现 602 的 SPI/XIP flash 的烧写
    - 从命令行启动 GDB
    - riscv64-unknown-elf-gdb freertos.elf –x 602.init
    - freertos.elf 是需要调试的程序
    - 其中 –x 指定了一些与 target 相关的初始化命令 (target 类型，连接方式，mem map)
    - mem map 会对断点 (BP) 的行为有影响，ro 区域会使用硬件断点 (Hardware Breakpoint)
    - GDB 会根据其中的配置连接到 OpenOCD
    - 可以支持远程调试，即 OpenOCD 与 GDB 不在同一台 PC 上
3. 在 GDB 下加载调试的 elf 文件 (代码) 内容
    - 对于运行于 ram 的代码，使用如下命令
        - load

  - 执行这个命令后，PC 会被设置到 elf 的入口地址。
- 对于运行于 XIP 的代码，无需也不能使用 load，因为 XIP 的程序需要在调试前使用烧写工具进行烧写，且对 XIP 区域进行写入可能会造成未知错误。另外，XIP 的程序需要依赖 ROM code(0x21000000) 进行相关的软硬件初始化，所以需要进行如下配置：
  - set $pc = 0x21000000
  - set $mie = 0
  - set $mstatus = 0x1880

4. 运行控制
   - 单步执行 (step 与 next): s, si, n, ni
     - 带有 i 的是指令级别，step 与 next 的区别在于 next 不会进入函数调用
     - si 4 <- 单步执行 4 个指令
   - 显示汇编指令上下文：set disassemble-next-line on
   - 从当前函数返回：finish / return
   - 继续程序执行 (continue): c
5. 断点
   - 在函数 main 处打一个断点 (breakpoint)：b main
   - 在函数 main 处打一个临时 (temporary) 硬件 (hardware) 断点：thb main
   - 硬件断点是有限资源，602 上有 4 个
   - 条件断点：当 argc 为 3 时停在 main：b main if argc == 3
   - 指定 (源码) 类型数据断点 (观察点): watch *(uint32_t *)addr
   - 列出所有断点：i[nfo] b
   - 使能/禁止 1 号断点：enable/disable 1
   - 删除 1 号断点：d 1
6. 查看信息 1
   - 读寄存器，并以 16 进制显示：p/x $a0
   - 查看寄存器: info register <- 简写成 i r
     - i r $mstatus

图 5.1: 查看寄存器

- 修改寄存器值：set $a0 = 0x5a5a5a5a
- 修改变量 x 的值为 5：set var x = 5
- 查看内存，以 16 进制输出 32 个 word：x/32xw 0x22008000
- 查看变量，输出 16 进制：p/x pxCurrentTCB
- 格式化内存为结构体：p/x *(TCB_t *)pxCurrentTCB
- 以字符串格式显示：p/s pxCurrentTCB->pcTaskName
- 输出结构体成员地址：p/x &(pxCurrentTCB->pcTaskName)
- “美化”结构体输出：set print pretty on

7. 查看信息 2
   - 获取当前调试文件信息：info files
   - 获取当前调试文件的全部函数：info functions
   - 获取寄存器值指向的代码位置：info line *($ra)
   - 获取某个地址相关的源码信息：info line *0x2200f7f4
   - 查看当前源码 (list)：l
   - 查看函数 main 的源码：l main
   - 查看文件 main.c 行号 123 开始的源码：l main.c:123
   - 显示当前的调用栈 (backtrace)：bt
   - 显示当前所处位置：where
8. 反汇编
   - 从当前 PC – 10 位置开始反汇编 10 条指令：x/10i $pc – 10
   - 反汇编函数 main：disas main
     - 以源码混合模式反汇编函数 main：disas /m main
     - 显示汇编指令的同时也显示机器码：disas /r main
   - 反汇编地址 0x22008000：disas 0x22008000
9. dump/restore 内存

- 保存从 0x21000000 开始的 128KB 内存到文件 rom.bin：
  - dump binary memory ./rom.bin 0x21000000 0x21020000
- 将文件 ram.bin 恢复到内存 0x22008000 处：
  - restore ram.bin binary 0x22008000

10. 文件路径映射

- 如果 elf 内的源码路径与当前调试环境里的源码路径不一致。比如：在 WSL 下编译而在 windows 上调试，使用如下命令可以 map 文件的搜索路径：
  - set substitute-path /mnt/c c:

11. 增加一个符号表

- 如果在调试 XIP 代码时，发现系统在 ROM 区域出现了问题。因为当前加载的是 XIP 用户程序的 elf 符号，所以 GDB 无法提示 ROM 代码的详细信息，此时，可以使用如下命令添加 bootloader 的符号表以便于调试：
  - add-symbol-file bootloader.elf
- 如果 elf 文件的 link 地址与运行地址有 offset，此命令也可以做相应的处理。

