myCPU 启动 Linux 指南

俞子舒 杨丰远 徐易难 周盈坤

2018 龙芯杯 中国科学院大学 1 队

我们假设目前包含 TLB 模块的 CPU 已经通过功能测试，基于 soc_up 搭建的 SoC 能部分启动 PMON。受限于我们的经验与水平，该指南目前仅给出基于 PMON 和 Linux 2.6.32 版本的相关资料。

一、myCPU 需要实现的内容

包含 TLB 的 myCPU 正确启动我们提供的 PMON、Linux 的充分条件是：

- 实现 MIPS32 Release 1 中除了 CP1、JTAG 等指令之外的 102 条指令（附录 1）
- 实现 12 种中断例外支持（附录 2）
- 实现 19 个 CP0 寄存器（附录 3）
- CPU 始终运行在核心态
- CPU 没有 BUG

以上内容的实现参考 MIPS 手册要求，至少有以下几点需要注意：

MIPS32 Release 1 中包含 branch likely 指令，如 BEQL、BNEL 等，这些指令可以通过编译选项 -mno-branch-likely 去除，但是我们仍然建议实现它们。主要原因是，目前 Linux 中的 ramdisk 并不是我们自己编译的，我们也没有源码，然而其中的 init 文件包含这一类的指令。因此，如果不实现这类指令的话，就算 Linux 启动了，init 也会执行错误。这个问题可以通过重新编译 busybox 生成 ramdisk 来实现，但我们还没有试过，感兴趣的可以自己尝试一下。

如果实现了 Cache，需要注意 CP0 Config1 寄存器，同时 Cache 指令中的 Required 部分需要实现。Linux 会根据检测到的 Cache 配置通过 Cache 指令进行 Cache 一致性处理，我们不需要在 Cache 的 RTL 代码层面处理一致性。PREF 指令可以处理为 NOP。

MIPS32 Release 1 中的 CP1 指令（浮点指令）需要报 Coprocessor Unusable 例外，而不是 Reserved Instruction，同时 CP0 Cause 寄存器的 CE 位也需要置上对应值。Linux 会根据 ExcCode 和 CE 位，对例外进行相应处理，即软件模拟浮点指令。

CP0 Processor Identification 寄存器我们使用的是 LOONGSON232 的 0x00004220，PMON 和 Linux 均会通过这个寄存器判断 CPU 类型。如果你希望修改它，做一个真正属于自己的 CPU，需要

修改 PMON 和 Linux 源码。

大赛指令系统规范中，CP0 Status 寄存器第 22 位 BEV 恒为 0，Cause 寄存器第 23 位 IV 恒为 0，例外入口地址统一设置为 0xbfc00380。PMON 的正确运行需要 myCPU 按照 MIPS 手册规定设置入口地址，不能全部使用 0xbfc00380。根据 MIPS 手册，在 BEV 位与 IV 位均为 1 时，中断的处理函数入口地址为 0xbfc00400。令人疑惑的一点是，目前功能测试环境会向以上两位写入 1，因此如果完全按照 MIPS 手册实现例外入口地址的话，功能测试会出错。这里的一个解决方案是，实现 BEV 位，但是不实现 IV 位，经过测试 PMON 和 Linux 都能正常启动。

二、Linux 调试方法

我们非常不希望看到上板调试 Linux 的场景，因此首先请写一个完整的单元测试，充分测试添加的指令、中断例外。比较方便的测试方法是修改功能测试、性能测试 func，修改功能测试时需要手动设计测试点。功能测试完成后，可以在性能测试包中增加测试程序，记得同时修改 Makefile 中的编译选项将-mips1 修改为-mips32。

如果在测试集上发现不了任何错误，而 PMON 或者 Linux 运行却出错了，那接下来的调试过程很可能会非常痛苦，因为你能做的基本只有修改 Linux 源码添加 printk 调试信息、上板抓波形、头脑 debug 等非常 naïve 的事情。

