第五届“龙芯杯”全国大学生系统能力培养大赛

清华大学“琉璃晨露”队

LLCL-MIPS
决赛设计报告

基于 SpinalHDL 的 MIPS32 Rev1 处理器

2021 年 8 月 14 日

黄嘉良

杨倚天

余泰来

刘松铭

目录

Chapter 1

总体介绍

1.1 背景

该项目旨在于龙芯杯官方提供的 FPGA 上实现支持 MIPS32 Rev 1 的小端序 CPU，并驱动板上外设，能够通过官方的功能、性能测试，并能够运行 U-Boot 引导程序，并启动 UCore 操作系统以及 Linux 操作系统。

1.2 硬件语言

本项目采用**非传统的硬件描述语言**（比如 Vhdl、Verilog 或 SystemVerilog），使用基于 Scala 构筑的硬件描述语言 SpinalHDL[1]，得益于 Scala 强大的语法表达能力、抽象能力以及面向对象的特性，能够更便捷、优雅地表达硬件。

1.3 MIPS ISA 支持

LLCL-MIPS 实现了共 91 条指令（涵盖运算、分支、访存、特权等），基本涵盖了支持 MIPS32 Rev 1 的编译器能够生成（除浮点指令外）的绝大多数指令，能够覆盖运行 Linux 所需的指令以及常见的应用。完整的指令支持可以参考2.2节。

LLCL-MIPS 实现了 32 个通用寄存器以及运行操作系统所必须的 CP0 寄存器。

1.4 CPU 架构

LLCL-MIPS 最终的架构是 8 级流水线，顺序双发射。其中取指阶段为 3 级，访存阶段为 2 级，特性包括分支预测（四路组相连 btb, bht, pht），缓存（2 路组相连），指令队列。由于 MIPS 的延迟槽在双发射中非常难处理，指令队列能够保证分支和延迟槽一定一起发射。

[1] https://spinalhdl.github.io/SpinalDoc-RTD/index.html

1.5 缓存

考虑到实际系统中访问外部存储的时间开销巨大，我们实现了基于 LRU 替换算法的组相连 L1 Cache，并支持 MIPS32 Rev 1 中的针对 L1 Cache 的 Cache 指令，能够正确刷新相应的 Cache Line。

1.6 Soc 设计

比赛官方提供的开发板上有丰富的外设资源，并且基于已有的工作和 Linux 支持的丰富驱动，能够使用这一系列外设进行更多的功能展示。截止目前我们的片上系统支持：

1. CPU：对外暴露三条 AXI 总线的 CPU
2. DRAM：支持板载 128MiB DDR3 颗粒 SDRAM 作为内存
3. 串口：UART 16550A
4. 片内 RAM：用于存放和运行一级引导程序
5. Flash：主要使用 FPGA 配置 Flash，存放 U-boot 二级引导程序
6. 以太网：能够通过网络加载操作系统到指定内存区域和挂载 NFS
7. GPIO：基于龙芯已实现的 GPIO 控制器进行调整
8. 图像输出：能够通过 VGA 进行输出
9. PS2：能使用 PS2 键盘在 Linux 的 tty1 上进行输入

1.7 系统软件

为了验证我们 CPU 实现的正确性，我们支持从简单到复杂的多个系统软件：

1. 清华 MIPS 32 位版本监控程序 [2]
2. NonTrivialMIPS 项目的 TrivialBootLoader[3]
3. U-Boot 引导程序较新版本 v2021.04
4. 清华大学 uCore-mips 操作系统[4]
5. Linux 操作系统最新稳定版本 v5.13

[2] https://github.com/z4yx/supervisor-mips32/
[3] https://github.com/trivialmips/TrivialMIPS_Software
[4] https://github.com/chyh1990/ucore-thumips

Chapter 2

CPU 详细设计

2.1 CPU 基本架构

LLCL-MIPS 的架构是 8 级流水顺序双发射，除了常规的优化模块还实现了延迟执行。下面就前端（取指阶段，共 3 级）和后端（译码、运算、访存以及写回共 5 个阶段）进行详细介绍：

