类别	嵌入式开发
文档名	RISC-V 体系结构编程与实践_基于百问网 Allwinner Dls 的学 习指南
当前版本	1.0
日期	2023. 02. 13
适用型号	DongshanPI-D1S
编辑	百问科技文档编辑团队
审核	韦东山

更新记录

第1章 环境搭建

1.1 开发板介绍

张天飞老师编写的《RISC-V体系结构编程与实践》,里面的源码是基于 QEMU 模拟器的,可以认为它是一款虚拟的开发板。如果需要在真实开发板上学习,可以使用百问网的 DongshanPI-D1S 开发板。

DongshanPI-D1S 是百问网推出的一款基于 RISC-V 架构的学习裸机、RTOS 的最小开发板。集成 CKLink 调试器,支持 OTG 烧录、GDB 调试、串口打印,并将主芯片所有的信号全部引出,其中左右两侧兼容了树莓派的电源信号定义,可以很方便扩展模块。

D1S 是全志公司针对智能解码市场推出的高性价比 AIoT 芯片,它使用阿里平头哥的 64bit RISC-V 架构的 C906 处理器,内置了 64M DDR2,支持 FreeRTOS、RT-Thread 等 RTOS,也支持 Linux 系统。同时集成了大量自研的音视频编解码相关 IP,可以支持 H. 265、H. 264、MPEG-1/2/4、JPEG 等全格式视频解码,支持 ADC、DAC、12S、PCM、DMIC、OWA 等多种音频接口,可以广泛应用于智能家居面板、智能商显、工业控制、车载等产品。



板子资料: <u>http://download.100ask.net/boards/Allwinner/Dls/index.html</u> 购买地址: <u>https://item.taobao.com/item.htm?id=688098912622</u>

- 2 -

1.2 下载资料

资料分两部分:开发板通用资料、《RISC-V体系结构编程与实践》的 D1S 源码。前者比较庞大,放在百度网盘;后者放在书籍配套的 GITEE 网站。

开发板通用资料:

打开 <u>http://download.100ask.net/boards/Allwinner/Dls/index.html</u>,可以看到"Dls 课程配套通用资料"对应的百度网盘地址,请自行下载。本课程主要使用下图所示的软件:

>

开发板通用资料 > 05_开发配套工具
名称
CKLinkServer
📙 Git
📙 make
📙 toolchain
xfel

《RISC-V 体系结构编程与实践》的 D1S 源码:

打开 <u>https://gitee.com/weidongshan/riscv_programming_practice</u>, 登录后按如下界 面操作:

\leftrightarrow \rightarrow C $($ a gitee.com/benshushu/riscv_programmin	g_practice/tree/master/		
💪 gitee 开源软件 企业版	¹¹¹ 高校版 私有云 博客		
。 该仓库未声明开源许可证文件(LICENSE),使	用请关注具体项目描述及其代码上游依赖。 💡		×
master ▼ _ \$> 分支 2		文件 ▼ Web IDE	克隆/下载 ▼
Q 捜索分支	4 20小时前	4.点击下载	国 3 次提交
分支(2) 2 可以看到两个分支 管理	eference code for lab		11天前
	eference code for lab		11天前
master 2.1 这是基于QEMU的源码	eference code for lab		11天前
DongShanPI_D1 2.2 这是基于D1S的	源码 forence code for lab 3.点击DongS	hanPI_D1	11天前

点击"克隆/下载"按钮之后,如下点击"下载 ZIP"即可:



如果你没有点击"下载 ZIP",而是使用 GIT 命令来下载,那么下载成功后还需要执行如下命令:

git checkout DongShanPI_D1

1.3 安装软件

需要安装如下 5 个软件, 它们都位于网盘资料"开发板通用资料\05_开发配套工具\"目录下:

① "Git\Git-2.39.1-64-bit.exe": 我们把它当做命令行,不能使用 Windows 自带的 DOS 命 令行、Powershell(在里面无法执行 make 命令)

② "make\make-3.81.exe": make 工具

③ "toolchain\Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui-20230210.tar.gz": 这是 Windows 版本的交叉编译工具,并且支持 TUI

④ "CKLinkServer\T-Head-DebugServer-windows-V5.16.6-20221102-1510.zip": 这是调试 服务软件

⑤ "xfe\xfe.exe": 烧写工具

1.3.1 Git Bash

双击"开发板通用资料\05_开发配套工具\Git\Git-2.39.1-64-bit.exe"即可安装。 启动 Git Bash 有两种方法:

① 点击"开始->Git->Git Bash"

 ② 在文件浏览器进入某个目录后,在空白处点击右键弹出菜单后选择 "Git Bash Here" 在 Git Bash 中各种命令的用法跟 Linux 完全一样,比如也有 "cd"、"1s"、"rm"等命 令。在 Git Bash 中,对路径的表示方法也跟 Linux 一样,比如 D 盘下的 abc 子目录使用 "/d/abc"表示,而不是 "D:\abc"。

