“龙芯杯”第五届全国大学生计算机系统能力培养大赛
清华大学“MIPS Great Again”队

“MIPS Great Again”队：初赛设计报告

刘泓尊 1456437967@qq.com

刘子昂 878574650@qq.com

闭浩扬 1456437967@qq.com

2021 年 8 月 16 日

目录

1 概述

本项目是在龙芯提供的 FPGA 实验平台 (其主要部件为 Xilinx 的 Artix 7 系列 FPGA) 上设计并实现的基于 MIPS32 的 CPU，并接入龙芯提供的部分外设接口，使得 CPU 连接实验板上周边硬件，成为一个较为简单的片上系统 (SoC). 除了满足初赛的功能和性能测试要求外，本项目能支持 MIPS32 Rev1 指令集较为完整的子集和 MIPS32 Rev2 指令集的部分指令，可配置基于 TLB 的虚存管理，并可配置 32 位浮点协处理器 1。本项目实现了带有 **2-bit 分支预测**的 **9 级流水线**的单发射处理器，分别采用 **4 路 4KB** 的**指令缓存**和**数据缓存**，最终能够运行龙芯提供的功能测试、性能测试和系统测试监控程序 (开启中断版本和开启 TLB 版本)。

2 CPU

2.1 流水线结构

CPU 采用单发射顺序执行，共有 9 级流水，并带有分支预测单元，可配置协处理器 1 和 TLB 单元。为了提高 IPC 同时提升整体的频率，指令和数据缓存使用 3 级流水，保证在命中情况下 IPC=1. 下面详细介绍各流水段的功能。流水线架构如图1所示。

2.1.1 取指阶段

取指阶段共有 3 级流水，最后将指令和 PC 打包流出到译码阶段。

取指第一阶段，根据分支与否、跳转与否以及是否发生异常，计算下一次的取指地址 PC。同时将本周期 PC 送入 Cache, 作为查询条件。指令 Cache 使用 BRAM 实现，在第二阶段可以得到结果。

取指第二阶段，得到指令 Cache 访问结果，得到命中信息与对应指令。本阶段同时进行分支预测，分支预测单元采用 2-bit 分支预测目标缓冲区 (BTB)。BTB 使用 LUTRAM(Xilinx Parameterized Macros) 实现，在当前周期就可以得到结果。之所以在取指第二阶段进行分支预测，是因为 MIPS 的延迟槽机制要求分支的下一条指令也必须执行。

取指第三阶段，如果上一流水段判断未命中，则启动 AXI 读事务状态机，从内存对应位置按 AXI Burst 方式读取一个缓存行 (本项目中为 32Byte) 的数据, 存入 cache，并将指令和 PC 流出。若命中，直接将 Cache 中的指令和 PC 流出。