2.1.1 前端

2.1.1.1 第一级

流水第一级拿到 PC 的值，向 ICache 发送取指请求，此时也通过 MMU 进行地址翻译。其中 PC 的更新受到以下几个因素的影响：分支预测的结果、分支预测错误纠正的结果、异常的结果以及正常的 pc 递增。这个阶段还会把当前的 pc 交给分支预测模块

当第二级流水出现预测到的分支时，会刷新第一级和第二级流水；当发生纠错和异常时会刷新整个前端部分（包括整个指令队列），所以分支预测错误的开销为 3 个周期。

2.1.1.2 第二级

流水第二级 ICache 根据读出的数据判断是否命中，如果命中则直接返回数据，不会有任何气泡插入，否则开始从内存中读取数据。这个阶段也会得到分支预测的结果，然后转给第一级的 PC 模块，如果分支预测第一条指令为分支（注意双发射一次会取回 2 条指令），那么会将下一次读出的指令全部作废（即分支预测成功的开销为 1 个周期），如果第二条指令为分支，那么会保留下一次读出指令的第一条（因为它是延迟槽）

当接收到 PC 模块传入的刷新信号时，会作废当前阶段送给第三级的指令

2.1.1.3 第三级

流水第三级会将第二级读出的指令放入指令队列中。如果此时后端已经需要指令而指令队列为空，那么它能够直通给后端（byPass），否则会压入指令队列。

指令队列的好处有两个，第一个是能重整指令流，可以进行预先的译码判断是否是分支指令，从而保证发射的时候分支和延迟槽一起发射；第二点是指令队列可以把前后端解耦开，只要队列没满，就可以一直取指而不会因为后面的阶段而阻塞。

2.1.2 后端

2.1.2.1 第四级

这个阶段进行译码和部分的数据前传，总体较为朴素。也可以将本阶段理解为发射阶段，简单理解的话，该阶段会保证有数据冲突的两条指令不会同时发射、两条访存指令不会同时发射、两条乘除法不能同时发射（因为我们只有一个乘法器和一个除法器）、两条写 CP0 的指令不能同时发射。

2.1.2.2 第五级

这个阶段进行运算，两条流水线共用一个复杂计算模块（包含一个乘法器和一个除法器），并且两条流水线各有一个简单计算模块，对于分支指令是否跳转的判断和纠正也在这个阶段。此外访存阶段的地址计算和地址翻译也放在了执行阶段。

2.1.2.3 第六级

这个阶段是访存的第一个阶段。如果是 store 则需要保证在这个阶段就已经拿到了要写入内存的值，这是因为我们的 DCache 有一个 Write Buffer，对于需要真的写回内存的值会先进入 Write Buffer，从而不会阻塞后面的指令。

这个阶段首先会给 DCache 发起访存请求。同时会查询 Write Buffer，如果 Write Buffer 写命中则会直接把本次写回内存的值写合并进 Write Buffer，如果是读命中则会将命中的结果流水到下一级。需要注意的是 DCache 仅有一个端口暴露给 CPU，因此需要选择哪一条流水线的请求传给 DCache。

这个阶段还会处理异常、对 CP0 的写入以及 Hi Lo 寄存器的写入：实际上所有指令的异常都能够在这一阶段被检测出来。因此会将发生异常的流水线将信息送给异常处理模块。

2.1.2.4 第七级

这是访存的第二个阶段。如果读或写 Cache 命中则会将访存第一阶段读出的数据进行选择和处理后直接完成事务，否则无论是读缺失还是写缺失都需要先进行替换和读内存。对于被替换出去的 Cache Line 会进入 Write Buffer，然后由 Write Buffer 独立控制写回内存的逻辑。

2.1.2.5 第八级

这一级将结果写回寄存器堆，因为是双发射，因此寄存器堆支持 4 读 2 写。

2.2 指令支持

LLCL-MIPS 共实现了 91 条指令，覆盖了启动 Linux 所需的全部指令，作为参考，下面列出支持的指令

- **逻辑指令** AND, ANDI, NOR, OR, ORI, XOR, XORI, SLL, SLLV, SRA, SRAV, SRL, SRLV
- **运算指令** ADD, ADDI, ADDU, ADDIU, SUB, SUBU, DIV, DIVU, MUL, MULT, MULTU, CLO, CLZ, MADD, MADDU, MSUB, MSUBU