遇到问题时，需要首先确认是否为 Cache 的原因导致出错。如果去掉 Cache 之后仍然出错，优先考虑 CPU 控制逻辑错误，重点关注跳转逻辑、数据前递、中断例外标记等。如果是 Cache 出错，可以根据打印出的错误信息，抓访存波形判断出错位置。但是可能会出现 Cache 出错然后 PMON 直接运行崩了，没有打印出错误信息，此时可以选择抓某些寄存器的特殊值如 CP0 Cause、Status，PMON 运行时基本不产生中断例外。

通过在 Linux 源码加入 printk，可以确认出错位置。注意 Linux 在很长的一段时间内都不会使用 TLB，直到 free unused memory 之后，开始执行用户态进程 init。此时如果出错会提示 init killed，默认不打印出错信息，可以在 arch/mips/kernel/traps.c 各个例外处理函数中去掉原有被注释掉的出错信息。

上板调试时，通过(* mark_debug = true *)来指定 debug 信号，它们会通过 debug core 引出，可以在上板的时候抓出来。在 setup debug 时，可以打开 advanced trigger 选项，会极大地提高抓波形效率，代价就是生成 bit 文件的时间变长。这一块的详细内容可以搜索 ug908 下载 Xilinx 提供的 ug908-vivado-programming-debugging.pdf 文档了解。

三、如何修改、编译 Linux

对 Linux 自身代码的修改基本集中在 arch/mips 文件夹下，这一部分的代码是架构相关的。

1. 编译 Linux

安装 gcc-4.3-ls232 之后，使用它编译 Linux，命令为

```
$ make ARCH=mips nscscc_defconfig
$ make ARCH=mips CROSS_COMPILE=mipsel-linux-
```

第一行命令会将 arch/mips/configs/nscscc_defconfig 处理后写到.config 文件，第二行命令根据.config 文件编译 Linux。如果没有修改 arch/mips/configs/nscscc_defconfig，那么每次运行第二个命令即可。第二个命令可以通过-jN 参数，N 为线程数（根据电脑配置决定），来加速编译。

2. 修改 ramdisk(initramfs)

Linux 运行正确后，如果希望修改初始文件系统内容，增加自己的文件，可以修改 ramdisk 将它们写入 Linux。Linux 通过.config 文件中 CONFIG_INITRAMFS_SOURCE 参数确定 ramdisk 来源，它可以是 txt 文件、目录位置或者打包（以及压缩）的 cpio 文件，目前指定的是 ramdisk.cpio。我们使用 initramfs，参见 Documentation/filesystems/ramfs-rootfs-initramfs.txt 文件。这个文件里面也说明了如何解压、生成 cpio 文件，在 Populating initramfs 章节。

需要注意的一点是，大赛提供的 PMON 命令中，console=ttyS0,115200 rdinit=sbin/init 是传给 Linux 的命令行参数，g 代表运行 load 进来的程序，rdinit 指明 init 进程 elf 文件的位置。如果修改 ramdisk，改变了 init 文件位置，需要同时修改 Linux 命令行参数。

四、Linux 启动过程[1]

在 PMON 上执行 load 命令时，实际上发生的操作是 PMON 读取 elf 文件信息，将它的二进制内容拷贝到对应的内存位置，之后通过 g 命令跳转到 Linux 执行。

Linux 会进入 init/main.c 中的 start_kernel 函数执行，调用一系列初始化函数来设置中断，执行进一步的内存配置，并加载初始 ramdisk。ramdisk 会作为 RAM 中的临时根文件系统使用，并允许内核在没有挂载任何物理磁盘的情况下完整地实现引导。最后，要调用 arch/mips/kernel/process.c 中 kernel_thread 来启动 init 函数。这是第一个用户空间进程，也是第一个使用标准 C 库编译的程序，在桌面 Linux 系统上，通常是 /sbin/init。最后，启动空任务，现在调度器就可以接管控制权了（在调用 cpu_idle 之后）。通过启用中断，抢占式的调度器就可以周期性地接管控制权，从而提供多任

[1] PMON 的启动流程可以参考：吴昌昊，官琴，基于龙芯 1A 平台的 PMON 源码编译和启动分析，PDF 下载链接 http://www.bgzdh.com.cn/UserFiles/20144171648320.pdf

