流水 CPU 实验报告

Misledom: 邵韵秋 王倩 杨天龙

2016 年 11 月

目录

一． 流水线模块设计

1. pc 模块

pc 模块是存储指令地址的寄存器，其基本功能是在每个时钟上升沿到来时，将 pc 寄存器中的地址传给指令存储器用来取址，并且自增一，指向当前指令的下一条指令的地址。另外，还需根据相应的输入完成暂停取址以及跳转的功能。

输入信号表

信号名称	类型	来源	功能描述
Clk	STD_LOGIC	Mcpu	时钟信号，上升沿时生效
rst	STD_LOGIC	Mcpu	异步置零，当 rst 为 0 时，回到默认状态
Branch_flag_o	STD_LOGIC	Id	跳转信号，为 1 时，pc 跳转到指定地址
Branch_addr_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	Id	跳转地址
stall	STD_LOGIC	Stall_ctrl	暂停信号，若为 0，pc 停止加一或跳转，保持原输出不变。

输出信号表

信号名称	类型	去向	功能描述
Pc_intst	STD_LOGIC_VECTOR(15 DOWNTO 0)	Ram2_ctrl	输出当前所需指令的地址，传给指令寄存器，用来取址
Pc	STD_LOGIC_VECTOR(15 DOWNTO 0)	If_id	输出当前指令的下一条地址

2. if-id 段间寄存器

基本功能为接受 pc 模块的输入，并且暂存，在下一个时钟上升沿到来时，把暂存的值传给 id。并且还能响应 stall_ctrl 的暂停指令，判断此时是要保持原值，还是插入 nop 指令，以及处理 int 指令中断，记录中断处理阶段。

输入信号表

信号名称	类型	来源	功能描述
Rst	STD_LOGIC	mcpu	异步置零
Clk	STD_LOGIC	mcpu	时钟信号，上升沿时生效
If_pc	STD_LOGIC_VECTOR(15 DOWNTO 0)	pc	Pc 阶段传入的 pc 值，指向下一条指令的地址
If_inst	STD_LOGIC_VECTOR(15 DOWNTO 0)	Ram2_ctrl	从指令存储器中读取出来的指令
Stall_pc	STD_LOGIC	Stall_ctrl	暂停 pc 信号，优先级最低，在取值访存冲突时，传给 id 阶段一条 nop 指令
Stall_id	STD_LOGIC	Strall_ctrl	优先级其次暂停信号 处理 lw 型冲突，此时传给 id 阶段的信息维持上一阶段

			的信息
Stall_ex	STD_LOGIC	Stall_ctrl	优先级最高的暂停信号，用于中断处理，记录转换中断处理的阶段。

输出信号表

信号名称	类型	去向	功能描述
id_intst	STD_LOGIC_VECTOR (15 DOWNTO 0)	id	输出 id 所需处理的指令
Id_pc	STD_LOGIC_VECTOR (15 DOWNTO 0)	id	输出 id 可能会用到的 pc 寄存器的值
Int_state	STD_LOGIC	id	输出中断处理的状态，默认为 0

3. id

基本功能为译码模块，接受 16 位二进制指令串，分析指令串的含义，并访问寄存器取得 ALU 计算所需要的数据，将解析后的指令类型，指令子类型，输入数据传递给下一阶段的模块以便进行计算。在这一阶段还需要处理 mem 和 exe 的访问寄存器引起的数据冲突，回写阶段的写寄存器引起的数据冲突。

此外，若指令为跳转指令，需要在译码阶段提前计算是否需要跳转，若指令为 int 中断指令，需要插入气泡将其分解为 sw_sp 等多条指令

输入信号表

信号名称	类型	来源	功能描述
rst	STD_LOGIC	Mcpu	异步置零
pc_i	STD_LOGIC_VECTOR(15 down to 0)	if_id	指令地址
inst_i	STD_LOGIC_VECTOR(15 down to 0)	if_id	指令内容
reg0_data_i	STD_LOGIC_VECTOR(15 down to 0)	Regfile	来自寄存器堆的输入 1
reg1_data_i	STD_LOGIC_VECTOR(15 down to 0)	regfile	来自寄存器堆的输入 2
ex_we_i	STD_LOGIC	exe	Ex 阶段的输出使能
ex_waddr_i	STD_LOGIC_VECTOR(15 down to 0)	Exe	Ex 阶段的输出地址
ex_wdata_i	STD_LOGIC_VECTOR(15 DOWNTO 0)	Exe	Ex 阶段的输出数据
mem_we_i	STD_LOGIC	mem	Mem 阶段的输出使能