- **分支指令** BEQ, BNE, BGEZ, BGTZ, BLEZ, BLTZ, BGEZAL, BLTZAL, J, JAL, JR, JALR, JR.HB
- **自陷指令** TEQ, TEQI, TNE, TNEI, TGE, TGEI, TGEU, TGEIU, TLT, TLTI, TLTU, TLTIU
- **特权指令** CACHE, SYSCALL, BREAK, MFC0, MTC0, TLBP, TLBR, TLBWI, TLBWR, ERET, WAIT
- **移动指令** LUI, SLT, SLTI, SLTU, SLTIU, MFLO, MFHI, MTHI, MTLO, MOVN, MOVZ
- **访存指令** LB, LBU, LH, LHU, LW, SB, SH, SW, LWL, LWR, SWL, SWR
- **其他指令** SYNC, PREF

由于 LLCL-MIPS 是单核 CPU，并且 Linux 支持以关中断的方式实现同步互斥，因此没有实现 LL/SC 指令

2.3 协处理器

在协处理器方面，LLCL-MIPS 仅支持 COP0，它能够提供对操作系统的必要支持。LLCL-MIPS 遵循 MIPS 规范共实现了 19 个 CP0 寄存器：

CP0 寄存器	Num/Sel	CP0 寄存器	Num/Sel	CP0 寄存器	Num/Sel
Index	0/0	BadVaddr	8/0	EPC	14/0
Random	1/0	Count	9/0	PRId	15/0
EntryLo0	2/0	EntryHi	10/0	EBase	15/1
EntryLo1	3/0	Compare	11/0	Config0	16/0
Context	4/0	Status	12/0	Config1	16/1
PageMask	5/0	Cause	13/0	ErrorEPC	30/0
Wired	6/0				

表 2.1: LLCL-MIPS 实现的 COP0 寄存器

2.4 内存管理

LLCL-MIPS 支持 MIPS32 中的基于 TLB 的虚拟地址翻译，TLB 项数可配置（需要是 2 的幂），目前采用 32 项 TLB。在取指第一阶段和 EX 阶段会查询 TLB 得到相应的物理地址：实现

区域	范围	是否 Cache	是否 Map
useg/kuseg	0x0000_0000~0x7FFF-FFFF	是	是
kseg0	0x8000_0000~0x9FFF_FFFF	是	否
kseg1	0xA000_0000~0xBFFF_FFFF	否	否
kseg3	0xE000_0000~0xFFFF_FFFF	是	是

表 2.2: 需要支持的段及其访存行为

上需要注意：

1. 在 kseg1，或者 Mapped 段且`cacheAttr = 2`，或者在 kseg0 并且`Config0.K0 = 2`，或者在 kuseg 且`Status.ERL = 1`才会 Uncached；
2. 在用户态访问 Kernel 权限的地址段会触发地址异常。

2.5 异常中断

为了运行操作系统，LLCL-MIPS 除了支持比赛要求的异常外，还需要支持其余的一些异常，下面列出 LLCL-MIPS 支持的所有异常：

类别	ExcCode	说明	类别	ExcCode	说明
Int	0x00	中断	Sys	0x08	系统调用
Mod	0x01	TLB Mod 异常	Bp	0x09	断点异常
TLBL	0x02	TLB Load 异常	RI	0x0a	保留指令异常
TLBS	0x03	TLB Store 异常	CpU	0x0b	协处理器不可用异常
AdEL	0x04	地址 Load 异常	Ov	0x0c	算数溢出异常
AdES	0x05	地址 Store 异常	Trap	0x0d	陷入异常

表 2.3: LLCL-MIPS 支持的异常

实现上值得注意的几点是：

1. 因为 LLCL-MIPS 仅支持协处理器 CP0，因此触发 CpU 异常后 Cause.CE 为 0；
2. 要发生中断，必须`Status.EXL = 0 & Status.ERL = 0 & Status.IE = 1`；
3. 由于我们实现了 MIPS32 Rev 2 中的 EBase 寄存器，当`Status.BEV=0`的时候，使用 EBase 的值作为异常向量的基址。