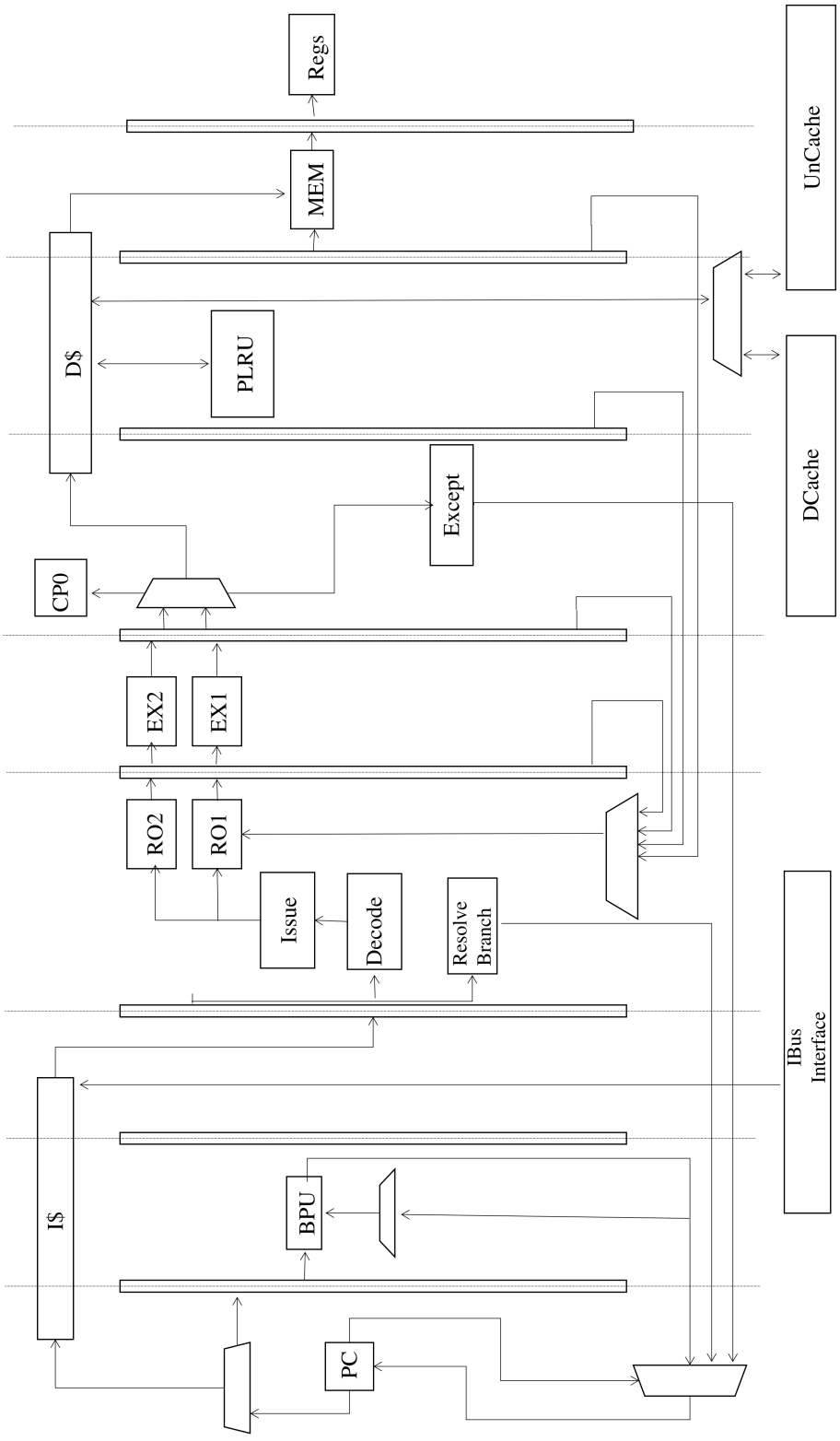
Regs
MEM
UnCache
D$
PLRU
DCache
Except
CP0
EX2
EX1
RO2
RO1
Issue
Decode
Resolve
Branch
IBus
Interface
I$
BPU
PC

图 1: CPU 流水线结构

2.1.2 译码阶段

译码阶段会读取操作数，并对流出的所有类型指令做简单的翻译，以完成指令分发。所需操作数可能会存在数据相关，所以译码阶段接收执行阶段、访存阶段、写回阶段的数据前推路径，以减少暂停的周期数。如果存在 load 数据相关，则暂停至访存阶段得到结果。

同时译码阶段进行分支判断，修正 BTB 的预测结果，以维护正确的指令流。所以 PC 会根据译码阶段的信息进行更新，同时也会更新 BTB。

2.1.3 执行阶段

执行阶段会生成指令的结果，同时生成访存请求和异常请求。本流水段设置若干加法器、一个乘法器和一个除法器，支持译码阶段每周期发射一条指令。乘除法或浮点指令可能造成流水线暂停，同时，在本阶段判断是否存在 CP0 冒险，并通过暂停流水线的方式解决 CP0-hazard(主要是 MFC0、MTC0 等特权指令).