在 Git Bash 中使用命令简单示范如下:

```
MINGW64:/d/abc
                                                                            X
veidongshan@DESKTOP-TP8DH2I MINGW64 /e/d1s_projects/riscv_programming_practice-
or-dongshan/chapter_2/benos
§ pwd
/e/d1s_projects/riscv_programming_practice-for-dongshan/chapter_2/benos
weidongshan@DESKTOP-TP8DH2I MINGW64 /e/d1s_projects/riscv_programming_practice-f
  -dongshan/chapter_2/benos
$ cd /d 2.进入D盘根目录
veidongshan@DESKTOP-TP8DH2I MINGW64 /d
$ cd abc <u>3</u>.进入abc子目录
weidongshan@DESKTOP-TP8DH2I MINGw64 /d/abc
                             4.列出当前目录的内
01_all_series_quickstart/
09_UART/
```

1.3.2 make

双击"开发板通用资料\05_开发配套工具\make\make-3.81.exe"即可安装。安装时,要记住安装的路径,需要把安装路径下的 bin 目录放入环境变量 Path 里。

1			-
elect Destination Location			0
Where should Make be installed?			1. S.
Setup will install Make into the fo	lowing folder.		
To continue, click Next. If you would like to	o select a different folder, click	Browse	
		-	
C:\Program Files (x86)\GnuWin32		Brow	/se
C:\Program Files (x86)\GnuWin32 要记住这个路径,它下面的bin	子目录要放入Path环境 ³	Brow 使量里	/se
C:\Program Files (x86)\GnuWin32 要记住这个路径,它下面的bin于	子目录要放入Path环境	Brow 使量里	/se
[C:\Program Files (x86)\GnuWin32 要记住这个路径,它下面的bin于	子目录要放入Path环境	Brow 使量里	/se
IC-YProgram Files (x86)\GnuWin32 要记住这个路径,它下面的bin	子目录要放入Path环境。	Brow 使量里	/se
C:\Program Files (x86)\GnuWin32 要记住这个路径,它下面的bin	子目录要放入Path环境?	Brow 使量里	/se
C:\Program Files (x86)\GnuWin32 要记住这个路径,它下面的bin子 At least 0.7 MB of free disk space is requir	子目录要放入Path环境。 ed.	Brow 变量里	/se

如下图把 "C:\Program Files (x86)\GnuWin32\bin" 添加进环境变量 Path:



验证: 启动 Git Bash 后执行"make -v"命令,如下图所示。



- 7 -

1.3.3 交叉工具链

把"开发板通用资料\05_开发配套工具\toolchain\Xuantie-900-gcc-elf-newlibmingw-V2.6.1-gdbtui-20230210.tar.gz"解压即可,注意路径名不要有中文。 解压后要确认如下目录里的文件不是0字节:

Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui-20230210 > Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui > riscv64-unknown-elf > bin

名称 ^	修改日期	类型	大小
📧 ar.exe	2023/02/10 19:59	应用程序	1,710 KB
s.exe	2023/02/10 19:59	应用程序	2,108 KB
📧 ld.bfd.exe	2023/02/10 19:59	应用程序	2,082 KB
📧 ld.exe	2023/02/10 19:59	应用程序	2,082 KB
nm.exe	2023/02/10 19:59	应用程序	1,700 KB
📧 objcopy.exe	2023/02/10 19:59	应用程序	1,811 KB
📧 objdump.exe	2023/02/10 19:59	应用程序	2,266 KB
📧 ranlib.exe	2023/02/10 19:59	应用程序	1,710 KB
📧 readelf.exe	2023/02/10 19:59	应用程序	1,628 KB
📧 strip.exe	2023/02/10 19:59	应用程序	1,811 KB
	确认	不是0字节	

使用有些解压工具比如 banzip 可能会得到 0 字节的文件,建议使用 7-Zip 解压。 解压成功后,可以看到"riscv64-unknown-elf-gcc.exe"文件,如下图所示:

tools (D:)	>	Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui-20230210 > Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui > bin
А	^	8秒 2. 此路径放入Path环境变量
*		📧 riscv64-unknown-elf-addr2line.exe
		📧 riscv64-unknown-elf-ar.exe
		📧 riscv64-unknown-elf-as.exe
		📧 riscv64-unknown-elf-c++.exe
		📧 riscv64-unknown-elf-c++filt.exe
		📧 riscv64-unknown-elf-cpp.exe
		📧 riscv64-unknown-elf-elfedit.exe
		riscv64-unknown-elf-g++.exe
		I. 确认有此文件
		📧 riscv64-unknown-elf-gcc-10.2.0.exe
		📧 riscv64-unknown-elf-gcc-ar.exe
		📧 riscv64-unknown-elf-gcc-nm.exe
		📧 riscv64-unknown-elf-gcc-ranlib.exe

