经典五级流水线 CPU 设计报告

北京理工大学 1 队
甄淇深、李航、贺隽文、吴杰

一、设计简介

此 CPU 设计采用经典五级流水线架构。能够在 100MHz 的主频下运行通过功能测试和 125MHz 的主频下运行性能测试。功能测试能够通过 89 个功能点和记忆小游戏。性能测试能够通过 10 个性能测试程序，性能测试得分约为 8.8。

二、设计方案

（一）总体设计思路

系统采用经典的五级流水架构，划分为取指、译码、执行、访存和回写五个阶段，级间采用锁存器存储数据，每个时钟周期依次向下传递数据。采用数据前推和流水线暂停方法解决数据相关和控制相关问题，其中流水线暂停使用统一控制机制实现。每个模块的详细设计情况如下所示。

1. 取指阶段

取指阶段以类 SRAM 接口提供指令地址，通过大赛提供的类 SRAM 转 AXI 发出取指请求，暂停流水线直至指令取回并在下一个时钟周期传递给译码阶段。除 cpu_axi_interface.v 中要求提供的类 SRAM 接口外，为适配 AXI 接口，增加 instr_cache 信号传递至 cpu_axi_interface.v。

2. 译码阶段

译码阶段主要包括控制信号使能单元(control_unit)、寄存器堆(regfile)以及转移指令相关处理。

控制信号使能单元通过分析指令类型，设定相应的控制信号的取值传递至后续阶段，决定了每条指令对应的操作。图一给出了重要的控制信号含义说明。

寄存器单元采用异步读取和同步写入，通过译码得到需读取的 Regfile 中寄存器的地址，并在同一周期内得到寄存器的值；对于写寄存器，由控制信号 RegWrite 决定是否写入，该信号由译码阶段产生后一级级传递至回写阶段，在时钟上升沿写入寄存器堆。

为避免转移指令带来的控制冒险问题，将转移指令的执行提前到译码阶段，在此阶段决

定是否发生跳转以及跳转目标地址，并提供给取指阶段作为新的指令地址。由于此时取指阶段为延迟槽指令，因此若发生跳转，则跳转目标地址会在下一个时钟周期被取指阶段选中，此时延迟槽指令已经正常向后流动，保证了延迟槽指令一定被执行。

3. 执行阶段

执行阶段通过译码阶段传递的控制信号决定相应的操作，对操作数进行计算，若操作数还未写回到寄存器，则使用数据前推选择正确的操作数。比较特殊的是乘除法运算，由于verilog 的乘法符号会默认调用 IP 核，对 CPU 的主频有一定影响，因此在实现上参考了已有的开源乘法器和除法器，每个 32 位的乘法和除法通过暂停流水线 32 个周期来完成，当运算完成后解除暂停，在下一个时钟周期将结果传递至访存阶段。

4. 访存阶段

访存阶段包括 HILO 模块以及 CP0 和异常处理模块，同时通过类 SRAM 接口发出访存信号，为适配 AXI 接口，增加 data_cache 信号决定是否使用系统 CACHE。和取指类似，访存阶段读/写数据时，都需要暂停流水线直至得到数据读取成功或写入成功的信号。

HILO 模块通过控制信号决定是否读/写 HILO 寄存器，数据由执行阶段的计算结果提供；CP0 和异常处理模块在下文单独陈述。

为解决数据相关问题，通过控制信号选择 HILO、CP0 以及数据 RAM 可能读出的数据，前推到执行阶段，并通过冲突处理单元产生的控制信号进行选择。

5. 回写阶段

回写阶段负责将计算结果通过同步方式写回到寄存器堆，由于数据在写回前依旧需要解决数据相关问题，因此计算结果也会作为执行阶段的数据来源。

（二）冲突处理 Hazard_unit 模块设计

针对流水线可能出现的数据冲突与控制冲突，将其综合考虑后设计了阻塞+清空+重定向的冲突处理方式。在出现寄存器依赖（后续指令需要使用前面指令写寄存器值）时，依赖方（较后的读寄存器指令）只可能出现于译码与执行两个阶段，分别用于进行分支判断及算术运算；而被依赖方（较前的写寄存器指令），包括运算或移位类指令、访存、跳转（写 pc 相关数据入寄存器）、cp0、hilo 指令，其可能处于执行、访存、回写等阶段并出现多条指令同时被依赖的情况。