同时，CPU 会在本阶段访问 MMU 进行地址翻译，得到物理地址或访存异常信息。如果开启协处理器 1，则本阶段也会执行浮点计算。MMU 也支持开启或关闭 TLB。

2.1.4 访存阶段

访存阶段有 3 级流水。负责异常处理、CP0 更新、TLB 更新、访问数据 Cache 以及处理 AXI 事务。

访存第一阶段，会进行异常处理，并给出 CP0 寄存器和 TLB 的更新信息，这些寄存器在下一阶段得到更新。如果不存在异常，则将访存地址送入 Cache，数据缓存由 BRAM 实现，将在下一流水段返回结果。

访存第二阶段，得到数据缓存的命中信息与数据。缓存替换策略采用 PLRU(Pseudo-LRU) 算法，在此阶段更新 PLRU 状态。

访存第三阶段，如果上一阶段缓存未命中，则在本阶段开启 AXI 读写事务。对于 Cachable 段的访存，会将 dirty 的块先通过 burst 方式向数据总线发送写请求，之后再开启读事务，读入一个缓存行的数据。对于 Uncachable 段访存，直接访问对应内存地址，读取 4Byte 的数据。得到数据之后更新写回寄存器的信息，并传递给下一级流水。

2.1.5 写回阶段

写回阶段根据来自执行和访存阶段的结果，进行通用寄存器、HILO、LLbit、浮点寄存器的写回操作。

2.2 指令集

下方按照功能划分列举了 CPU 所支持的 MIPS 指令，已经是初赛要求指令集的超集，各条指令的具体编码以及功能在 MIPS 文档中有详细的描述。

- **自陷指令** TGE, TEGU, TLT, TLTU, TEQ, TNE, TGEI, TGEIU, TLTI, TLTIU, TEQI, TNEI
- **分支指令** BLTZ, BGEZ, BLTZAL, BGEZAL, BEQ, BNE, BLEZ, BGTZ, JR, JALR, J, JAL
- **逻辑指令** AND, OR, XOR, ANDI, ORI, XORI, NOR, SLL, SRL, SRA, SLLV, SRLV, SRAV

- **算术指令** ADD, ADDU, SUB, SUBU, ADDI, ADDIU, MUL, MULT, MULTU, DIV, DIVU, MADD, MADDU, MSUB, MSUBU, CLO, CLZ
- **访存指令** SB, SH, SW, SWL, SWR, LB, LH, LWL, LWR, LW, LBU, LHU, LL, SC
- **特权指令** SYSCALL, BREAK, TLBR, TLBWI, TLBWR, TLBP, ERET, MTC0, MFC0
- **条件移动指令** SLT, SLTU, SLTI, SLTIU, MOVN, MOVZ
- **无条件移动指令** LUI, SLT, SLTU, MFHI, MFLO, MTHI, MTLO
- **协处理器 1(浮点指令)** MOVF, MOVT, MFC1, MTC1, CFC1, CTC1, LWC1, SWC1, ADD.S, SUB.S, MUL.S, DIV.S, SQRT.S, ABS.S, MOV.S, NEG.S, ROUND.W.S, TRUNC.W.S, CEIL.W.S, FLOOR.W.S, CVT.W.S

2.3 协处理器 0

CP0 是 MIPS 规范中必要的一个协处理器，它提供了操作系统所必须的功能抽象，例如异常处理、内存管理和资源访问控制等。

在 CP0 中有多个 32 位寄存器，各个寄存器均通过 MTC0 和 MFC0 读写。另外，诸如 TLBWI、TLBWR 和 TLBP 等特权指令还有异常的发生也有可能会影响其值。

表1中列出了必须实现的 CP0 寄存器。

表 1: 必要的 CP0 寄存器

编号	名称	功能
8	BadVAddr	最近发生的与地址相关的异常所对应的地址
9	Count	计数器
11	Compare	计时中断控制器
12	Status	处理器状态及控制
13	Cause	上一次异常的原因
14	EPC	上一次异常发生的地址
15	PRId	处理器版本和标识符
16	Config0	处理器配置
30	ErrorEPC	上一次系统错误发生的地址

为了实现 TLB MMU 的功能，还需要表2中所列出的寄存器。

表 2: MMU 所需要的 CP0 寄存器

编号	名称	功能
0	Index	TLB 数组的索引
1	Random	随机数
2	EntryLo0	TLB 项的低位
3	EntryLo1	TLB 项的低位
4	Context	指向内存中页表入口的指针
5	PageMask	控制 TLB 的虚拟页大小
6	Wired	控制 TLB 中固定的页数
10	EntryHi	TLB 项的高位

