**LALA-Core设计报告**

东北大学1队

包焕垲、吕浩然、李隆轩、邱芃尧

2023年8月

目录

[一、设计简介 3](#_Toc142051232)

[（一）CPU结构 3](#_Toc142051233)

[（二）Cache 3](#_Toc142051234)

[（三）AXI 3](#_Toc142051235)

[二、设计方案 3](#_Toc142051236)

[（一）总体设计思路 3](#_Toc142051237)

[（二）流水线设计 5](#_Toc142051238)

[1. IF1段 5](#_Toc142051239)

[2. IF2段 5](#_Toc142051240)

[3. ID段 5](#_Toc142051241)

[4. EXE段 5](#_Toc142051242)

[5. DT段 5](#_Toc142051243)

[6. MEM1段 5](#_Toc142051244)

[7. MEM2段 5](#_Toc142051245)

[8. WB段 6](#_Toc142051246)

[（三）Cache设计 6](#_Toc142051247)

[（四）AXI设计 6](#_Toc142051248)

[三、设计结果 7](#_Toc142051249)

[（一）设计交付物说明 7](#_Toc142051250)

[（二）设计演示结果 7](#_Toc142051251)

[四、参考设计说明 9](#_Toc142051252)

[五、参考文献 9](#_Toc142051253)

# 一、设计简介

本项目实现了一个基于LoongArch32R指令集设计的CPU，支持基础整数指令、例外与中断等特权指令功能。CPU流水线采用八级流水结构，实现了两路组相联icache和dcache，并且经过封装后可以使用AXI接口与外部设备进行通信。

## （一）CPU结构

为解决在添加icache和dcache后取指令取数据关键路径过长的问题，我们将传统的五级流水线设计进行修改和扩展：将取指令阶段扩展成IF1、IF2两段；将取数据阶段扩展成DT、MEM1、MEM2三段；在我们目前的测试中，在xc7a200tfbg676-2开发板下，成功将频率提升到90MHz。

## （二）Cache

cache采用两路组相联设计。单路cache大小为4KB，两路cache总容量8KB，因此icache和dcache总体共16KB。写分配策略采用LRU算法进行调度。

## （三）AXI

使用状态机解决AXI中icache、dcache以及uncache的访存冲突，同时在读取新数据的同时将旧数据写回，实现读写并行，减少数据交换的时间。

# 二、设计方案

## （一）总体设计思路

我们的设计实现了基础的54条基础指令，支持8种例外中断。流水线总共分为8级，依次为：IF1、IF2、ID、EXE、DT、MEM1、MEM2、WB。通过增加cache访存的周期来切断关键路径。同时在数据前递模块中，对于load和csr类型指令，对其有数据相关的下一条指令进行阻塞，直到WB阶段，进一步减少该路径的时序。