在阻塞方式上，为了数据避免从访存到执行再到译码此类连续重定向而时序过长，以及多个指令同时被依赖时丢失数据的情况，我们设计了双优先级的综合阻塞方式。其中以取值、

取数的等待暂停为高优先级，而译码阶段指令依赖于尚无法得到结果的执行阶段指令（如访存、cp0、hilo 等）的阻塞为低优先级。

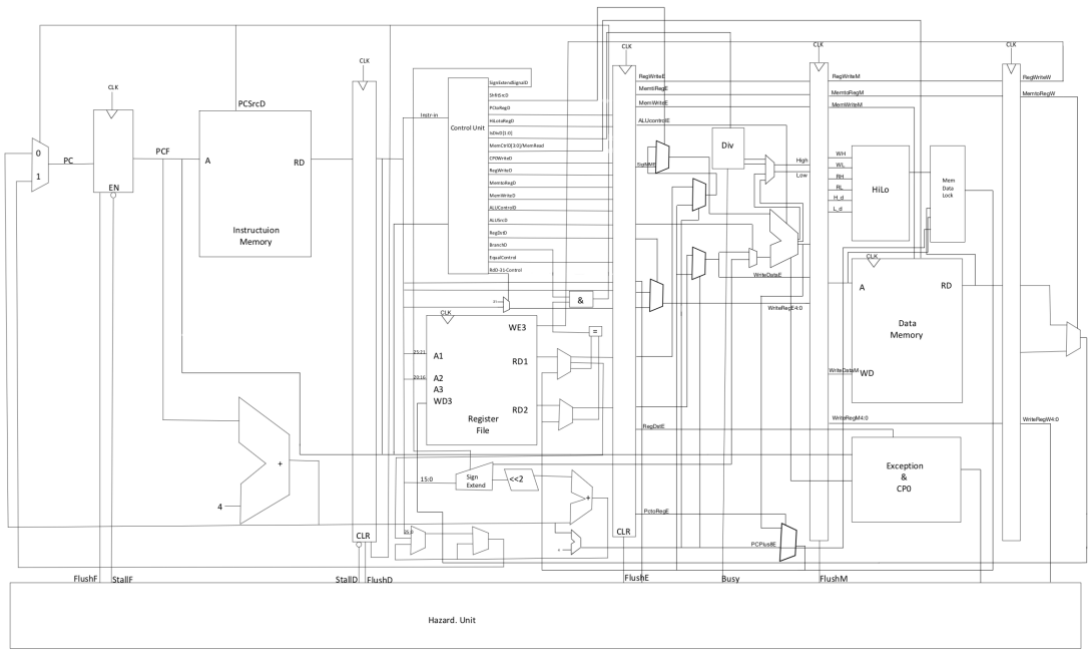
在清空方式上，当发生低优先级阻塞且无高优先级阻塞时清空执行阶段指令，避免持续出现低优先级阻塞而使流水线进入循环，除此之外，异常类指令也需要清空流水线各阶段指令。

在重定向方式上，通过选择器将访存阶段的数据综合为单数据线重定向到执行阶段与译码阶段，将回写阶段的数据重定向到执行阶段（到译码阶段的重定向通过 regfile 直接在写寄存器的同时输出正确数据）。而冲突处理模块输出控制信号选择执行与译码阶段的依赖模块是否接受重定向的数据。