同时，为了支持自定义异常向量，还需要额外实现一个 MIPS 32 Rev 2 中的 EBase 寄存器，这样就可以运行系统测试了。

2.4 协处理器 1

本项目可配置使用协处理器 1 的浮点功能，配备 32 个 32 位浮点寄存器，可进行单精度浮点运算并支持异常检测。其中单精度浮点运算使用 AXI 接口的 Xilinx IP Floating-point (7.1) 实现，极大方便了我们的开发。为了支持浮点异常控制，还需要实现 FCRs(Floating Point Control Registers) 寄存器的如表3字段。同时需要 `config1` 字段的最低位为 1 表示协处理器 1 可用。

表 3: 协处理器 1 需要的 FCRs 寄存器字段

编号	名称	功能
0	fcc	浮点比较条件码
1	fs	设定非规格化浮点结果置为 0
2	cause	浮点异常原因
3	enables	5 种浮点异常的使能
4	flags	没有使能浮点异常时记录异常原因
5	rm	浮点取整方式的设定

2.5 中断和异常

2.5.1 中断

MIPS 的中断一共有 8 个，从 0 开始编号。其中 0 号中断和 1 号中断是软件中断，由软件设置 Cause 寄存器中的对应位来触发。其余 6 个中断为硬件中断，由外部硬件触发。在实现中，由 Count/Compare 寄存器组合而成的定时中断的中断号为 7。

如果该中断满足触发条件则会触发异常并且进入中断处理程序。中断的触发要求如下条件全部满足：

1. `Status` 寄存器中对应的中断被打开；
2. 全局中断使能，即 `Status.IE` 为 1；
3. 当前不在异常处理程序中，即 `Status.EXL` 和 `Status.ERL` 为 0。

2.5.2 异常

MIPS 的异常是“精确异常”，也就是在异常发生前的指令都会执行完毕，异常发生之后的指令不会继续执行。在异常发生时，CPU 会跳转到对应的异常向量处执行异常处理代码并设置 CP0 中对应的寄存器记录异常的原因和一些额外的信息，同时还会进入 Kernel Mode。处理异常代码的异常向量由“基地址 + 偏移”来决定，偏移是根据异常来确定的，基地址是由 CP0 的 EBase 寄存器决定（对于初赛，我们设置为 0xbfc00180）。

在我们的实现中，在流水的各个阶段产生了异常，不会马上触发，而是被记录下来，顺着流水直到 MEM 阶段第一级流水后再进行触发。特别地，异常返回指令 ERET 被实现为一种特殊的异常，也保留到 MEM 阶段的第一级流水触发。

需要支持的异常按优先级先后在表4中列出。

表 4: 主要支持的异常

异常简称	异常说明	异常简称	异常说明
Int	中断	Sys	系统调用
Mod	TLB 修改异常	Bp	断点
TLBL	TLB Load 异常	RI	保留指令
TLBS	TLB Store 异常	CpU	协处理器不可用
AdEL	地址 Load 异常	Ov	算术溢出
AdES	地址 Store 异常	Tr	自陷异常

2.6 内存管理

MIPS 为操作系统的内存管理提供了较为简单的支持，虚拟地址通过 MMU 转换为物理地址。MIPS 标准对虚拟地址和物理地址的映射如表5所示。

如果不开启 TLB，那么地址映射都是线性映射，只需要将虚拟地址的高 3 位置为 0 即可满足。如果开启 TLB（默认没有开启 TLB），则具体的地址转换由 TLB 来完成，TLB 可以认为是在 CPU 内部的地址转换表的高速缓存。具体的内容需要由操作系统来进行填充。如果在 TLB 中没有找到对应虚拟地址则会触发一个 TLB miss 异常，操作系统对该异常处理，并且将对应转换表填入 TLB 中的某一项来完成对地址的转换。我们一共实现了 32 个 TLB 项。

