NSCSCC2021 个人赛 CPU 设计报告

学校：青岛理工大学
姓名：姜海天

一、设计简介

实现的 CPU 为经典五级流水线 CPU[1]，能够完整运行监控程序，功能测试和性能测试均为 100 分。对于该 CPU，由于 SRAM 采用状态机取指和读写数据[2]，因此性能较低，两个周期才能取指一次，五级流水线实际只有 2，3 条工作指令，为了提高性能，为 CPU 设计了两套运行模式，分别为 Normal Mode 和 Fast Mode，一般情况下处于 Normal Mode，只有在监控程序执行 G 命令的时候，CPU 会进入 Fast Mode，执行完成之后返回 Normal Mode，Fast Mode 下的性能是 Normal Mode 的 1.735 倍。

Fast Mode 下能够一个周期取到一条指令，该模式下是从 Block RAM 中取指，CPU 会在接收到 G 命令和地址之后将相应程序存入 Block RAM 再执行测试程序。该方式的设计思路来源于 x86 的实模式与保护模式的切换，Block RAM 的功能则类似于 Cache。

二、设计方案

（一）总体设计思路

整个系统主要分为 CPU 核、SRAM 控制器、串口控制器、串口收发器、Block RAM 和仲裁模块几大部分。

SRAM 采用状态机控制，在 Normal Mode 下，CPU 取指需要 2 个周期，取数据需要 3 个周期，写数据需要 1 个周期，因为 CPU 写完后让状态机自己工作即可。读写数据可以从 Ext RAM 也可以从 Base RAM，在读写 Base RAM 时候，CPU 取指需要暂停。

在 Fast Mode 下 CPU 从 Block RAM 取指，每 1 个周期取到 1 条指令，此时对两组 SRAM 的读写则同等对待，Base RAM 不再需要考虑取指与读写数据的冲突。

仲裁模块负责两大部分，一是读写数据的仲裁，读写数据有 Base RAM、Ext RAM 和 Serial Buffer 三个方向，需要根据相关条件判断；二是指令的仲裁，读入的指令可能来自 Base RAM 或 Block RAM，需要相关判断逻辑。