2.6 缓存

无论是在性能测试，还是实际运行操作系统时，访存的延迟都很高，因此 LLCL-MIPS 提供 L1 指令缓存和数据缓存。指令缓存和数据缓存均采用虚拟索引和物理标志（VIPT），多路组相连，单路 4KiB，并且行大小和路数可配置。

2.6.1 指令 Cache

指令 Cache 的标志 (tag) 和数据均采用 Xilinx 的 BRAM 实现；替换策略采用基于树的 (Tree-Based) 伪 LRU 算法。当指令命中 Cache 的时候当前周期能够返回数据，不会暂停流水线；对于指令 Miss 的情况，会通过 AXI 总线的 Burst 模式从内存中加载一个 Cache Line 大小的数据，替换（如果所有的路都满）或者加载进（优先选择没有被填充的路）合适的位置；

2.6.2 数据 Cache

数据 Cache 中标志 (tag) 采用 LUT RAM 实现，数据采用 BRAM 实现，使用写回、写分配的策略，对于需要写回的数据，通过一个额外的写缓存（WriteBuffer）进行加速。替换策略采用基于树的 (Tree-Based) 伪 LRU 算法。

当数据命中 Write Buffer 或者 Cache 的时候可以当前周期完成事务，不会暂停流水线。对于数据 Miss 的情况比较复杂：

- 当发生读缺失的时候，如果待加载的位置是脏 (Dirty)，那么首先要将 Cache 中该行压入 WriteBuffer，然后从内存中加载对应的 Cache Line；
- 当发生写缺失的时候，如果待加载的位置是脏 (Dirty)，那么首先要将 Cache 中该行压入 WriteBuffer，然后从内存中加载对应的 Cache Line，最后再写入数据。

此外 MIPS32 Rev 1 要求提供 CACHE 指令，该指令繁杂且细节相当多，不便也不必全部实现，因此 LLCL-MIPS 仅支持对 ICache 和 DCache 刷新特定地址对应索引处的 Cache Line。经过最终的检验，能够支持 Linux 操作系统。

2.6.3 访存行为

因为 MIPS 虚存系统需要同时支持四段有不同访问行为的区域（参考2.2）

我们在 DCache 中额外处理了 Uncache 的访存请求，例如当一条指令访存的地址落在 kseg0 区域，那么此时不会经过 DCache，而是直接将请求放到总线上。因此我们的 CPU 对外一共暴露出三条 AXI 总线：指令缓存、数据缓存、不经过缓存的请求。