需要把"riscv64-unknown-elf-gcc.exe"文件所在目录放入 Path 环境变量里,具体方法可以参考《1.3.2 make》。结果如下图所示:

eldongshan 15/15/ 52±(0)			
变量	值	C:\Windows\System32\OpenSSH\	新建(N)
MOZ PLUGIN PATH	D:\PROGRAM FIL	C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common	
OneDrive	C:\Users\weidon	C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR	编辑(E)
OneDriveConsumer	C:\Users\weidon	E:\cortexA7_windows_tools\gcc-arm-linux-gnueabihf-6.2.1\bin	
Dath	C:\Users\weidon	C:\Program Files\STMicroelectronics\STM32Cube\STM32Cub	浏览(B).
TEMD	C:\Users\weidon	C:\Program Files\CodeBlocks\MinGW\bin	
TMD	C:\Users\weidon	C:\Python27\	删除(D)
TIVIP	C:\Users\weidon	C:\Program Files (x86)\openocd\0.6.0\bin	
		C:\Program Files\CMake\bin	
		%SystemRoot%\system32	上移(U)
		%SystemRoot%	
		%SystemRoot%\System32\Wbem	下移(O)
统变量(S)	_	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\	下移(O)
统变量(S)	店	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\	下移(0)
统变量(S) 变量	值	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd	下移(O) 编辑文本(
统变量(S) 变量 ComSpec	值 C:\WINDOWS\sy	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd C:\Users\weidongshan\AppData\Roaming\Python\Python38\	下移(0) 编辑文本(
統变量(S) 变量 ComSpec DriverData	值 C:\WINDOWS\sy C:\Windows\Syst	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd C:\Users\weidongshan\AppData\Roaming\Python\Python38\ D:\xfel-windows-v1.2.9	下移(O) 编辑文本(1
統变量(S) 变量 ComSpec DriverData MOSQUITTO_DIR	值 C:\WINDOWS\sy C:\Windows\Syst C:\Program Files	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd C:\Users\weidongshan\AppData\Roaming\Python\Python38\ D:\xfel-windows-v1.2.9 C:\Program Files\dotnet\	下移(O) 编辑文本(
統变量(S) 变量 ComSpec DriverData MOSQUITTO_DIR NUMBER_OF_PROCESSORS	值 C:\WINDOWS\sy C:\Windows\Syst C:\Program Files 16	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd C:\Users\weidongshan\AppData\Roaming\Python\Python38\ D:\xfel-windows-v1.2.9 C:\Program Files\dotnet\ D:\Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui-2023021	下移(0) 编辑文本(1
統变量(S) 变量 ComSpec DriverData MOSQUITTO_DIR NUMBER_OF_PROCESSORS OS	值 C:\WINDOWS\sy C:\Windows\Syst C:\Program Files 16 Windows_NT	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd C:\Users\weidongshan\AppData\Roaming\Python\Python38\ D:\xfel-windows-v1.2.9 C:\Program Files\dotnet\ D:\Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui-2023021 C:\Program Files (x86)\GnuWin32\bin	下移(O) 编辑文本(
統变量(S) 变量 ComSpec DriverData MOSQUITTO_DIR NUMBER_OF_PROCESSORS OS Path	值 C:\WINDOWS\sy C:\Windows\Syst C:\Program Files 16 Windows_NT C:\Program Files	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd C:\Users\weidongshan\AppData\Roaming\Python\Python38\ D:\xfel-windows-v1.2.9 C:\Program Files\dotnet\ D:\Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui-2023021 C:\Program Files (x86)\GnuWin32\bin	下移(O) 编辑文本(
統变量(S) 变量 ComSpec DriverData MOSQUITTO_DIR NUMBER_OF_PROCESSORS OS Path PATHEXT	值 C:\WINDOWS\sy C:\Windows\Syst C:\Program Files 16 Windows_NT C:\Program Files .COM;.EXE;.BAT;.	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd C:\Users\weidongshan\AppData\Roaming\Python\Python38\ D:\xfel-windows-v1.2.9 C:\Program Files\dotnet\ D:\Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui-2023021 C:\Program Files (x86)\GnuWin32\bin	下移(0) 编辑文本(
統变量(S) 变量 ComSpec DriverData MOSQUITTO_DIR NUMBER_OF_PROCESSORS OS Path PATHEXT	值 C:\WINDOWS\sy C:\Windows\Syst C:\Program Files 16 Windows_NT C:\Program Files .COM;.EXE;.BAT;.	%SystemRoot%\System32\Wbem %SYSTEMROOT%\System32\WindowsPowerShell\v1.0\ %SYSTEMROOT%\System32\OpenSSH\ C:\Program Files\Git\cmd C:\Users\weidongshan\AppData\Roaming\Python\Python38\ D:\xfel-windows-v1.2.9 C:\Program Files\dotnet\ D:\Xuantie-900-gcc-elf-newlib-mingw-V2.6.1-gdbtui-2023021 C:\Program Files (x86)\GnuWin32\bin	下移(O) 编辑文本(1