表 5: MIPS 规范的虚拟地址空间

段	虚拟地址	权限	物理地址
kseg3/kseg2	0xC000000-0xFFFFFFFF	Kernel	由 TLB 转换
kseg1	0xA0000000-0xBFFFFFFF	Kernel	0x00000000-0x1FFFFFFF
kseg0	0x80000000-0x9FFFFFFF	Kernel	0x00000000-0x1FFFFFFF
useg	0x00000000-0x7FFFFFFF	User	由 TLB 转换

对于初赛，我们设置的属性如表6所示. 在我们提交的作品中，为了提高性能测试分数，没有打开 TLB。

表 6: 本项目初赛采用的地址空间

段	虚拟地址	权限	物理地址与缓存方式
kseg3/kseg2	0xC000000-0xFFFFFFFF	Kernel	Unmapped(0xC000000-0xFFFFFFFF) & Cached
kseg1	0xA0000000-0xBFFFFFFF	Kernel	Mapped(0x00000000-0x1FFFFFFF) & Uncached
kseg0	0x80000000-0x9FFFFFFF	Kernel	Mapped(0x00000000-0x1FFFFFFF) & Cached
useg	0x00000000-0x7FFFFFFF	User	Unmapped(0x00000000-0x7FFFFFFF)& Cached

2.7 外部接口

CPU 整体的结构如图 2 所示，CPU 本身暴露三个 AXI 4 Master 接口，分别进行指令读取、数据存取和不过缓存的数据存取。Cache 控制器与 CPU 之间使用自定义的总线信号进行握手和数据传输，以避免在内部使用 AXI 协议带来的不必要延时；而对外，则将必要的访存转换为符合 AXI 规范的信号，与 SoC 进行交互。

由于初赛 myCPU 目录中的 CPU 顶层模块对外只允许暴露一个 AXI-3 Master 接口，我们使用 Xilinx 的 AXI Crossbar IP 将其转换为三个 AXI-4 Slave 接口以适应 CPU 的需要。为了