Chapter 3

Soc 设计

3.1 Soc 总体情况

顶层设计和地址分配均通过 Vivado 的 Block Design 工具进行设计，整体的结构如下：

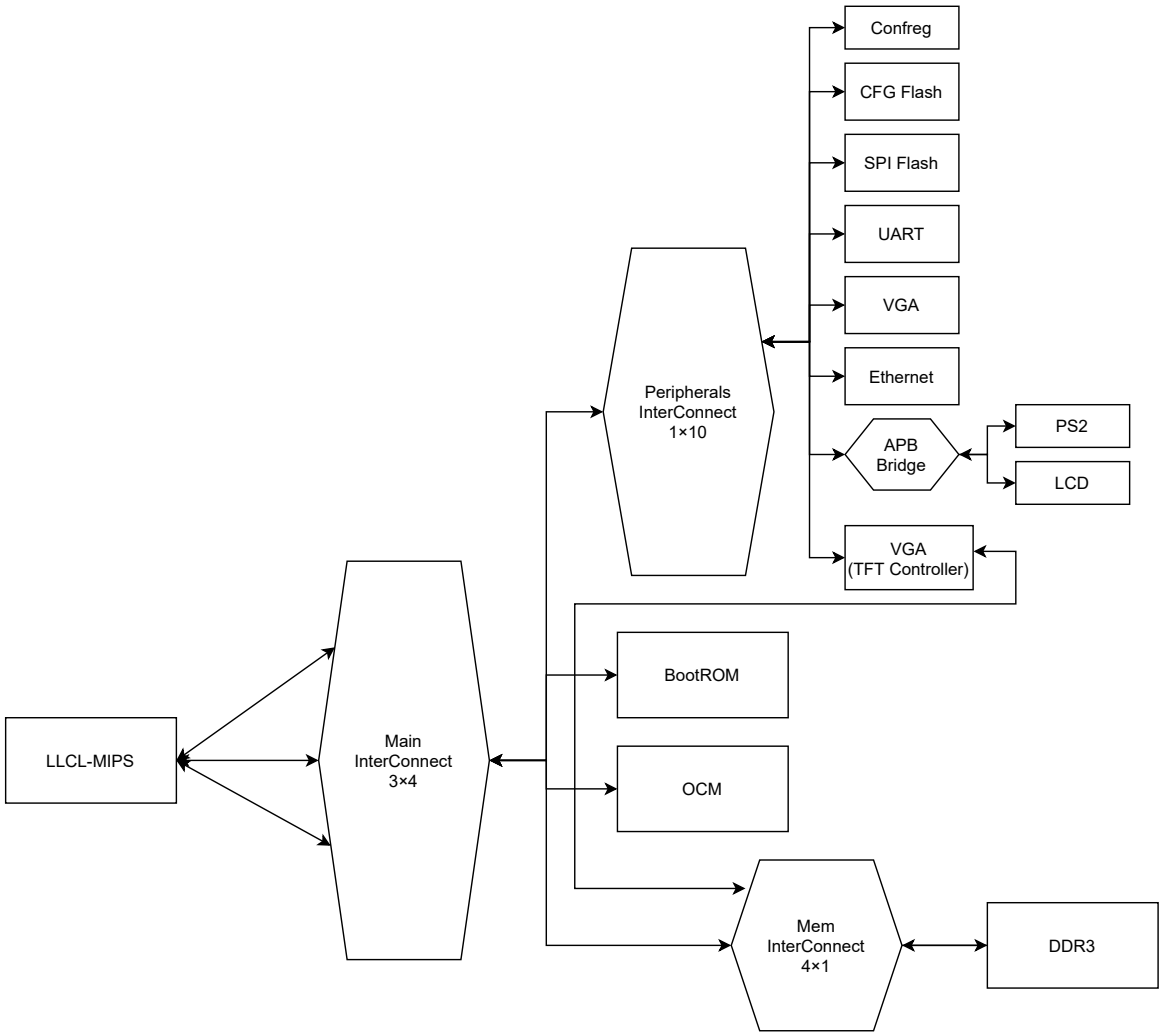

图 3.1: Soc

由于 Linux 已经带有丰富的外设驱动，并且 Xilinx 中已经带有丰富的外设控制器，因此在正确实现 CPU 后，可以方便地搭建起 Soc，驱动板上的外设。LLCL -MIPS 支持的外设如下：

1. DRAM：支持以板载 128MiB DDR3 作为内存；

2. 串口：UART 16550A；
3. 片内 RAM：用于存放和运行一级引导程序，其中 BootROM 的大小为 128KiB，OCM 大小为 64KiB；
4. Flash：主要使用 FPGA 配置 Flash，存放 U-boot 二级引导程序；
5. 以太网：运行 U-Boot 后能够通过以太网加载 Linux 操作系统和挂载 NFS；
6. GPIO：基于龙芯已实现的 GPIO 控制器；
7. VGA：能够在 VGA 上进行图像输出
8. PS2：能够外接键盘进行输入