在核内，在EXE段完成分支跳转判断，在MEM2段处理异常处理、状态寄存器的读写。

在核外，采用AXI接口进行封装，配有icache、dcache和uncache。

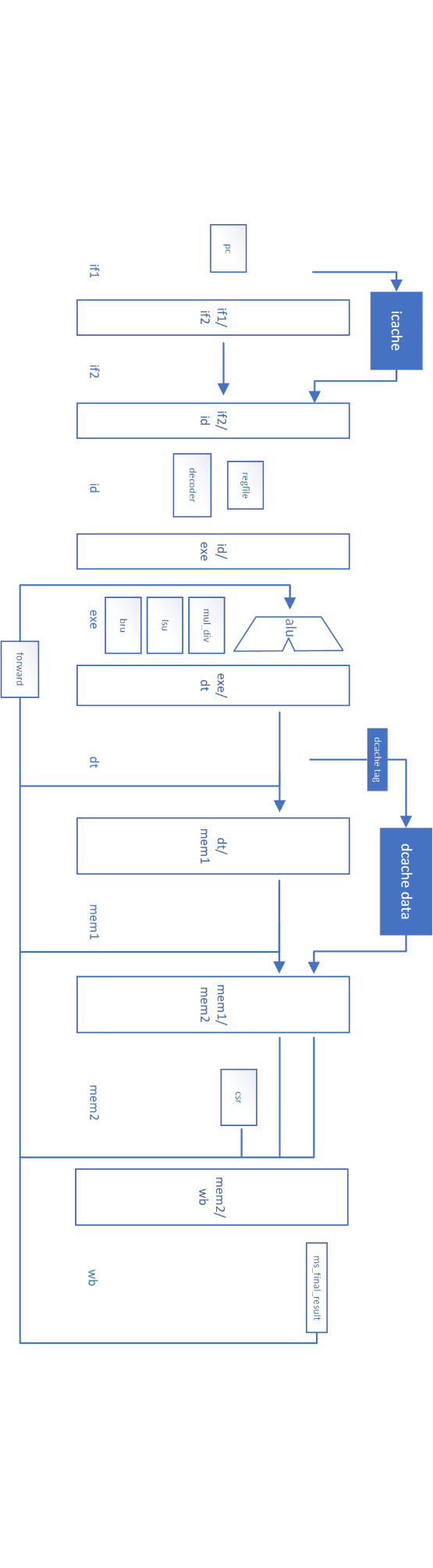


图1八级流水线总体结构

## （二）流水线设计

八级流水线各部分的功能和设计方案如下：

### IF1段

在本流水段中，完成pc的选取，同时向icache发出读请求。若icache未命中，则阻塞暂停整个流水线，等待icache。

### IF2段

在本流水段无额外操作，为icache提供更多周期，减少icache访存的时序需求。

### ID段

在本流水段中，在接收到指令数据后，进入译码器模块，得到控制信号以及后续处理的数据。

### EXE段

在本流水段中，包含了算术逻辑模块alu、乘除法顶层模块mul\_div\_top以及访存模块lsu。其中alu模块中包含了add、sub、slt、sltu，、and、nor、or、xor、sll、srl、sra、lui等12种基础运算。mul\_div\_top模块中包含了1周期的乘法器和32周期的试商除法器，乘除法器运算时阻塞暂停整个流水线。lsu模块用于处理和生成向dcache发送的数据。

### DT段

在本流水段中，主要向dcache发送读请求，减少了dacache tag段组合逻辑的时序需求。若dcache未命中，则阻塞暂停整个流水线，等待dcache。同时本段也为前递提供数据。

### MEM1段

在本流水段中，除去为前递提供数据部分，无额外操作，为dcache提供更多周期，减少dcache访存的时序需求。

### MEM2段

在本流水段中，在接收到访存数据后，进行处理。同时本段中包含了csr模块，对于状态寄存器的读写同寄存器堆写同步读异步的逻辑，对于例外中断信号，若发生例外中断，则在本周期发出flush信号。

### WB段

在本流水段中，除去将数据写回至寄存器堆，无额外操作。

## （三）Cache设计

在我们的设计中cache采用两路组相联设计。单路cache大小为4KB，两路cache总容量8KB，因此icache和dcache总体共16KB。写分配策略采用LRU算法进行调度。其中cache单路设置了128个cacheline，每cacheline内有一个line存储单元，每个单元可存储8条数据。故指ache的单路容量为128\*1\*8\*4Byte，即4KB大小。

当访问命中时，可使cpu核流水地完成读或写请求，无需阻塞主流水线。当未命中时会立即阻塞流水线，使主流水线保持当前请求，并向AXI接口发出读请求。此时，如果预计要写回的cacheline中存在脏数据，则cache会在发送读请求的同时发送写请求，将cacheline中的脏数据写回。当cache模块接到AXI接口发回的新数据并写入后，主流水线的请求会转为命中状态，阻塞结束。