不使得指令预取阻塞访存，crossbar 上的仲裁顺序为 Passthrough > Data Cache > Instruction Cache。

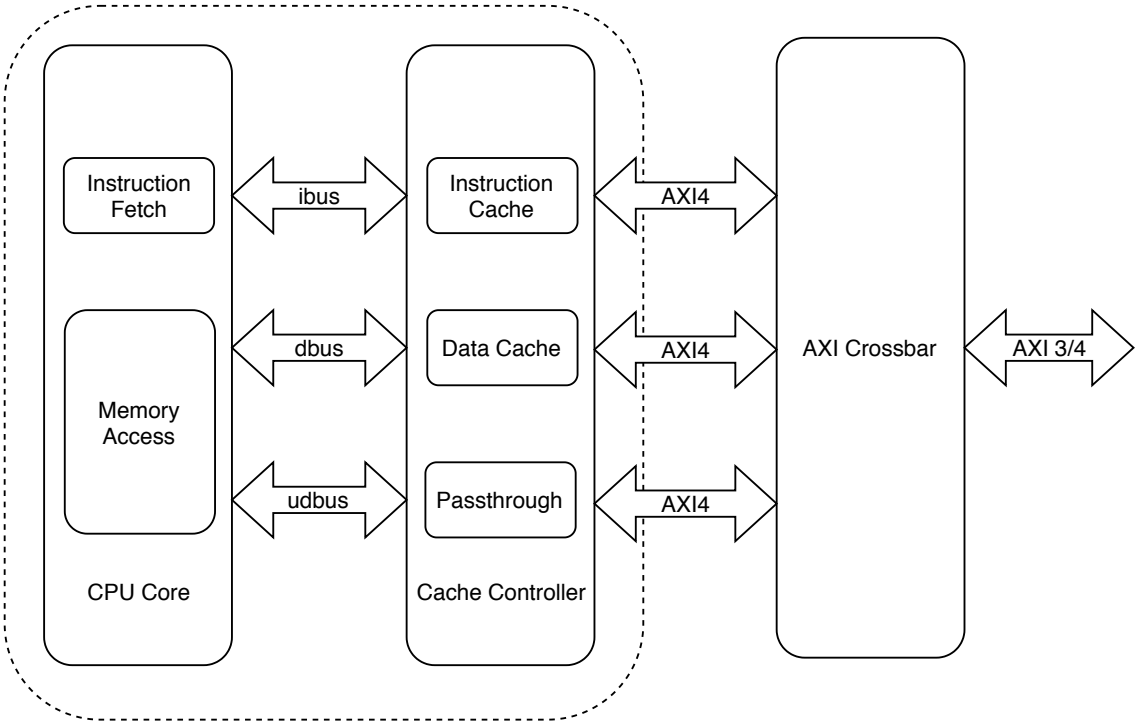

图 2: CPU 整体接口架构

2.8 缓存设计

CPU 提供指令缓存和数据缓存，分别处理取指和访存请求。所有的缓存均为虚拟索引物理标签（VIPT）。截至初赛提交前，我们还没有实现缓存冲刷指令 CACHE。指令和数据缓存都是 4 路 4K，替换策略为 PLRU(Pseudo-LRU)。

2.8.1 指令缓存

指令缓存提供了取指支持，大小和相连度可配置，采用三级流水，其第一个流水段接受请求，并且读取对应的 TAG 和数据。第二个流水段判断是否命中，第三个流水段给出数据。如果不命中，在第三流水段通过 AXI 和外设通信读入。

2.8.2 数据缓存

数据缓存提供了访存支持，包括经过缓存和不经过缓存两个类型的访存。根据 MIPS 标准要求，对于不同段的访存，数据缓存行为不同，具体行为见表6; 数据缓存采用和指令缓存一致的结构，都采用 BRAM(Xilinx Parameterized Macros)，大小和相联度可配置，TAG 和 DATA 缓存分开，便于 CPU 控制访问, 缓存策略采用**写回 + 写分配**。

对于可缓存的读请求，会在 MEM 第一阶段访存 BRAM，在第二阶段得到 TAG 和 DATA，计算命中信息和数据，同时计算可能进行的缓存替换块的地址。在 MEM 第三阶段，如果需要写回脏块，则先进行 DCACHE 总线的 AXI 写事务，之后再开启突发读事务，更新缓存，得到访存结果。

对于可缓存的写请求，会在 MEM 第一阶段访存 BRAM，在第二阶段得到 TAG 和 DATA，计算命中信息和数据，同时计算可能进行的缓存替换块的地址。在 MEM 第三阶段，如果命

中，则直接修改 DATA 缓存，并将 Dirty 位置 1。如果没有命中，则先写回脏块（如果有必要），之后再读取目标地址的缓存行，将要更新的数据和读出的缓存行一起写入 TAG 和 DATA 缓存，并置 Dirty 位为 1.

对于不过缓存的直读直写，直接访存 UNCACHE 总线，通过 AXI 读取或写入 4Byte 即可，不需要更新或读取 Cache.

3 SoC 设计

SoC 设计部分参考 NonTrivialMIPS 的片上系统，我们的 CPU 将暴露 3 个 AXI-Master 接口，接入片上系统并能够成功上板运行。SoC 的架构与 NonTrivialMIPS 相同，如图3所示。其中各个组件之间均使用标准的 AXI4 总线协议进行连接，并使用了多个 AXI Interconnect。图中无阴影的矩形表示 AXI Master，带阴影的矩形表示 AXI 总线互联（Interconnect）或适配器（Bridge），圆角矩形表示外设。图中的所有实线表示其为具体设备（及其连接关系），虚线表示由多个具体设备构成的一类设备（及其连接关系），双线表示到 FPGA 片外的连接。