3.2 地址映射

根据我们的 Soc 设计3.1，能够发现有多个 AXI 互联模块，可以在 Block Design 中为其方便地配置对应的地址映射：

设备	接口类型	类别	Offset Addr	Range	High Addr
DDR3 Controller	S_AXI	memaddr	0x0000_0000	128M	0x07FF_FFFF
OCM Controller	S_AXI	Mem0	0x0800_0000	64K	0x0800_FFFF
CFG Flash	AXI_FULL	MEM0	0x1A00_0000	16M	0x1AFF_FFFF
Ethernet	S_AXI	Reg	0x1C00_0000	64K	0x1C00_FFFF
VGA	S_AXI_MM	Reg	0x1C01_0000	64K	0x1C01_FFFF
PS2	APB	Reg	0x1C02_0000	4K	0x1C02_0FFF
nt35510 Controller(LCD)	APB	Reg	0x1C03_0000	4K	0x1C03_0FFF
SPI Flash	AXI_LITE	Reg	0x1C04_0000	4K	0x1C04_0FFF
Frame Buffer Reader	s_axi_CTRL	Reg	0x1C05_0000	64K	0x1C05_FFFF
Frame Buffer Writer	s_axi_CTRL	Reg	0x1C06_0000	64K	0x1C06_FFFF
Interrupt Controller	s_axi	Reg	0x1D00_0000	64K	0x1D00_FFFF
Bootram Controller	S_AXI	Mem0	0x1FC0_0000	128K	0x1FC1_FFFF
Uart	S_AXI	Reg	0x1FD0_2000	8K	0x1FD0_3FFF
Confreg	interface_aximm	reg0	0x1FF0_0000	64K	0x1FF0_FFFF

表 3.1: 地址分配

3.3 中断连接

MIPS 一共支持 8 个中断，其中 2 个是软件中断，6 个是硬件中断。LLCL-MIPS 将 `IP[7]` 单独留给时钟中断，最后的中断配置如下：

中断外设	接收方	中断号
串口	CPU	2
PS2	CPU	3
AXI 中断控制器	CPU	6
以太网	AXI 中断控制器	0
SPI Flash	AXI 中断控制器	1
CFG Flash	AXI 中断控制器	2

表 3.2: 中断连接情况

Chapter 4

系统软件

4.1 监控程序

监控程序仅涵盖 20 多条指令，支持串口作为输入输出设备，能够提供基本的交互功能，并且它也是比赛要求运行的系统测试，因此不再赘述。

4.2 引导程序

4.2.1 Bootloader

因为 Linux 或者 uCore 这样的操作系统，编译后的 elf 文件较大（其中 UCore 大小为 1.2MiB，最新版本的 Linux 内核编译后 uImage 也超过 8MiB），无法像功能测试或者性能测试那样直接放到 FPGA 片内的 RAM 上，因此为了启动这些操作系统，需要通过引导程序从网络或者 Flash 上加载到内存中运行。U-Boot 是嵌入式系统普遍使用的引导程序，但是其编译之后的大小也在 400KiB 以上，因此我们还需要额外一级引导程序，来加载 U-Boot，从而加载 Linux。

最低一级的引导程序是一个**极小**的裸机引导程序，能够直接放到板内的 RAM 上，提供从 Flash 加载 elf 的基本功能。由于之前有较为完善的工作[1]，我们采用了 NonTrivialMIPS 项目的一级引导程序 TrivialBootloader，支持启动时内存检查、内存初始化，以及从 Flash 的指定地址拷贝 elf 格式文件到内存中并执行。通过 TrivialBootloader 可以直接加载 uCore 操作系统，也能够先启动 U-Boot 然后再加载 Linux 操作系统。

4.2.2 U-Boot

我们在之前项目[2]的基础上，从 v2019.07 升级到了较新的 U-Boot 稳定版本 v2021.04 作为 Linux 的引导程序，U-Boot 本身包含了许多硬件的驱动，并且也具有基本的交互功能，支持从 CFG Flash 中拷贝 Linux 操作系统编译后的 uImage 到指定内存区域，或者从以太网口加载 uImage 到制定内存区域，再进行引导。当我们移植最新新的稳定 Linux 版本后，内核大小已经接近 9MiB，在 CFG Flash 中放置 bit 文件和 u-Boot 后，很难再放下整个 Linux 内核，方便的

[1] https://github.com/trivialmips/TrivialMIPS_Software

[2] https://github.com/trivialmips/u-boot-trivialmips

是，U-Boot 支持 tftp 从网络加载 uImage 到内存指定区域，然后再进行引导，这样不仅方便修改内核，也解决了内核存放在哪儿的问题。U-Boot 编译方法如下:

```
make CROSS_COMPILE=mipsel-linux-gnu- llcl_defconfig
make CROSS_COMPILE=mipsel-linux-gnu-
mipsel-linux-gnu-strip u-boot
```