务处理能力。[2]

Linux 初始化过程中具体调用的各个函数对于 CPU 执行来说大同小异，大概只有中断例外、TLB 相关的内容是与架构相关的，在此对其余部分就不多介绍了，感兴趣的可以利用搜索引擎查一下详细的资料。

关于例外处理函数。通过 objdump 可以看到，Linux 的代码段是从 0x80002000 开始的，而很多的例外处理函数入口都在 0x80000000 开始的一小段空间。这一部分的代码是 Linux 在运行时判断 CPU 类型之后，根据不同的 CPU 写过去的，如 TLB 例外相关的处理函数在 arch/mips/mm/tlbex.c 中设置。

关于 TLB 的使用。Linux 调用 inflate_fast 函数解压 ramdisk，这里的访存基本都是字节的读写。之后 Linux 会读取 ramdisk 的内容，读出其中 sbin/init 文件的 elf 信息，将代码段拷贝到对应的物理地址，此时 Linux 第一次使用到 TLB。在 Linux 完成所有初始化工作后，它会创建 init 进程并跳转过去执行，其 pid 为 1。此时是 CPU 第一次执行用户进程，也就意味着 PC 取指第一次需要经过 TLB。init 进程会执行很多启动项，最终得到我们看到的 Linux 命令行。

关于浮点单元的模拟，这一部分的内容基本都在 arch/mips/math-emu/cp1emu.c 文件中。init 进程会第一次执行到 CP1 指令时，发生 CpU 例外，Linux 判断到这一例外之后进入 handle_cpu 函数，判断 CP1 的状态并打开软件模拟。之后用户进程每次发生 CpU 例外时，都会陷入 Linux 内核，由内核进行软件模拟。如果 Linux 发现不是用户进程发生的 CpU 例外，内核会直接 panic。由于软件模拟的速度需要消耗上百个周期，所以浮点运算密集的程序在我们的 CPU 上面运行会非常慢。

附录 1. 指令列表

ADD Add Word
ADDI Add Immediate Word
ADDIU Add Immediate Unsigned Word
ADDU Add Unsigned Word
CLO Count Leading Ones in Word
CLZ Count Leading Zeros in Word
DIV Divide Word
DIVU Divide Unsigned Word
MADD Multiply and Add Word to Hi, Lo
MADDU Multiply and Add Unsigned Word to Hi, Lo
MSUB Multiply and Subtract Word to Hi, Lo

[2] 参考 M. Jones， Linux 引导过程内幕，https://www.ibm.com/developerworks/cn/linux/l-linuxboot/index.html，原文为 i386 架构，但是系统启动的整体思路是一致的。MIPS 架构通常使用 PMON 或者 u-boot 作为引导加载程序。

MSUBU Multiply and Subtract Unsigned Word to Hi, Lo
MUL Multiply Word to GPR
MULT Multiply Word
MULTU Multiply Unsigned Word
SLT Set on Less Than
SLTI Set on Less Than Immediate
SLTIU Set on Less Than Immediate Unsigned
SLTU Set on Less Than Unsigned
SUB Subtract Word
SUBU Subtract Unsigned Word
BEQ Branch on Equal
BGEZ Branch on Greater Than or Equal to Zero
BGEZAL Branch on Greater Than or Equal to Zero and Link
BGTZ Branch on Greater Than Zero
BLEZ Branch on Less Than or Equal to Zero
BLTZ Branch on Less Than Zero
BLTZAL Branch on Less Than Zero and Link
BNE Branch on Not Equal
J Jump
JAL Jump and Link
JALR Jump and Link Register
JR Jump Register
LB Load Byte
LBU Load Byte Unsigned
LH Load Halfword
LHU Load Halfword Unsigned
LL Load Linked Word
LW Load Word
LWL Load Word Left
LWR Load Word Right
PREF Prefetch
SB Store Byte
SC Store Conditional Word
SH Store Halfword
SW Store Word
SWL Store Word Left
SWR Store Word Right
SYNC Synchronize Shared Memory
AND And
ANDI And Immediate
LUI Load Upper Immediate
NOR Not Or
OR Or
ORI Or Immediate