同时对于强序非缓存，通过uncache模块进行读写。当主流水线发来访存请求时，默认为访存缺失，阻塞流水线，并将请求转发给AXI接口进行访存。

## （四）AXI设计

在设计过程中，我们将CPU封装为AXI接口，用于与系统环境和外设的交互。

其中，dcache的请求和uncached类型的访存请求是互斥关系，因此不会同时出现。uncached类型的读请求和写请求也是互斥关系，不会同时出现。如果同时出现指令读和数据读，则先完成指令读，然后完成数据读。写行为同时只会出现一种，且可以在读行为的过程中同时进行。写操作或者读操作完成后，需等待另一种行为完成才能再次回到检测状态，以保证二者的一致性。

表1 访存请求处理顺序

|  |  |
| --- | --- |
| 优先级 | 类型 |
| 1 | icache读请求 |
| 2 | dcache读请求和写请求 |
| 3 | uncached读请求/写请求 |

# 三、设计结果

## （一）设计交付物说明

|  |  |
| --- | --- |
| **|-score.xlsx** | Excel表格，包含功能测试、性能测试得分的计算 |
| **|-design.pdf** | PDF文件，为myCPU设计报告，本文档 |
| **|--sram\_src/** | **（无内容）** |
| **|--src/** | 目录，AXI目录 |
| **| |--mycpu/** | 目录，存放AXI相关工程的源码文件 |
| **| | |--ip/** | 目录，存放本设计调用的Xilinx IP |
| **| |--perf\_clk\_pll.xci** | 性能测试时钟分频Xilinx IP文件 |
| **|--bit/** | 目录，存放预赛作品最后的bit流文件 |
| **| |--func\_test/** | 目录，存放功能测试bit流文件 |
| **| | |--soc\_axi\_func/** | 目录，存放AXI功能测试bit流文件 |
| **| | |--soc\_sram\_func/** | **（无内容）** |
| **| |--perf\_test/** | 目录，存放性能测试bit流文件 |