12. 脚本

- 使用用户脚本可以增加一些操作，比如目前有脚本 freertos_fault.gdb。其功能是在 freertos 系统 fault 之后，尝试恢复当前任务的上下文，以供分析。
- 脚本是文本文件，可以随时更新
- 使用方法：
  - source freertos_fault.gdb
  - freertos_fault

13. Freertos awareness

- GDB 支持 thread 调试
- OpenOCD 支持一些 OS 的 awareness，包括 FreeRTOS。
  - 但是目前不支持 RISC-V 上的 FreeRTOS awareness。
- 目前的代码是 hack 性质，还有不少问题
- 代码，编译，使用流程请看：http://10.28.10.249:3000/dytang/bl602_openocd/src/as_patch_set
- 使用效果：
  - info threads

```
(gdb) info threads
[New Thread 1107426736]
[New Thread 1107425840]
[New Thread 1107425344]
[New Thread 1107422864]
[New Thread 1107422256]
[New Thread 1107421488]
[New Thread 1107417040]
[New Thread 1107416304]
[New Thread 1107415696]
[New Thread 1107417744]
[New Thread 1107412336]
[New Thread 1107420880]
[New Thread 1107427504]
[New Thread 1107424240]
[New Thread 1107423632]
[New Thread 1107424848]
[New Thread 1107412944]
[New Thread 1107411120]
[New Thread 1107414288]
[New Thread 1107411728]
[New Thread 1107420272]
[New Thread 1107419568]
[New Thread 1107418960]
[New Thread 1107418352]
[New Thread 1107413552]
[New Thread 1107410512]
[New Thread 1107414896]
  Id   Target Id                                  Frame
  2    Thread 1107426736 (Name: IDLE)             0x2200f902 in prvCheckTasksWaitingTermination () at 3rdParty/FreeRTOS/Source/tasks.c:3647
  3    Thread 1107425840 (Name: Reg2)             Reg2_loop ()
    at ChipTest/FreeRTOS/full_demo/RegTest.S:220
  4    Thread 1107425344 (Name: Reg1)             reg1_loop ()
    at ChipTest/FreeRTOS/full_demo/RegTest.S:143
  5    Thread 1107422864 (Name: PolSEM2)          vTaskDelay (
    xTicksToDelay=0) at 3rdParty/FreeRTOS/Source/tasks.c:1374
  6    Thread 1107422256 (Name: PolSEM1)          vTaskDelay (
    xTicksToDelay=0) at 3rdParty/FreeRTOS/Source/tasks.c:1374
  7    Thread 1107421488 (Name: Rec3)             vTaskExitCritical ()
    at 3rdParty/FreeRTOS/Source/tasks.c:4325
  8    Thread 1107417040 (Name: GenQ, State: Running) vTaskExitCritical ()
    at 3rdParty/FreeRTOS/Source/tasks.c:4307
  9    Thread 1107416304 (Name: CNT2)             vTaskExitCritical ()
    at 3rdParty/FreeRTOS/Source/tasks.c:4325
```

图 5.2: 使用效果

- 查看所有线程的调用栈:
  - thread apply all bt

```
(gdb) thread apply all bt
[New Thread 1107409904]

Thread 29 (Thread 1107409904):
#0  vTaskExitCritical () at 3rdParty/FreeRTOS/Source/tasks.c:4307
#1  0x2201176e in uxQueueMessagesWaiting (xQueue=0x4201d560 <ucHeap+7136>) at 3rdParty/FreeRTOS/Source/queue.c:1936
#2  0x22014036 in prvSendFrontAndBackTest (pvParameters=0x4201d560 <ucHeap+7136>) at 3rdParty/FreeRTOS/Demo/Common/Minimal/GenQTest.c:301
#3  0x00000000 in ?? ()
Backtrace stopped: frame did not save the PC

Thread 28 (Thread 1107414896):
#0  vTaskSuspend (xTaskToSuspend=0x0) at 3rdParty/FreeRTOS/Source/tasks.c:1793
#1  0x220137ec in vSecondaryBlockTimeTestTask (pvParameters=0x0) at 3rdParty/FreeRTOS/Demo/Common/Minimal/blocktim.c:395
#2  0x00000000 in ?? ()
Backtrace stopped: frame did not save the PC

Thread 27 (Thread 1107410512):
#0  vTaskSuspend (xTaskToSuspend=0x0) at 3rdParty/FreeRTOS/Source/tasks.c:1793
#1  0x22012b22 in vLimitedIncrementTask (pvParameters=0x4201b4b0 <ulCounter>) at 3rdParty/FreeRTOS/Demo/Common/Full/dynamic.c:206
#2  0x00000000 in ?? ()
Backtrace stopped: frame did not save the PC

Thread 26 (Thread 1107413552):
#0  vTaskSuspend (xTaskToSuspend=0x0) at 3rdParty/FreeRTOS/Source/tasks.c:1793
#1  0x22012f3a in prvChangePriorityHelperTask (pvParameters=0x0) at 3rdParty/FreeRTOS/Demo/Common/Full/dynamic.c:535
#2  0x00000000 in ?? ()
Backtrace stopped: frame did not save the PC

Thread 25 (Thread 1107418352):
```