（三）cp0 & 异常处理模块设计

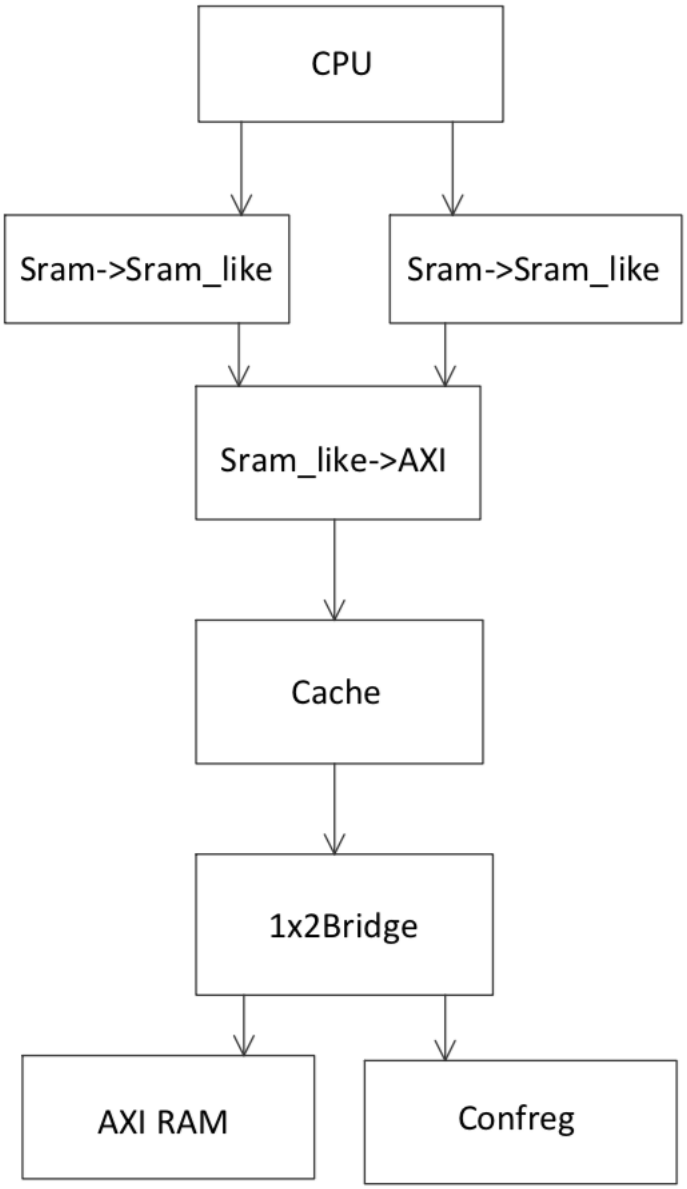
cp0 设计上实现了大赛要求的 6 个 cp0 寄存器，为了方便异常处理和 cp0 之间的数据传输问题，将异常处理放在 cp0 模块中，本 cpu 实现了精确异常，发生异常时，并不立即处理异常。而是通过 ExceptType 信号一路将取指、译码、执行阶段发生的异常传递到访存阶段进行统一处理。异常处理完毕之后，向 hazard 模块给出 flush 信号进行清空流水线，同时向取指阶段给出异常处理例程的入口地址。