验证: 启动 Git Bash 后执行 "riscv64-unknown-elf-gcc -v" 命令,如下图所示 (Git Bash 支持命令补全功能,输入 "risc" 后按 TAB 键会自动补全命令)。



1.3.4 调试服务软件

先解压文件:"开发板通用资料\05_开发配套工具\CKLinkServer\T-Head-DebugServer-windows-V5.16.6-20221102-1510.zip"。

再双击里面的"setup.exe"即可安装。

1.3.5 烧写工具

把"开发板通用资料\05_开发配套工具\xfel"目录复制到其他非中文路径即可。

还需要把"xfel.exe"文件所在目录放入 Path 环境变量里,具体方法可以参考《1.3.2 make》。结果如下图所示:



验证: 启动 Git Bash 后执行"xfel —help"命令,如下图所示。

weidongshan@DESKTOP-TP8DH2I MINGw64 ~	
\$ xfelhelp	
<pre>xfel(v1.2.9) - https://github.com/xboot/xfel</pre>	
usage:	
xfel version	- show chip version
xfel hexdump <address> <length></length></address>	- Dumps memory region in
hex	
xfel dump <address> <length></length></address>	- Binary memory dump to
stdout	
xfel read32 <address></address>	- Read 32-bits value fro
m device memory	
xfel write32 <address> <value></value></address>	- Write 32-bits value to
device memory	
xfel read <address> <length> <file></file></length></address>	- Read memory to file
xfel write <address> <file></file></address>	- Write file to memory
xfel exec <address></address>	- Call function address
xtel reset	- Reset device using wat
chdog	
xtel sid	- Show sid information
xfel jtag	- Enable jtag debug
xtel ddr [type]	- Initial ddr controller
with optional type	
xtel sign <public-key> <private-key> <tile></tile></private-key></public-key>	- Generate ecdsa256 sign

1.4 安装驱动

DongshanPI-D1S 开发板各接口如下图所示:



D1S 自身支持 USB-OTG 烧录 (对应上面的接口 "3. OTG 烧录接口"),这需要安装对应的驱动程序。

DongshanPI-D1S 开发板集成了 CKLink 调试器 (对应上面的接口 "4. 调试&串口接口"), 它有 2 个功能: 调试、USB 串口, 需要安装 2 个驱动程序。

1.4.1 OTG 烧录驱动程序

使用 USB 线连接开发板的"3.0TG 烧录接口"到电脑后,先按住"2.烧录模式按键"不 松开,然后按下、松开"5.系统复位按键",最后松开"2.烧录模式按键",开发板就会进入 烧录模式。

第一次使用烧录模式时,要先安装驱动程序,先运行程序"开发板通用资料\05_开发配 套工具\xfel\Drivers\zadig=2.7.exe",然后如下图操作:

🖾 Zadig		– 🗆 🗙		
Device O	ptions <u>H</u> elp 1. 勾选			
	List All Devices			
Unkno 🗸	Ignore Hubs or Composite Parents	✓ ☐ Edit		
Driver	Create a Catalog File Sign Catalog & Install Autogenerated Certificate	More Information WinUSB (libusb)		
USB ID	Advanced Mode	libusb-win32		
WCID	Log Verbosity >	libusbK WinUSB (Microsoft)		
13 devices	found.	Zadig 2.7.765		
Zadig Device Options Help 2.让开发板进入烧录模式后 × 在下拉框找到ID为1F3A的设备				
Unknown	Device #1	✓ ☐ Edit		
Driver V USB ID 1 WCID ²	VinUSB (v6. 1. 7600. 16785) WinUSB (v6. 1. 7600. 16385)	选择WinUSB More Information WinUSB (libusb) libusb-win32 libusbK WinUSB (Microsoft)		
13 devices	ound. 4.无论按钮是什么文字,	都点击 2.7.765		

注意: 上图的第 4 步里, 按钮内容可能是"Install Driver"、"Replace Driver"或 "Reinstall Driver", 都一样点击。

- 13 -

验证:安装好驱动程序后,使用按钮让板子进入烧录模式,然后在 Git Bash 中执行命 令,可以检测到设备:



如果没找到设备,可以多次尝试:

- ① 使用按钮让开发板进入烧录模式
- ② 重新安装驱动、甚至重启电脑
- ③ 插到电脑的其他 USB 口

1.4.2 USB 串口和调试器

使用 USB 线连接开发板的"4. 调试&串口接口"到电脑后,它会自动安装 2 个驱动程序, 打开设备管理器可以看到如下设备:

- 過 设备管理器	_	\times
文件(F) 操作(A) 查看(V) 帮助(H)		
✓ 畳 DESKTOP-T 按连接列出设备(V)		 ^
↓ 💭 🔤 10 inch 🕴 按容器列出设备(C) 1. 点 击		
🔤 HID ៖ 按类型列出资源(Y)		
IIID-→ 按连接列出资源(N)		
♥ USB 显示隐藏的设备(W)		
A 符合 HID 标准的供应商定义设备		
🙀 符合 HID 标准的手写笔		
V 🛄 BenQ G910WAL		
💷 通用即插即用监视器		
✓ 🛄 C24F390		
C24F390 (NVIDIA High Definition Audio)		
通用即插即用监视器		
▼ I C-Sky CKLink-Lite 2. 可以看到		
CKLink-Lite		
♥ USB Composite Device		
₩ USB 串行设备 (COM30)		
→ 夏 DESKTOP-TP8DH2I USB申口		
> EPSONA9EFB6 (L3150 Series)		
> 🚔 Fax		
> 🚍 Foxit Phantom Printer		~
按容器显示设备。		

第2章体验第一个程序

2.1 编译烧录运行

2.1.1 编译



然后在 Git Bash 中执行"make"命令,可以生成 benos_payload.bin 文件,如下图所示:

♦ MINGW64:/d/abc/riscv_programming_practice/chapter_2/benos – □ ×
weidongshan@DESKTOP-TP8DH2I MINGW64 /d/abc/riscv_programming_practice/chapter_2/
benos (DongShanPI_D1) \$ make4.执行make命令
rm -rf build_src build_sbi *.bin *.map *.elf *.dis
CC build_src/kernel_c.o
CC build_src/sys-uart_c.o
AS build_src/boot_s.o
OBJCOPY benos.bin
OBJDUMP benos.dis
CC build_spi/spi_lib_c.o
AS build_sbi/sbi_boot_s.o
AS build_sbi/sbi_payload_s.o
OBJCOPY mysbi.bin
OBJDUMP mysbi.dis
DB1COPY benos_payload.elt
The bootloader head has been fixed, spl size is 24576 bytes.
MKSUNXI benos_payload.bin 5.得到这个文件
OBJOMP Denos_pay load.urs

2.1.2 烧录运行

使用 2 条 USB 线,分别连接开发板的"3.0TG 烧录接口"、"4.调试&串口接口",使用串 口工具打开串口,波特率设为 115200,8 个数据位,1 个停止位,不使用流量控制。 烧录方法如下:

① 先让开发板进入烧录模式:

方法为:先按住"2.烧录模式按键"不松开,然后按下、松开"5.系统复位按键",最后 松开"2.烧录模式按键"。

② 然后在 Git Bash 开执行"make burn"命令

如下图所示:

MINGW64:/d/abc/riscv_programming_practice/chapter_2/benos	-		\times
			-
weidongshan@DESKTOP-TP8DH2I MINGW64 /d/abc/riscv_programming_practice/ch	apt	er_	2/
benos (DongShanPI_D1)			
\$ make burn			
xfel spinor write 0 benos pavload.bin			
100% [======] 24.000 кв. 95.87	3 K	B/s	
100% Г===================================	83	КВ/	s
		/	

	🚀 3. Serial (COM) 🛛 🕹 👋	÷	
		Welcome	RISC-V!
Welcome	RISC-V!		

烧写成功后,按下、松开"5.系统复位按键"即可启动程序,可以在串口看到输出信息:

2.2 调试

2.2.1 GDB 常用命令

使用 GDB 调试时,涉及两个软件: ① 在 Git Bash 中运行的"riscv64-unknown-elf-gdb":它发出各类调试命令,比如连接调 试服务软件(T-HeadDebugServer)、单步运行、查看变量等等

② T-HeadDebugServer: 它就是"调试服务软件",负责接收、处理各类调试命令

命令	简写形式	说明
4 4		连接调试服务器,比如:
target		target remote 127.0.0.1:1025
run	r	运行程序
continue	c, cont	继续运行
	b	设置断点,比如:
hreat		b sbi_main.c:121
break		b sbi_main
		b *0x20000
delete	d	删除断点
disable	dis	禁用断点
info breakpoints	info b	显示断点信息
next	n	执行下一行
nexti	ni	执行下一行(以汇编代码为单位)
step	S	一次执行一行,包括函数内部
setpi	si	执行下一行
list	1	显示函数或行
		显示表达式,比如:
		print a
print	р	print \$pc // 打印寄存器
		print *0x20000//打印内存
		print /x a // 16进制打印
		显示内存内容,比如:
х		x 0x20000
		x /x 0x20000 //16 进制
info registers	infor r	打印所有寄存器的值
		设置变量,比如:
set		set var a=1
		set *(unsigned int *)0x28000 =

常见的命令如下表所示:

		0x55555555
		set var \$pc=0x22000
finish		运行到函数结束
help	h	显示帮助一览
backtrace	bt, where	显示 backtrace
armhol_filo		加载符号表,比如
Symbol-111e		symbol-file benos.elf