图 5.3: 调用栈效果

- 切换线程 2(可能会出问题)：thread 2

```
(gdb) thread 2
[Switching to thread 2 (Thread 1107426736)]
#0  0x2200f902 in prvCheckTasksWaitingTermination ()
    at 3rdParty/FreeRTOS/Source/tasks.c:3647
3647    }
(gdb) bt
#0  0x2200f902 in prvCheckTasksWaitingTermination ()
    at 3rdParty/FreeRTOS/Source/tasks.c:3647
#1  0x2200f7f4 in prvIdleTask (pvParameters=0x0)
    at 3rdParty/FreeRTOS/Source/tasks.c:3394
#2  0x00000000 in ?? ()
Backtrace stopped: frame did not save the PC
(gdb) thread 3
[Switching to thread 3 (Thread 1107425840)]
#0  Reg2_loop () at ChipTest/FreeRTOS/full_demo/RegTest.S:220
220             li x5, 0x24
(gdb) bt
#0  Reg2_loop () at ChipTest/FreeRTOS/full_demo/RegTest.S:220
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb)
```

图 5.4: 切换线程 2 效果

- 打印线程 2 的 TCB：
  - p/x *(TCB_t *)1107426736

```
(gdb) p *(TCB_t *)1107426736
$3 = {
  pxTopOfStack = 0x4201fcd8 <ucHeap+17240>,
  xStateListItem = {
    xItemValue = 3232158965,
    pxNext = 0x4201bbf4 <ucHeap+628>,
    pxPrevious = 0x4201fa34 <ucHeap+16564>,
    pvOwner = 0x4201fdb0 <ucHeap+17456>,
    pvContainer = 0x4201b27c <pxReadyTasksLists>
  },
  xEventListItem = {
    xItemValue = 7,
    pxNext = 0x762ee8ba,
    pxPrevious = 0x2d0d2f43,
    pvOwner = 0x4201fdb0 <ucHeap+17456>,
    pvContainer = 0x0
  },
  uxPriority = 0,
  pxStack = 0x4201fbc0 <ucHeap+16960>,
  pcTaskName = "IDLE\000\064\247v\210\207\031{\343s\001",
  uxCriticalNesting = 0,
  uxTCBNumber = 27,
  uxTaskNumber = 3840114188,
  uxBasePriority = 0,
  uxMutexesHeld = 0,
  ulNotifiedValue = 0,
  ucNotifyState = 0 '\000',
  ucDelayAborted = 0 '\000'
}
(gdb)
```

图 5.5: 打印线程 2 效果

14. 其它

- 在 GDB 下可以通过 mon(monitor) 命令调用调试代理内部的命令
- 例如：调用 OpenOCD 的内建命令来调整 JTAG 频率：
  - mon adapter_khz
  - mon adapter_khz 8000

15. Risc-v SBA

- 如果要在 RISC-V 运行时查看内存数据，那么需要：
  - RISC-V core 需要支持 System Bus Access <- 602 支持
  - OpenOCD 中打开 SBA：riscv set_prefer_sba on <- tgt_* 中默认配置
  - 当 GDB 处于 continue 状态而不能接受更多用户命令时
  - 启动一个 telnet 程序，连接到 OpenOCD telnet 服务器端口 (默认 4444)
- 使用 mdw 命令可以进行 mem dump word：

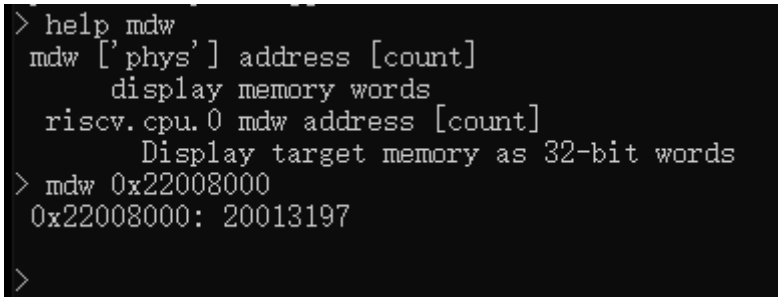

图 5.6: 使用 mdw 效果

- 其它的 telnet 命令可以使用 help 获取