下图是整体模块设计，并未连线，_i 代表输入端口，_o 代表输出端口，对应的两组端口大多名称完全相同。

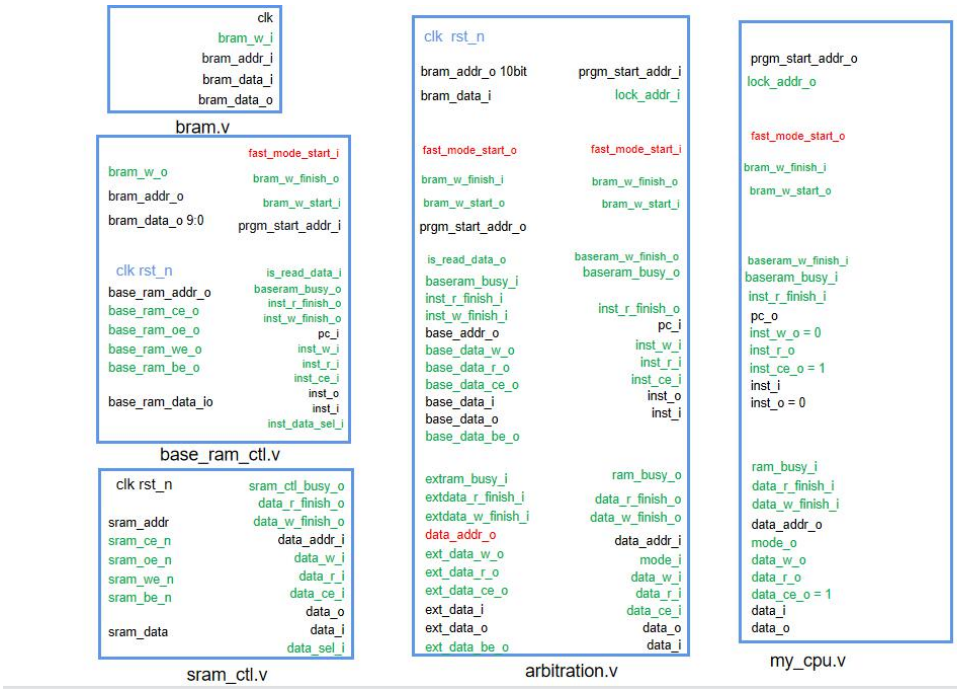

图 1. 整体模块设计图（1）

下图则是串口相关内容的模块连接图：

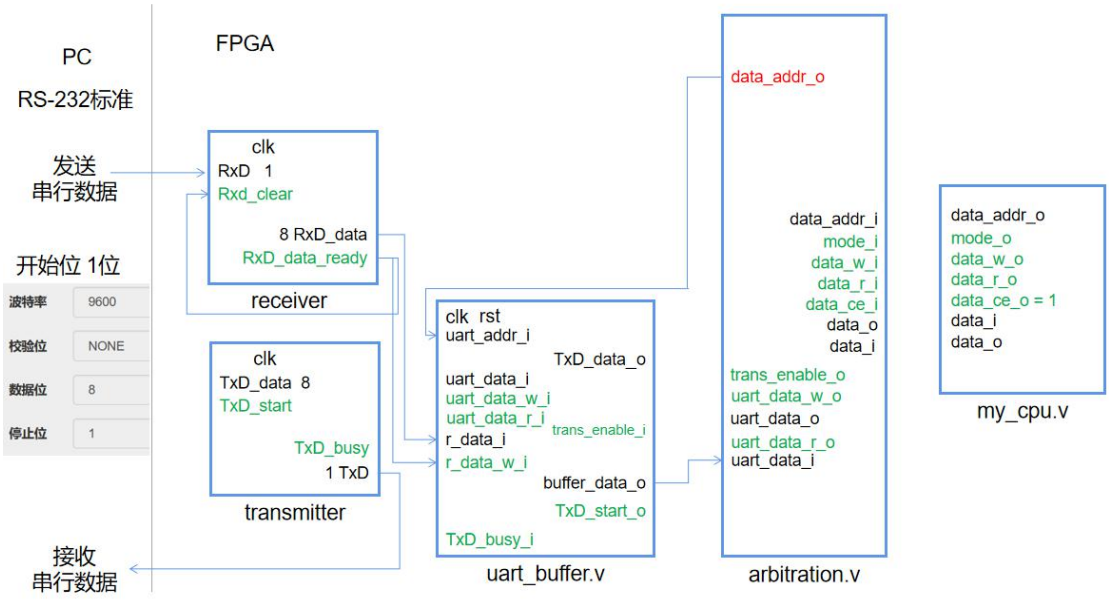

图 2. 整体模块设计图（2）

对于 CPU 的设计，采用的是经典五级流水线结果，大多模块都是常见内容不再赘述，后面会重点说明关于 Fast Mode 的相关模块。

（二）Fast Mode 转换模块设计

该模块充分体现了软硬件交互机制的相互配合，通过 kernel 源码来设计 CPU 硬件结构，达到性能优化的目的，好处是性能能够得到很大提升，缺点是 Fast Mode 仅适用于 kernel，专用性强。

性能测试流程是这样，当 term 执行 G 命令并且输入地址，将其发送给 kernel 之后，kernel 会给 term 发送 0x06 使得计时开始，kernel 执行程序结束后，会发送 0x07 使得计时结束，从而计算得到程序运行时间。

而 Fast Mode 的设计，则是根据来源于该流程。当获取到 G 命令和地址之后，CPU 暂停，将对应程序写入到 Block RAM，进入 Fast Mode，然后给 term 发送 0x06，之后执行程序，结束后发送 0x07，获得运行时间，之后返回 Normal Mode。