XOR Exclusive Or
XORI Exclusive Or Immediate
MFHI Move From HI Register
MFLO Move From LO Register
MOVF Move Conditional on Floating Point False
MOVN Move Conditional on Not Zero
MOVT Move Conditional on Floating Point True
MOVZ Move Conditional on Zero
MTHI Move To HI Register
MTLO Move To LO Register
SLL Shift Word Left Logical
SLLV Shift Word Left Logical Variable
SRA Shift Word Right Arithmetic
SRAV Shift Word Right Arithmetic Variable
SRL Shift Word Right Logical
SRLV Shift Word Right Logical Variable
BREAK Breakpoint
SYSCALL System Call
TEQ Trap if Equal
TEQI Trap if Equal Immediate
TGE Trap if Greater or Equal
TGEI Trap if Greater of Equal Immediate
TGEIU Trap if Greater or Equal Immediate Unsigned
TGEU Trap if Greater or Equal Unsigned
TLT Trap if Less Than
TLTI Trap if Less Than Immediate
TLTIU Trap if Less Than Immediate Unsigned
TLTU Trap if Less Than Unsigned
TNE Trap if Not Equal
TNEI Trap if Not Equal Immediate
BEQL Branch on Equal Likely
BGEZALL Branch on Greater Than or Equal to Zero and Link Likely
BGEZL Branch on Greater Than or Equal to Zero Likely
BGTZL Branch on Greater Than Zero Likely
BLEZL Branch on Less Than or Equal to Zero Likely
BLTZALL Branch on Less Than Zero and Link Likely
BLTZL Branch on Less Than Zero Likely
BNEL Branch on Not Equal Likely
CACHE Perform Cache Operation
ERET Exception Return
MFC0 Move from Coprocessor 0
MTC0 Move to Coprocessor 0
TLBP Probe TLB for Matching Entry
TLBR Read Indexed TLB Entry

TLBWI Write Indexed TLB Entry
TLBWR Write Random TLB Entry
WAIT Enter Standby Mode

附录 2. 中断例外支持

Exception Code Value		Mnemonic	Description
Decimal	**Hexadecimal**		
0	**0x00**	Int	Interrupt
1	**0x01**	Mod	TLB modification exception
2	**0x02**	TLBL	TLB exception (load or instruction fetch)
3	**0x03**	TLBS	TLB exception (store)
4	**0x04**	AdEL	Address error exception (load or instruction fetch)
5	**0x05**	AdES	Address error exception (store)
8	**0x08**	Sys	Syscall exception
9	**0x09**	Bp	Breakpoint exception
10	**0x0a**	RI	Reserved instruction exception
11	**0x0b**	CpU	Coprocessor Unusable exception
12	**0x0c**	Ov	Arithmetic Overflow exception
13	**0x0d**	Tr	Trap exception

附录 3. CP0 寄存器

Index Register (CP0 Register 0, Select 0)
Random Register (CP0 Register 1, Select 0)
EntryLo0, EntryLo1 (CP0 Registers 2 and 3, Select 0)
Context Register (CP0 Register 4, Select 0)
PageMask Register (CP0 Register 5, Select 0)
Wired Register (CP0 Register 6, Select 0)
BadVAddr Register (CP0 Register 8, Select 0)
Count Register (CP0 Register 9, Select 0)
EntryHi Register (CP0 Register 10, Select 0)
Compare Register (CP0 Register 11, Select 0)
Status Register (CP Register 12, Select 0)
Cause Register (CP0 Register 13, Select 0)
Exception Program Counter (CP0 Register 14, Select 0)
Processor Identification (CP0 Register 15, Select 0)
EBase Register (CP0 Register 15, Select 1)
Configuration Register (CP0 Register 16, Select 0)
Configuration Register 1 (CP0 Register 16, Select 1)
TagLo Register (CP0 Register 28, Select 0, 2)
TagHi Register (CP0 Register 29, Select 0, 2)