4.3 uCore-thumips

uCore-thumips 是针对简化后的 MIPS32 实现，uCore 项目自带针对 MIPS 平台的 Bootloader，同时涵盖了完整的操作系统支持：包括异常处理、内存管理以及进程切换，能够较为全面地展现 CPU 的基本能力，值得注意的是 uCore-thumips 最初是针对一个 MIPS32 R1 的简化版本，部分特性如延迟槽它并不支持。由于之前已经有针对 MIPS32 R1 的适配版本[3]，因此无需再做重复的工作。配置好`kern/include/thumips.h`中的串口与设备内存相关参数后，再进行交叉编译就能得到`ucore-kernel-initrd`，这是 uCore 内核的 elf，由于我们已经有更为完善引导程序 TrivialBootloader 了，不必再使用 uCore 项目自带的 Bootloader。

4.4 Linux

Linux 是著名的开源操作系统之一，支持非常多的指令集架构。在之前的项目基础上[4]，将 Linux 从 v5.2.9 升级到了最新的稳定版本 v5.13，并适配了 LLCL -MIPS：LLCL-MIPS 基于 MIPS 4Kc CPU 作为基础进行移植（因为这是 MIPS32 R1 的 CPU，和 LLCL-MIPS 吻合），移植过程中主要遇到了以下问题：

1. 尽管在`arch/mips/include/asm/mach-*/cpu-features-overrides.h`中已经去除了 PREF 指令，但最终编译出来的内核仍然带有 PREF；
2. 直接下载 busybox 1.32.1 并编译作为 Linux 启动后的基本组件，但是在 Debian 上安装的`mipsel-linux-gnu-`交叉编译器下会包含 MIPS32 Rev2 的指令，并且缺少对软浮点的支持。

针对以上问题，因为 LLCL-MIPS 没有预取功能，因此将 PREF 实现为空操作；此外采用较新的 buildroot v2021.02.3 版本来提供 Linux 内核的组件，它会自动下载对应的工具链，从而能够支持 MIPS32 Rev1 以及软浮点。

此外，LLCL-MIPS 基本实现了所有 MIPS32 Rev1 必须的 CP0 寄存器以及指令，因此基本不需要修改内核代码，就能够正确运行 Linux 操作系统。启动操作系统后，得益于 buildroot 提供的基本用户态组件，可以运行一些基本的应用来验证我们 CPU 实现的正确性：例如 vi 编辑器、top 系统状态管理器等。编译 Linux 的方法即：

[3] https://github.com/trivialmips/ucore-thumips

[4] https://github.com/trivialmips/linux-nontrivial-mips

```
make CROSS_COMPILE=mipsel-linux-gnu- ARCH=mips llcl_defconfig
make CROSS_COMPILE=mipsel-linux-gnu- ARCH=mips -j8 uImage
```

编译后，能够在arch/mips/boot/下找到uImage，这是引导程序 U-Boot 能够加载的格式。

附录 A

附录

A.1 致谢

LLCL-MIPS 开发过程中离不开往届参赛选手的宝贵经验，许多软件能够顺利运行在 LLCL-MIPS 上也都得益于之前完善的工作和文档。感谢陈嘉杰、陈晟祺、高一川、王邈对本项目的倾力帮助。此外感谢指导教师陈康、刘卫东、陆游游和李山山老师的建议。

参考资料

[1] *AMBA AXI and ACE Protocol Specification.*

[2] *MIPS32 Architecture For Programmers Volume I: Introduction to the MIPS32 Architecture.*

[3] *MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set.*

[4] *MIPS32 Architecture For Programmers Volume III: The MIPS32 Privileged Resource Architecture.*

[5] 姚永斌. 超标量处理器设计. 清华大学出版社.

[6] 张宇翔，王邈，刘家昌. *NaiveMIPS* 设计文档. https://github.com/z4yx/NaiveMIPS-HDL/tree/brd-NSCSCC/documentation.

[7] 陈晟祺，周聿浩，刘晓义，陈嘉杰. *NonTrivialMIPS* 设计文档. https://github.com/trivialmips/nontrivial-mips/tree/master/report.