当然，这种方式的本质，其实就是一个“伪 cache”而已，主要意图是使得取指时间缩短到原来的一半，性能能够提高 60%～80%，如果是对于通用 CPU，这不是好的设计，实现 cache 是更佳的选择，鉴于 SRAM 不支持突发传输以及备赛时间有限，故实现了与之类似的 Fast Mode 功能来加速 CPU 运行。

执行 Fast Mode 之后，其他模块的逻辑也会随着这个“开关”的打开而变化，以便于能够让该状态下的 CPU 正常运行。具体每个模块的描述信息会在后面以表格形式呈现。

三、设计结果

（一）设计交付物说明

```
├── constrs_1
│   └── imports
│       └── new          // 约束文件
├── sim_1
│   ├── imports
│   └── new              // 仿真文件
│       └── include
└── sources_1
    ├── imports          // 整个硬件系统源码，并未在该工程直接开发
    │   └── new          // 由其他工程直接导入
    │       ├── EX       // EX 阶段模块
    │       ├── ID       // ID 阶段模块
    │       ├── IF       // IF 阶段模块
    │       ├── MEM      // MEM 阶段模块
```

```
│       └─── UART    // 串口 buffer 模块
│       └─── other files // 没有在各阶段目录的
│                        // 包含顶层模块、仲裁模块、SRAM 控制器
└─── ip
    ├─── blk_mem_gen_0 // Block RAM
    ├─── mult_gen_0     // 0 周期乘法器
    └─── pll_example     // pll
```

下面对所有模块及其文件进行一一说明：

表 1. 模块功能说明表

划分		文件名	功能
CPU 核	IF 阶段	fast_start_transition.v	模式切换过渡
		inst_i_ctl.v	处理即将写入到 IF/ID 的指令
		pc_stall.v	取指暂停逻辑
		pc.v	PC 赋值
	ID 阶段	central_ctl.v	非跳转指令控制器
		reg_files.v	寄存器堆
		rW_select.v	写入目标选择
		imm_extension.v	立即数扩展处理
		data_hazard_jump.v	跳转冒险的数据旁路
		stall_pipeline.v	load 和数据、跳转冒险导致暂停流水线
	EX 阶段	ALU_ctl.v	ALU 控制器
		ALU.v	ALU
		data_hazard_ex.v	数据冒险的数据旁路
	MEM 阶段	data_hazard_lwsw.v	load 和 store 指令导致的数据冒险
		loaddata_stall.v	load 指令访存导致的暂停
		r_w_instram_stall.v	读写 Base RAM 导致的暂停
		sw_stall.v	连续 store 导致的暂停
		mode_convert.v	Fast Mode 和 Normal Mode 转换
		IF_ID / ID_EX/ EX_MEM/MEM_WB	流水线寄存器

		my_cpu.v	CPU 顶层模块
		arbitration.v	仲裁模块
		base_ram_ctl.v	Base RAM 控制器
		sram_ctl.v	Ext RAM 控制器
		define.v	全局宏定义