2.2.2 benos_payload 程序组成

《RISC-V 体系结构编程与实践》中的代码分为两部分:
① mysbi.elf:运行于 M 模式的底层软件,提供各种系统服务
② benos.elf:运行于 S 模式的应用软件 benos_payload 是这两部分程序的组合:
benos_payload.elf = mysbi.elf + benos.elf benos_payload.bin = mysbi.bin + benos.bin

烧写、运行 benos_payload. bin 时,会先运行 mysbi 程序, mysbi 再启动 benos。调试 benos_payload. elf 时,我们可以先调试 mysbi,等 benos 启动后再调试 benos。

2.2.3 调试准备工作

首先,启动 CKLink 的调试功能,方法为:把下图中蓝色箭头所指的拨码开关上的薄膜撕开,把所有拨码开关拨向左边(USB 接口那边):



然后,启动调试服务软件"T-HeadDebugServer",它会自动检测到芯片,如下图所示:



如果没有上图所示信息,有多种可能:

① 没有自动识别:

可以手动识别,如下图所示:

T-HeadDebugServer 点击圆形按钮, 它会变为三角形□ \times 再点击三角形按钮 File View Control Setting Lools <u>H</u>elp 学 🔒 🗡 -• 🐪 🖸 🔉 S 👂 🖶 🗛 🖂 🎬 🔋

② 板子上的程序有 Bug,导致板子死机了:可以让板子先进入烧录模式,再按照步骤①操作
 ③ 提示 1025 端口被占用:运行任务管理器,把所有"T-HeadDebugServer"杀掉,再重新运行"T-HeadDebugServer"

当"T-HeadDebugServer"识别出芯片后,就可以调试程序了,有2种方式:

① 使用命令行模式调试

② 使用 TUI 模式调试

2.2.4 命令行调试示例

当"T-HeadDebugServer"识别出芯片后,就可以在 Git Bash 里执行"riscv64-unknownelf-gdb benos_payload.elf"来调试程序了。

```
示例如下:
```

weidongshan@DESKTOP-TP8DH2I MINGW64 /d/abc/riscv_programming_practice/chapter_2/benos (DongShanPI D1) \$ riscv64-unknown-elf-gdb benos_payload.elf Reading symbols from benos_payload.elf... (gdb) target remote 127.0.0.1:1025 // 连接调试服务软件 Remote debugging using 127.0.0.1:1025 0x00000000000a22a in ?? () (gdb) load // 加载benos_payload.elf Loading section .text.boot, size 0x44 lma 0x20000 section progress: 100.0%, total progress: 0.38% Loading section .text, size 0x538 lma 0x20044 section progress: 100.0%, total progress: 7.81% Loading section .rodata, size 0xc0 1ma 0x2057c section progress: 100.0%, total progress: 8.88% Loading section .data, size 0x1000 lma 0x21000 section progress: 100.0%, total progress: 31.66% Loading section .payload, size 0x3000 lma 0x22000 section progress: 100.0%, total progress: 100.00% Start address 0x0000000000000000, load size 17980 Transfer rate: 209 KB/sec, 1997 bytes/write. (gdb) x /x 0x20000 // 检查0x20000是否被正确写入, // 我们调试程序时可能因为上次的死机导致无法load // 这时可以让板子进入烧录模式,再重新连接、重新加载 0x20000 <text begin>: 0x0300006f (gdb) b sbi main // 设置断点为sbi_main函数 Breakpoint 1 at 0x204bc: file sbi/sbi_main.c, line 80. (gdb) c // 执行 Continuing. Breakpoint 1, sbi main () at sbi/sbi main.c:80 80 sbi set pmp(0, 0, -1UL, PMP RWX); // 下一步 (gdb) n 84 val = read csr(mstatus);

- 22 -

```
(gdb) b sbi main.c:102 // 设置断点为sbi main.c的102行
Breakpoint 2 at 0x20564: file sbi/sbi_main.c, line 102.
(gdb) info b // 查看所有断点
Num
       Type
                    Disp Enb Address
                                              What
1
                    keep y 0x0000000000204bc in sbi main
       breakpoint
                                              at sbi/sbi main.c:80
       breakpoint already hit 1 time
2
       breakpoint
                    keep y 0x000000000020564 in sbi_main
                                              at sbi/sbi main.c:102
(gdb) i b
          // 查看所有断点, 简写的命令
                    Disp Enb Address
                                              What
Num
       Type
                    keep y 0x0000000000204bc in sbi main
1
       breakpoint
                                              at sbi/sbi_main.c:80
       breakpoint already hit 1 time
2
       breakpoint
                    keep y 0x000000000020564 in sbi main
                                              at sbi/sbi main.c:102
(gdb) c // 继续执行
Continuing.
Breakpoint 2, sbi main () at sbi/sbi main.c:102 // 碰到断点了
                                           // 执行完下一句代码就会跳到benos程序
102
              asm volatile("mret");
(gdb) si // 单步执行并进入函数,每次执行一条汇编语句
0x00000000022000 in payload bin () // 现在要执行另一个程序benos了
(gdb) symbol-file benos.elf // 读取benos.elf的符号表,否则你调试时无法知道函数、代码等信息
Load new symbol table from "benos.elf"? (y or n) [answered Y; input not from terminal]
Reading symbols from benos.elf...
Error in re-setting breakpoint 1: Function "sbi_main" not defined.
Error in re-setting breakpoint 2: No source file named sbi main.c.
(gdb) si // 单步执行并进入函数,每次执行一条汇编语句
9
              la sp, stacks start
(gdb) b kernel main // 设置断点为kernel main函数
Breakpoint 3 at 0x22020: file src/kernel.c, line 6.
(gdb) c // 继续执行
Continuing.
Breakpoint 3, kernel main () at src/kernel.c:6
```