（四）本 CPU 数据控制流图



图一 CPU 数据控制流图

（五）SOC 框架图



图二 SOC 框架图

三、设计结果

（一）设计交付物说明

1. 提交设计说明

提交设计的目录层次与比赛框架要求相同，以下对 myCPU 目录做一定说明

- myCPU/	
| - ip/	
| | -sys_cache/	
| | | system_cache_0.xci	Xilinx System Cache IP
| mycpu_top.v	CPU 顶层模块，cache 与信号转换
| mips.v	CPU 内核
| cpu_axi_interface.v	类 sram 转 AXI 接口（龙芯开源代码）
| define.vh	宏定义文件
| get_pc.v	选择 pc 来源模块
| PC_ALU.v	pc+4 模块
| get_instr.v	取值模块
| instr_decode.v	取值到译码流水锁存器模块
| decode.v	指令分解模块
| control_unit.v	控制单元模块
| getRdD.v	判断是否为 31 号寄存器
| regfile.v	寄存器堆模块
| getEqualD.v	跳转寄存器条件判断模块
| get_PCSrcD.v	跳转信号综合
| sig_extend.v	数据扩展模块
| shift_unit.v	位移选择模块
| get_PCBranchD.v	跳转地址生成模块
| get_J_PCBranchD.v	跳转地址生成模块
| get_PCBranchD_final.v	跳转地址生成模块
| get_PCPlus8.v	pc 写入寄存器值生成模块
| decode_exe.v	译码到执行流水锁存器模块
| get_WriteRegE.v	选择写寄存器号模块
| get_SrcAE.v	选择运算数模块
| get_SrcBE.v	选择运算数模块
| get_ShiftSrc.v	选择移位指令移位数模块

	get_WriteDataE.v	选择写存储器数据模块
	SrcAE_SrcBE_ALU.v	加减逻辑及移位运算模块
	mul.v	多周期乘法运算模块
	div.v	多周期除法运算模块
	exe_accessMem.v	执行到访存流水锁存器模块
	access_mem.v	访数据存储器模块
	hilo.v	hilo 寄存器模块
	M_E_databack.v	访存阶段重定向数据综合模块
	cp0_reg.v	cp0 & 异常处理模块
	accessMem_writeback.v	访存到回写流水锁存器模块
	get_ResultW.v	回写数据选择模块
	hazard_unit.v	冲突处理模块

2.、仿真、综合、上板演示的操作步骤

打开各 run_vivado 目录下由 Vivado2018.3 创建的 Vivado 工程 mycpu.xpr，即可直接打开并进行仿真、综合实现并生成比特流下板。