## （二）设计演示结果

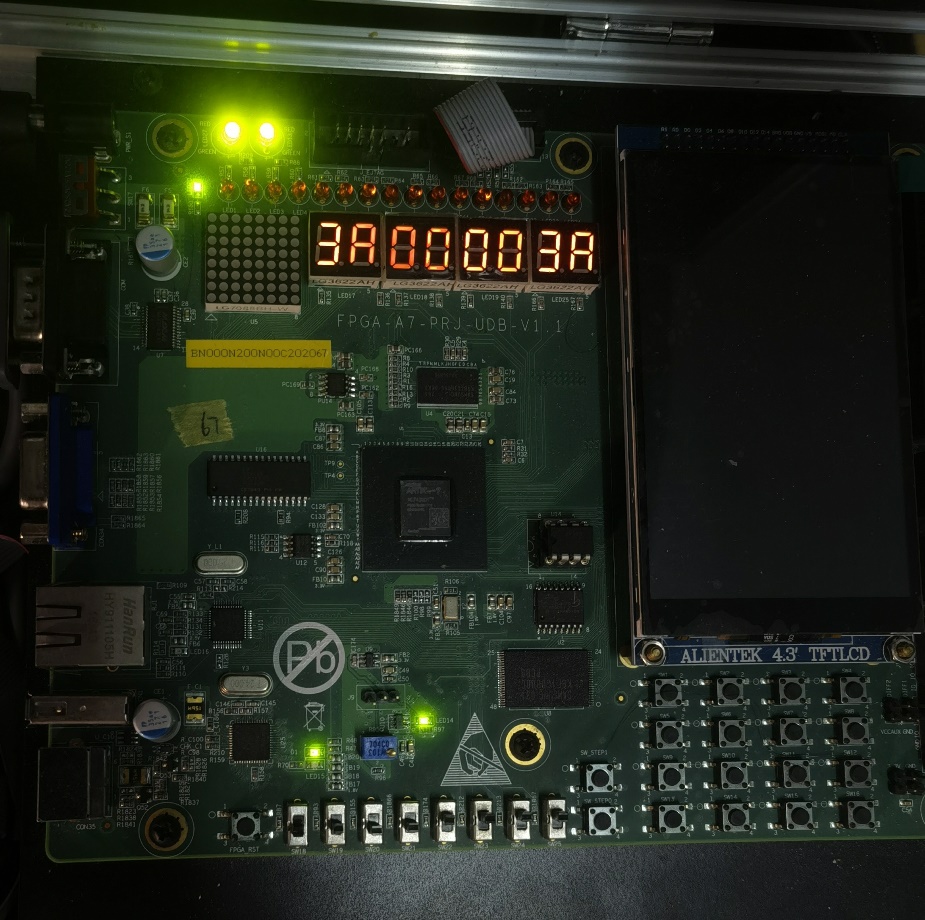


图2 功能测试上板

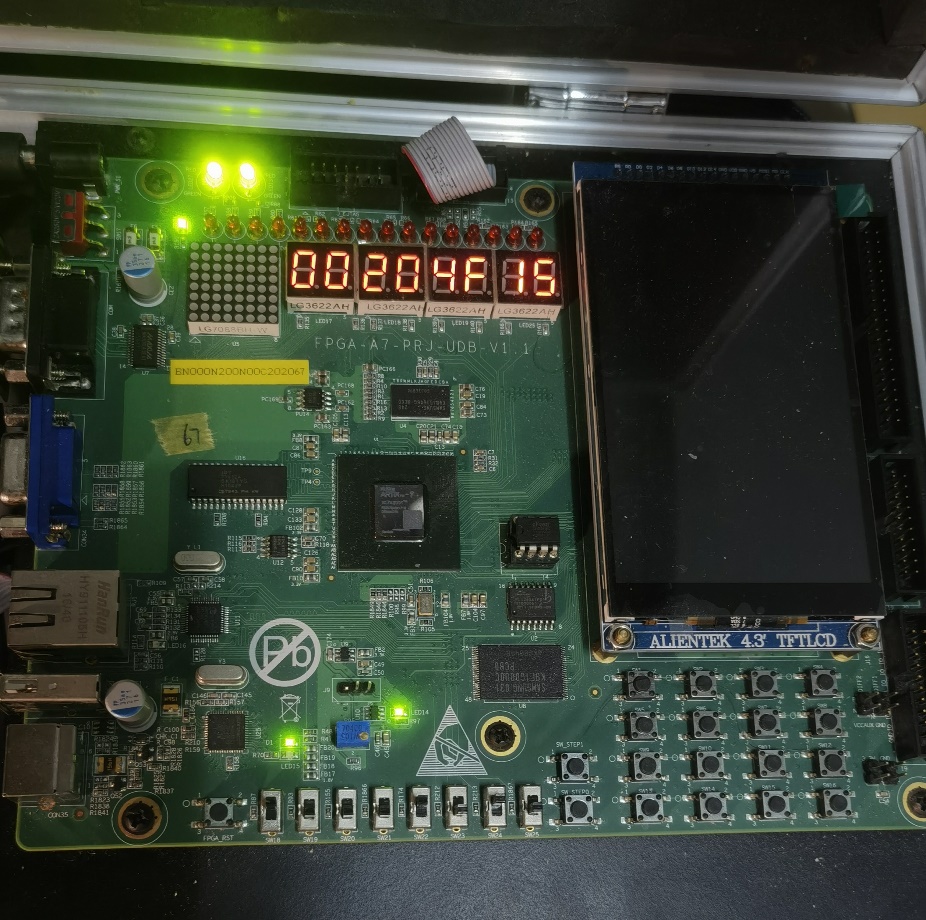


图3 性能测试上板

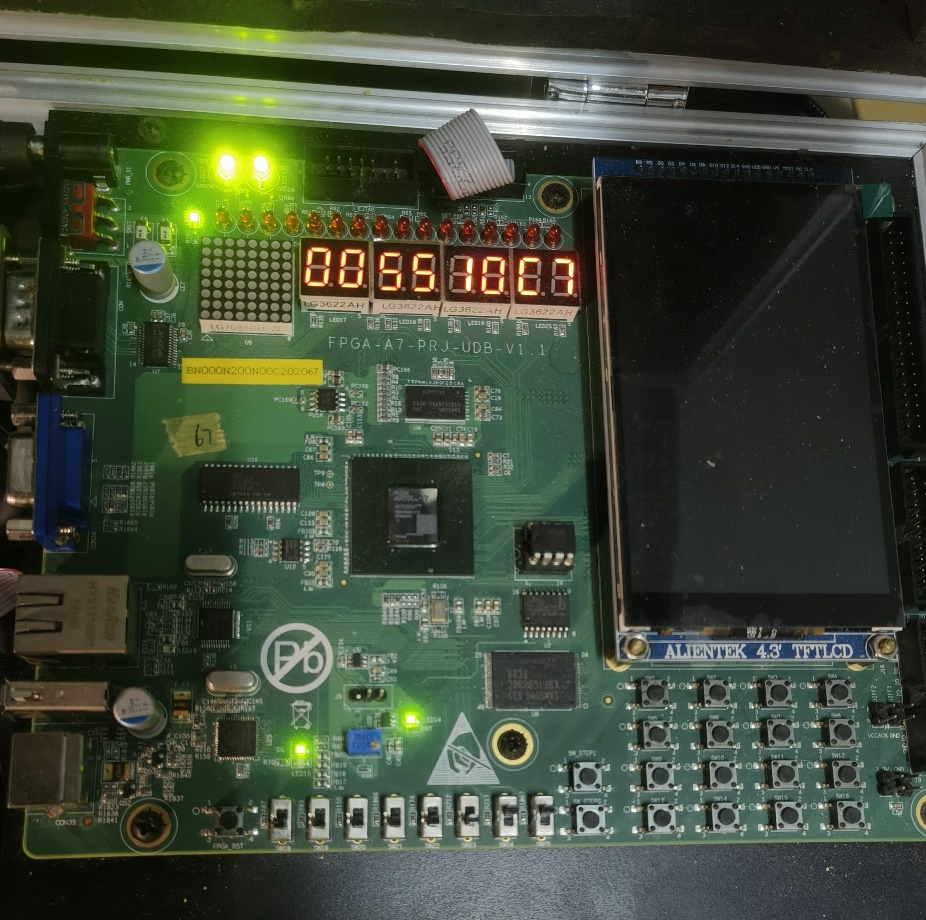


图4 性能测试上板

# 四、参考设计说明

我们的项目最初是参考了《CPU设计实战》文献[1]的传统经典五级流水线的结构进行设计。同时在译码器部分、例外中断、控制寄存器部分参考了chiplab中demo的部分设计文献[2]。在这些基础上进行了完善和优化。

在核外cache和axi的设计中，我们参考了我校往届的参赛作品文献[3]-[5]，并针对性的对我们的设计进行了修改和优化，以完成本次比赛的功能测试和性能测试。

# 五、参考文献

[1] 汪文祥，邢金璋. 《CPU设计实战》[M]. 北京：机械工业出版社. 2021.

[2] LoongsonEdu. chiplab[OL]. [2023.8]. https://gitee.com/loongson-edu/chiplab.

[3] polarispw. NE-Core[OL]. [2023.8]. https://github.com/polarispw/NE-Core.

[4] fluctlight001. April\_CPU[OL]. [2023.8]. https://github.com/fluctlight001/April\_CPU.

[5] fluctlight001. cpu\_for\_nscscc2022\_single[OL]. [2023.8]. https://github.com/fluctlight001/cpu\_for\_nscscc2022\_single.