- 23 -

6	sy	s_clock_i	nit();			
(gdb) i	r //	列出所有	寄存器的	值		
ra		0x204d0	0x204d0			
sp		0x24ff0	0x24ff0			
gp		0x0	0x0			
tp		0x0	0x0			
t0		0x1000	4096			
t1		Oxffffff	fffffff0	00	-4096	
t2		0x109	265			
fp		0xa00000	900	0xa0000	0900	
s1		0x0	0			
a0		0x0	0			
al		0x1f	31			
a2		Oxffffff	fffffff	ff	-1	
a3		0x0	0			
a4		0xa00000	100	4294967	3216	
a5		0x0	0			
a6		0x80	128			
a7		0x1c0	448			
s2		0x375bff	17	9287759	59	
s3		0xff32de	с	2675952	44	
s4		0x2eebef	fb	7872143	31	
s5		Oxffffff	ffffdf9f	fd	-2121731	
s6		0x355077	ff	8944660	47	
s7		Oxffffff	ffef7eee	e9	-276893975	
s8		0x27034	159796			
s9		0xffffff	ffe6376f	f3	-432574477	
s10		0xffffff	ffb9d37b	fc	-1177322500	
s11		0x78b47e	70	2025094	768	
t3		0x1	1			
t4		0xefe8	61416			
t5		0x8001	32769			
t6		0x0	0			
рс		0x22020	0x22020	<kernel< td=""><td>_main+8></td><td></td></kernel<>	_main+8>	
(gdb) 1	// 歹	们出代码				
1	#inclu	de "clock	. h″			
2	#inclu	de "uart.	h″			

```
3
4
       void kernel_main(void)
       {
5
6
           sys_clock_init();
7
              uart init();
8
9
               while (1) {
10
              uart_send_string("Welcome RISC-V!\r\n");
(gdb) 1
11
                       ;
12
              }
13 }
(gdb) c // 继续执行
Continuing.
Program received signal SIGINT, Interrupt. // 按Ctrl+C停止程序
read32 (addr=38797436) at include/io.h:23
23
      }
(gdb) quit // 退出调试
```

```
上述调试过程中,用到的命令都有注释,摘抄如下:
```

```
$ riscv64-unknown-elf-gdb benos_payload.elf // 开始调试
(gdb) target remote 127.0.0.1:1025 // 连接调试服务软件
(gdb) load // 加载benos_payload.elf
(gdb) x /x 0x20000 // 检查0x20000是否被正确写入,
              // 我们调试程序时可能因为上次的死机导致无法load
              // 这时可以让板子进入烧录模式,再重新连接、重新加载
(gdb) b sbi_main // 设置断点为sbi_main函数
(gdb) c
             // 执行
          // 下一步
(gdb) n
(gdb) b sbi main.c:102 // 设置断点为sbi main.c的102行
(gdb) info b // 查看所有断点
(gdb) i b // 查看所有断点, 简写的命令
(gdb) c // 继续执行
(gdb) si // 单步执行并进入函数,每次执行一条汇编语句
(gdb) symbol-file benos.elf // 读取benos.elf的符号表,否则你调试时无法知道函数、代码等信息
```

(gdb) si // 单步执行并进入函数,每次执行一条汇编语句
(gdb) b kernel_main // 设置断点为kernel_main函数
(gdb) c // 继续执行
(gdb) i r // 列出所有寄存器的值
(gdb) 1 // 列出代码
(gdb) 1
(gdb) c // 继续执行
Program received signal SIGINT, Interrupt. // 按Ctrl+C停止程序
(gdb) quit // 退出调试

benos_payload.elf 是 2 个程序的组合,调试的要点在于:

① 调试第1个程序时,默认从 benos_payload. elf 里得到符号表

② 执行到第2个程序时,需要使用"symbol-file benos.elf"命令读取 benos.elf 的符号表,否则你调试时无法知道函数、代码等信息。

③ 怎么知道执行到了第2个程序?可以在 sbi_main.c 里如下红框代码处设置断点(比如"b sbi_main.c:102"),执行到断点后,再执行"si"命令就开始运行第2个程序了:

```
* 运行在M模式
 71:
72: */
73: void sbi_main(void)
 74: {
       unsigned long val;
        /*
        * 配置PMP
 79:
         * 所有地址空间都可以访问 */
       sbi_set_pmp(0, 0, -1UL, PMP_RWX);
81:
        /* 设置跳转模式为S模式 */
       val = read csr(mstatus);
84:
       val = INSERT_FIELD(val, MSTATUS_MPP, PRV_S);
       val = INSERT FIELD(val, MSTATUS MPIE, 0);
       write csr(mstatus, val);
       /* 设置M模式的Exception Program Counter, 用于mret跳转 */
       write csr(mepc, FW JUMP ADDR);
       write csr(mtvec, 0x20000);
92:
93:
       /* 设置S模式异常向量表入口地址 */
94:
       write csr(stvec, FW JUMP ADDR);
       /* 关闭S模式的中断 */
       write csr(sie, 0);
       /* 关闭S模式的页表换 */
       write csr(satp, 0);
                                执行完它,
       /* 切换到S模式 */
101:
                                就启动第2个程序
       asm volatile("mret");
103: ] « end sbi main »
```

2.2.5 TUI 调试示例

当"T-HeadDebugServer"识别出芯片后,就可以在 Powershell 里执行"riscv64-unknownelf-gdb -tui benos_payload.elf"来调试程序了。

注意:在Git Bash 中无法使用 TUI 功能, 需要使用 Powershell。

先启动 Powershell: 进入源码目录后, 按住 shift 键同时点击鼠标右键, 在弹出的菜单 里启动 Powershell, 如下图所示:

riscv_programming_practice-for-dongshan > chapter_2 > benos			
名称	^		
huild_sbi			
build_src	查看(V)	>	
include	排序方式(O)	>	
spi	分组依据(P)	>	
tools	刷新(E)		
6 benos.bin	自定义文件夹(F)		
benos.dis	*に目よ/D)		
benos.elf	和四(F)		
benos.map	樹満 電命タ(1)	Ctrl+7	
benos_payload.bin	Git GUI Here	Currz	
benos payload.elf	♦ Git Bash Here 2.点击		
benos_payload.map	在此处打开 Powershell 窗口(S)		
🗋 Makefile	授予访问权限(G)	<u> </u>	
😏 mysbi.bin	120 J. WINNER(O)		
mysbi.dis	新建(W)	>	
mysbi.elt	厪性(R)		
■ mysoumap 1 左順和日昱下	按住 _{abif} 后同时占土息	長士碑	
1.1工你吗日水下,	1y注SHILL 凹凹 从面 臥	你口姓	

在 Powershell 窗口,执行如下命令即可开始调试: riscv64-unknown-elf-gdb -tui benos_payload.elf

执行上述命令后,可以得到如下界面(源码窗口里的汉字是乱码,暂时无法解决):



使用 TUI 的便利在于可以方便地观看源码、反汇编码、寄存器,显示这些信息的窗口被称为 layout。使用以下命令可以显示这些 layout:

- ① layout src: 显示源码窗口
- ② layout asm: 显示汇编窗口
- ③ layout regs: 在之前的窗口上再显示寄存器窗口
- ④ layout split:显示源码、汇编窗口
- ⑤ layout next: 显示下一个 layout
- ⑥ layout prev:显示上一个 layout 能输入各类 GDB 命令的窗口是"命令窗口",它总是显示的。

要同时显示源码和寄存器,可以执行如下2个命令:

layout src

layout regs

要同时显示反汇编码和寄存器,可以执行如下2个命令:

layout asm

layout regs

要同时显示源码和反汇编码,可以执行如下命令:

layout split

使用 TUI 模式时,只是方便我们观看源码、反汇编码、寄存器,具体操作还是在命令窗 口输入 GDB 命令,请参考《2.2.4 命令行调试示例》。

2.2.6 gdb 脚本

如果不想每次执行"riscv64-unknown-elf-gdb benos_payload.elf"或"riscv64unknown-elf-gdb -tui benos_payload.elf"后,都手工执行以下命令来连接调试服务 软件:

(gdb) target remote 127.0.0.1:1025 // 连接调试服务软件

可以把这些命令写入一个名为".gdbinit"的文件里,注意这个文件名的第1个字符是 ".",它表示在Linux系统下它是一个隐藏文件。在Windows的文件浏览器里我们可以看见 它,但是在Git Bash 里执行"1s"命令看不到它,需要执行"1s -a"命令才能看见。

你可以在".gdbinit"里放入更多命令,下面是一个例子:

```
target remote 127.0.0.1:1025
load
```

b sbi_main.c:102 - 29 -