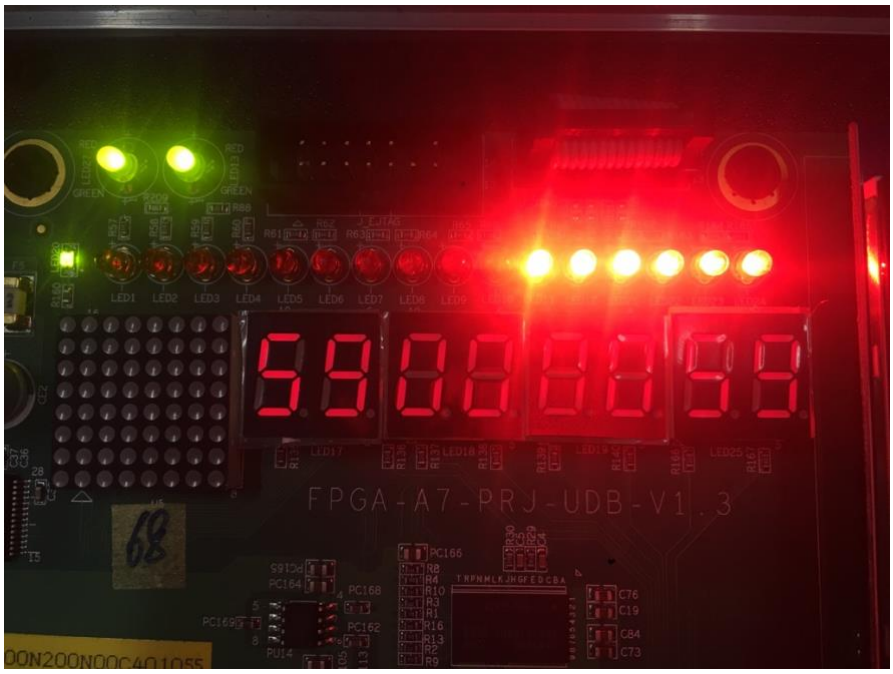
3、说明

功能测试与性能测试的源码只有一行代码是不一样，其他全部相同，在功能测试下 get_instr.v 文件中多出来一行特判代码，而在性能测试下把这行注释掉了。

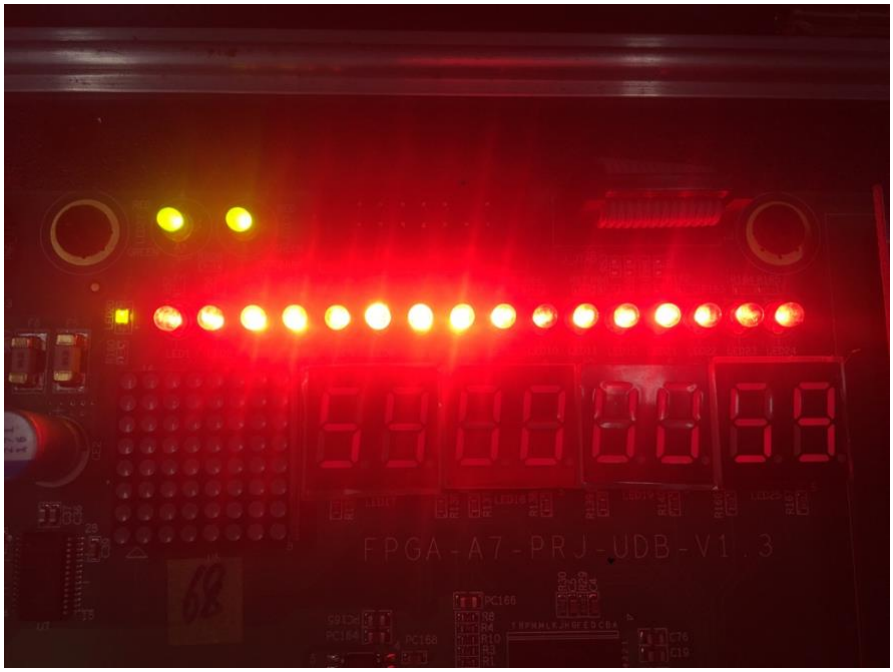
这个 bug 困扰了我们很多天，我们在进行板上 ila 调试的时候，我们 cpu 给出一个正确的 pc，一路一路追查下去，最后返回给我们的指令却是一个错误的指令，而且整个 inst_.coe 文件中也只有这一处有这个问题出现，所以我们在这里做了一个特判处理。而在性能测试中并没有这个问题出现。(以上均基于仿真全通过的情况下)所以我们只好做了这个处理（感觉是外部代码出了问题）。