仿真和综合实现均能通过，以下是上板演示，使用 term 进行测试：

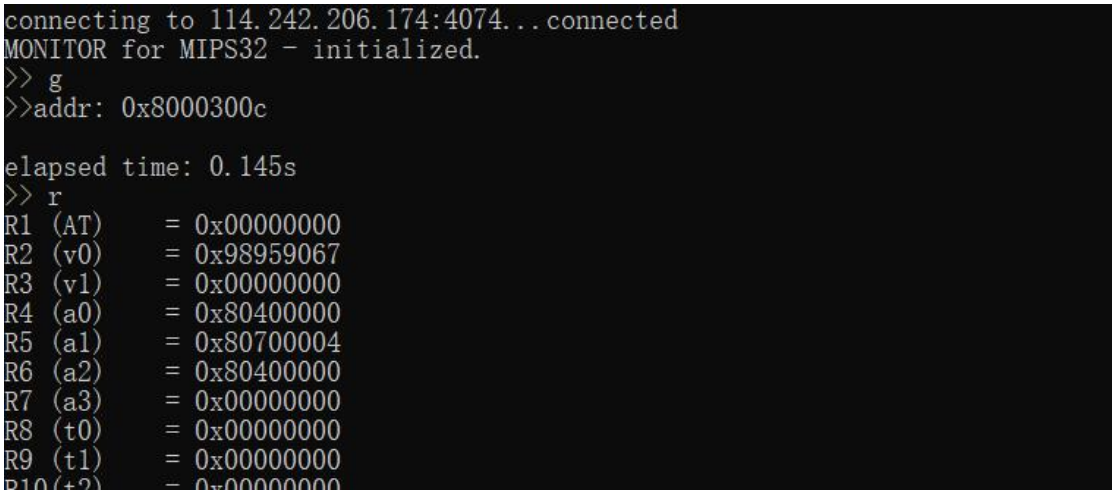

图 3. 上板运行测试

以下为综合、实现的结果：

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
synth_1 (active)	**constrs_1**	**synth_design Complete!**								**1650**	**889**	**0.0**	**0**	**0**
impl_1	constrs_1	write_bitstream Complete!	2.155	0.000	0.131	0.000	0.000	0.288	0	1599	889	1.0	0	3
Out-of-Context Module Runs														
mult_gen_0_synth_1	mult_gen_0	synth_design Complete!												
blk_mem_gen_0_synth_1	blk_mem_gen_0	synth_design Complete!								0	0	1.0	0	0
pll_example		Using cached IP results												

图 4. 综合实现结果

（二）设计演示结果

以下是在测试平台获得的性能测试结果：

图 5. STREAM 测试结果

图 6. MATRIX 测试结果

图 7. CRYPTONIGHT 测试结果

以下是当 CPU 接入 51.5MHz 时钟时候的测量数据：

表 2. 性能测试结果

测试程序	运行时间（s）	实际运行平均频率（MHz）
UTEST_STREAM	0.137	-
UTEST_MATRIX	0.260	-
UTEST_CRYPTONIGHT	0.560	-
UTEST_1PTB	9.123	42.09
UTEST_2DCT	9.126	21.03
UTEST_3CCT	9.125	24.54
UTEST_4MDCT	9.138	24.51

当前整个硬件系统 SRAM 是频率提升的关键点，目前最高支持 51.5MHz，而单就 CPU 而言最高频率 60MHz，由于目前没有总线设计，CPU 与 SRAM 频率一致，所以受到了限制。

四、反思与展望

CPU 的 Fast Mode 是在性能优化环节增加的，在原有的 CPU 基础上，增加了“开关”使得 CPU 能够在 Normal Mode 和 Fast Mode 切换，但是，由于最开始的顶层设计并未考虑两种运行模式，所以对于 Fast Mode 的启动与返回的设计比较冗杂，并且目前 CPU 还存在已知但未解决的 bug，连续执行 G 命令会导致后面的程序计时不准确，这可能由于 Normal Mode 没有完全恢复到初始态导致，鉴于初赛时间限制未能完全修复，初赛提交后将继续进行相关修复工作。

此外，两种运行模式的设计属于基于 kernel 软件的硬件设计，完全依赖于 kernel 程序，专用性强通用性差，属于创新设计，后期还是期待将其改造成 cache 以提高性能。对于 SRAM 超频以及时序优化的探索还在路上，希望未来能够进一步研究。

五、参考设计说明

出了官方给出模板之外的内容外，使用的 IP 核有：Block Memory Generator 和 Multipler；对于信号的定义形式（使用_o 代表输出，_i 代表输入）则是参考了雷思磊的《自己动手写 CPU》[3]；SRAM 的状态机则是参考了《A Practical Introduction to SRAM Memories Using an FPGA》。

六、参考文献

[1] David A. Patterson, John L. Hennessy. Computer Organization and Design[M]. Elsevier: David A. Patterson, John L. Hennessy, 2013.
[2] 汪文祥, 刑金璋. CPU 设计实战[M]. 机械工业出版社: 汪文祥, 刑金璋, 2021.
[3] 雷思磊. 自己动手写 CPU[M]. 电子工业出版社: 雷思磊, 2014.