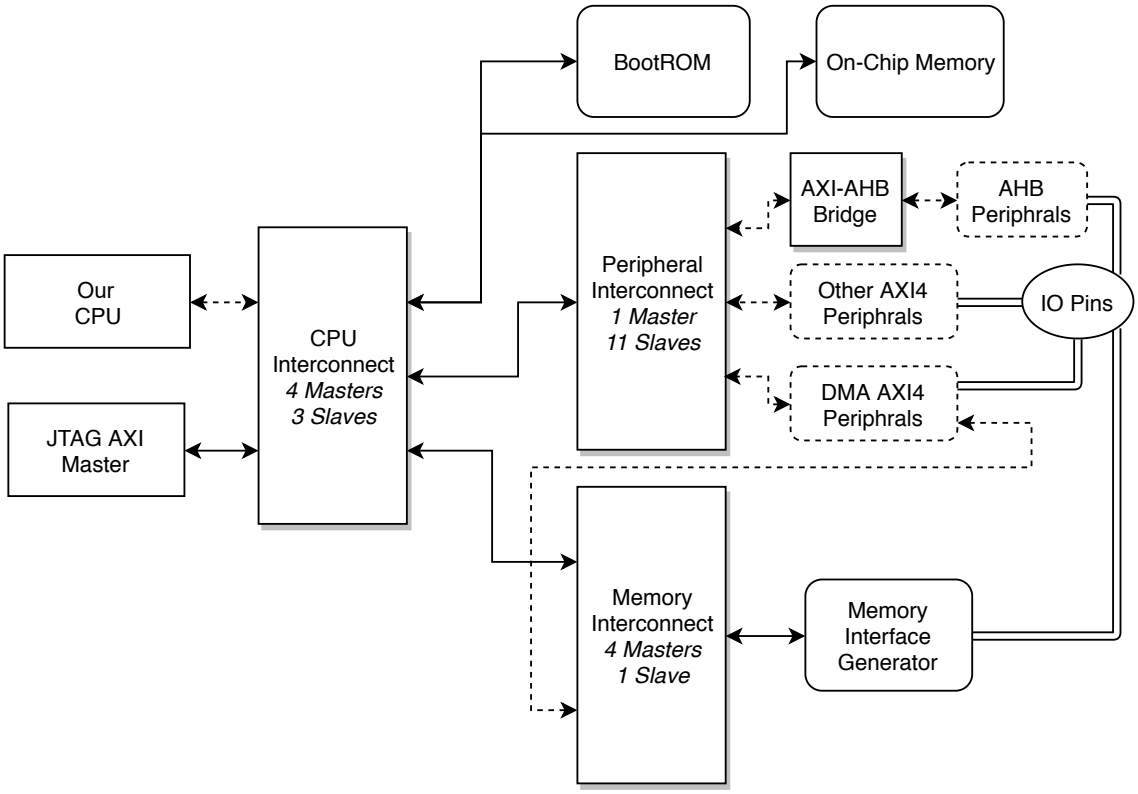

图 3: SoC 架构

3.1 地址分配

表 7 列举了各个设备的物理地址空间分配，其对于总线上的所有 Master 设备都是一致的。但处于复杂度考虑，我们适配的软件并没有用到所有的外设。

表 7: 外设物理地址分配

名称	起始地址	结束地址	有效大小	类型
DDR 控制器	0x00000000	0x07FFFFFF	128 MB	存储
OCM 控制器	0x08000000	0x0800FFFF	64 KB	存储
CFG Flash 控制器（内存映射）	0x1A000000	0x1AFFFFFF	16 MB	存储
以太网控制器	0x1C000000	0x1C00FFFF	64 KB	寄存器
VGA 控制器	0x1C010000	0x1C01FFFF	64 KB	寄存器
PS/2 控制器	0x1C020000	0x1C020FFF	4 KB	寄存器
LCD 控制器	0x1C030000	0x1C030FFF	4 KB	寄存器
SPI Flash 控制器	0x1C040000	0x1C040FFF	4 KB	寄存器
USB 控制器	0x1C050000	0x1C050FFF	4 KB	寄存器
Framebuffer Reader 控制器	0x1C060000	0x1C06FFFF	64 KB	寄存器
Framebuffer Writer 控制器	0x1C070000	0x1C07FFFF	64 KB	寄存器
外部中断控制器	0x1D000000	0x1D00FFFF	64 KB	寄存器
BootROM 控制器	0x1FC00000	0x1FC1FFFF	128 KB	存储
串口控制器	0x1FD02000	0x1FD03FFF	8 KB	寄存器
GPIO 控制器	0x1FF00000	0x1FF0FFFF	64 KB	寄存器