（二）设计演示结果

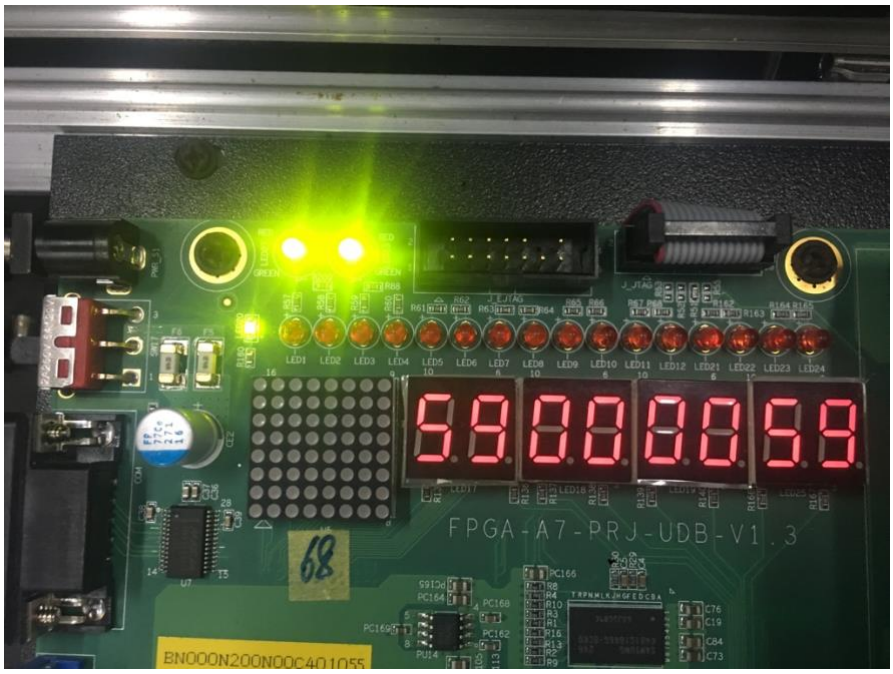
1、功能测试



图三 无延迟

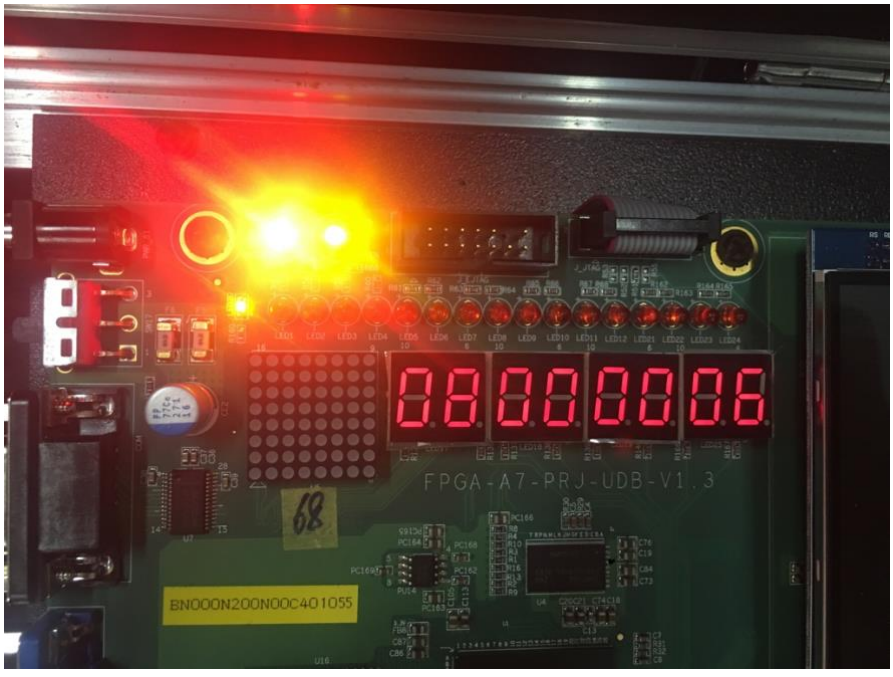


图四 短延迟

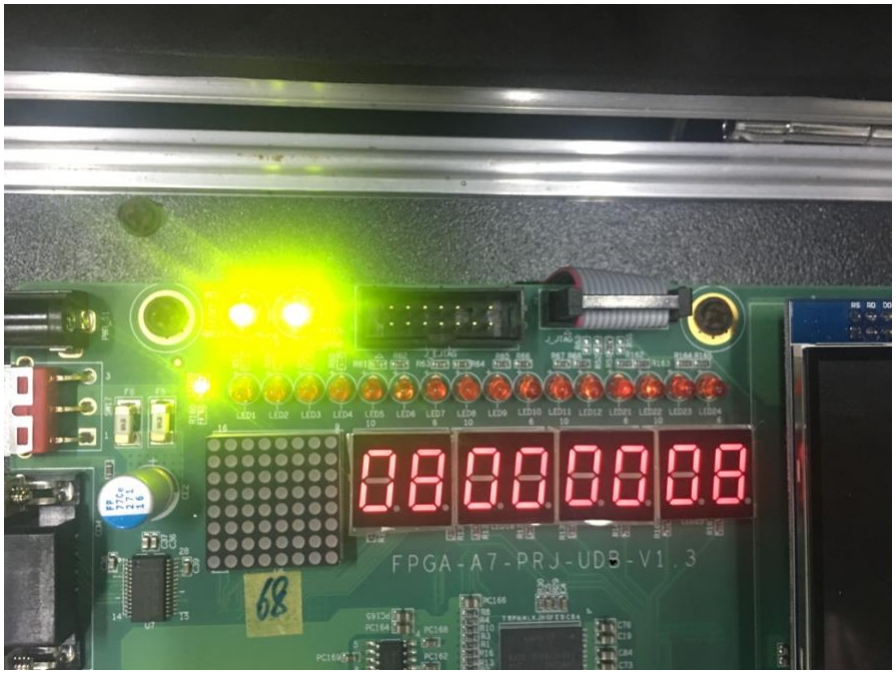


图五 长延时

2、记忆测试小游戏

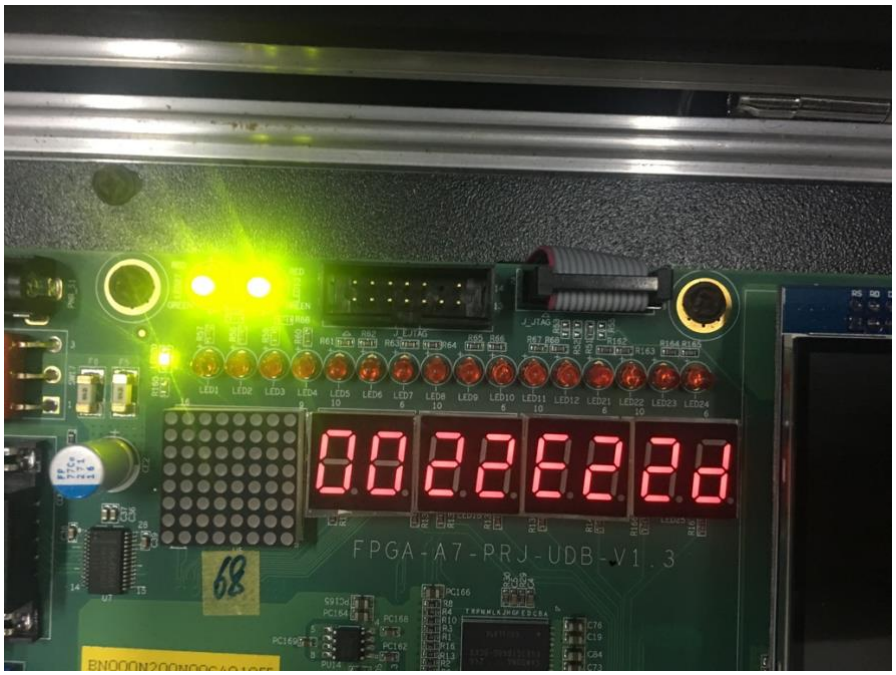


图六 记忆结果错误



图七　记忆结果正确

3、性能测试



图八 性能测试 bitcount

四、参考设计说明

流水线数据通路参考数字设计和计算机体系结构[1]。

除法器模块的设计参考自己动手写 CPU[2]。

类 SRAM 转 AXI 的模块参考龙芯开源代码。

Cache 模块采用 Xilinx IP 核 AXI System Cache。

五、参考文献

[1] 戴维·莫尼·哈里斯. 数字设计与计算机体系结构[M]. 北京:机械工业出版社,2016.4.

[2] 雷思磊. 自己动手写 CPU[M]. 北京:电子工业出版社,2014.9.

[3] Imagination Technologies LTD.. MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set Manual[M]. 2016.5.

[4] Imagination Technologies LTD.. MIPS® Architecture For Programmers Vol. III: MIPS32® / microMIPS32™ Privileged Resource Architecture[M]. 2015.7.