4 系统软件适配

4.1 TrivialBootloader

作为板上系统的一部分，BootROM 中包含了系统每次上电或重置时都会首先执行的代码，起始物理地址为 `0x1FC00000`。由于这部分程序是固化在 FPGA 中的，我们直接采用了 TrivialBootloader 进行第一步引导。

TrivialBootloader 将进行系统的全局初始化、启用异常处理，并进行基本的内存检测。它还支持从 Flash 读取 ELF，将其复制到内存来启动；从串口直接向内存加载数据启动；直接跳转到内存 0x80000000 启动等多种启动方式。

4.2 uCore-thumips

uCore-thumips[1] 是针对简化后的 MIPS 32 实现：MIPS32S 平台的 uCore 移植版本。该项目针对 MIPS32S 平台实现了对应的 Bootloader、初始化流程、异常处理、内存管理和上下文切换流程。相比标准的 MIPS 32，MIPS32S 缺少部分指令且不支持延迟槽。针对这些不同，uCore-thumips 对 uCore 操作系统的编译选项进行了相应的修改，并提供了额外的库函数实现缺失的指令（如 `divu`）的功能。

4.2.1 编译方法

在非 mipsel 平台编译、调试 uCore-thumips 需要使用面向 mipsel 架构的交叉编译、调试工具链，所需工具主要包括 binutils、gcc 和 gdb。

Debian 系统下，`gcc-mipsel-linux-gnu` 和 `binutils-mipsel-linux-gnu` 软件包分别提供了预编译的目标平台为 mipsel 的 binutils 和 gcc。其它操作系统的工具链可参考 LinuxMIPS

[1] https://github.com/z4yx/ucore-thumips

项目文档[2]自行编译。此外 Sourcery CodeBench Lite[3] 提供了预编译的 mipsel 工具链。

交叉编译时，指定 `CROSS_COMPILE` 环境变量或修改 Makefile 中 `CROSS_COMPILE` 变量为所使用的交叉编译器，即可使用 make 进行编译。编译后得到镜像 `ucore-kernel-initrd` 和 `boot/loader.bin` 分别为系统内核 ELF 和 Bootloader。

进行移植时，需针对片上系统对 Makefile 中相应配置进行修改，包括延迟槽、浮点模块等编译选项、为用户 App 预留存储大小等，还要进行设备树的适配。

5 致谢

本项目的开发得到了清华大学计算机系老师和同学们的大力支持，这让我们有底气和信心参加这样一个高难度的竞赛。特别是我们的指导老师刘卫东老师、陈康老师、陆游游老师，他们的倾力指导和宝贵建议让我们敢于攻克一个个难以发现的 bug. 计算机系的陈嘉杰学长、陈晟祺学长、高一川学长、王邈学长，他们作为参赛过来人的宝贵经验让我们少走了很多弯路，在此向他们表示衷心的感谢！

A 本项目所用 IPCore 等开源模块

- Xilinx IP Divider Generator (5.1)
- Xilinx IP AXI Crossbar (2.1)
- Xilinx IP Floating-point (7.1)
- `xpm_memory_sdpram` (Xilinx Parameterized Macros)
- `xpm_memory_tdpram` (Xilinx Parameterized Macros)
- `xpm_memory_dpdistram` (Xilinx Parameterized Macros)

B 参考资料

本项目的设计、开发过程需要参考包括且不限于下面列出的书籍、文献和资料：

- *计算机组成与设计: 硬件/软件接口*. David A.Patterson
- *See MIPS Run Linux.* Dominic Sweetman
- *自己动手写 CPU.* 雷思磊
- *MIPS® Architecture For Programmers I, II, III.* Imagination Technologies LTD.
- *Vivado 使用误区与进阶*. Ally Zhou
- *NonTrivialMIPS 设计文档*. 陈晟祺，周聿浩，刘晓义，陈嘉杰
- *Xilinx IP Core 相关手册*. Xilinx 公司

[2] https://www.linux-mips.org/wiki/Toolchains

[3] https://sourcery.mentor.com/GNUToolchain/release2189