mem_waddr_i	STD_LOGIC_VECTOR(15 downto 0)	mem	Mem 阶段的输出地址
mem_wdata_i	STD_LOGIC_VECTOR(15 DOWNTO 0)	mem	Mem 阶段的输出数据

输出信号表

信号名称	类型	去向	功能描述
aluop_o	STD_LOGIC_VECTOR(4 DOWNTO 0)	id_ex	Alu 指令子类型
alusel_o	STD_LOGIC_VECTOR(4 DOWNTO 0)	id_ex	Alu 指令类型
rego_data_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	id_ex	Alu 操作数 1
reg1_data_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	id_ex	Alu 操作数 2
reg0_addr_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	id_ex	Alu 操作数的寄存器地址
reg1_addr_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	id_ex	Alu 操作数的寄存器地址
we_o	STD_LOGIC	id_ex	写寄存器使能
stall_o	STD_LOGIC	stall_ctrl	暂停流水线使能
branch_flag_o	STD_LOGIC	pc	跳转使能
branch_addr_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	pc	跳转地址

4. id-ex 段间寄存器

基本功能为接受译码阶段的输入，并暂存，将 ex 阶段所需要的计算数和判断条件传给 ex 模块。同时，还需要完成针对不同的暂停信号进行插入 nop 或继续输出上一次暂存信息的功能。

输入信号表

信号名称	类型	来源	功能描述
Rst	STD_LOGIC	mcpu	异步置零
Clk	STD_LOGIC	mcpu	时钟信号，上升沿时生效
Stall_pc	STD_LOGIC	Stall_ctrl	优先级最低暂停信号，在取值访存冲突时，此阶段不做处理，将 id 阶段传入的值传给 ex 模块
Stall_id	STD_LOGIC	Strall_ctrl	优先级其次暂停信号 处理 lw 型冲突，此时传入一条 nop 指令给 ex 模块

Stall_ex	STD_LOGIC	Stall_ctrl	优先级最高的暂停信号，用于中断处理，此阶段不进行处理，将 id 阶段出入的值传给 ex 模块
Id_aluop	STD_LOGIC_VECTOR(2 down to 0)	Id	将不同指令划分到大类中的小类，判读是属于某个小类
Id_alusel	STD_LOGIC_VECTOR(2 down to 0)	Id	将所有指令分成 8 个大类，判断是属于某个大类
Id_reg0	STD_LOGIC_VECTOR(15 DOWNTO 0)	Id	传入的第一个操作数
Id_reg1	STD_LOGIC_VECTOR(15 DOWNTO 0)	Id	传入的第二个操作数
Id_we	STD_LOGIC	Id	是否写寄存器信号
Id_waddr	STD_LOGIC_VECTOR(3 DOWNTO 0)	id	写寄存器地址

输出信号表

信号名称	类型	去向	功能描述
ex_aluop	STD_LOGIC_VECTOR(2 DOWNTO 0)	ex	将不同指令划分到大类中的小类，判读是属于某个小类
ex_alusel	STD_LOGIC_VECTOR(2 DOWNTO 0)	ex	将所有指令分成 8 个大类，判断是属于某个大类
ex_reg0	STD_LOGIC_VECTOR(15 DOWNTO 0)	ex	Ex 阶段所需的第一个操作数
ex_reg1	STD_LOGIC_VECTOR(15 DOWNTO 0)	ex	Ex 阶段所需的第二个操作数
ex_we	STD_LOGIC	ex	是否写寄存器信号
ex_waddr	STD_LOGIC_VECTOR(3 DOWNTO 0)	ex	写寄存器地址

5. ex 模块

基本功能是作为整个 CPU 的 ALU 模块，根据输入的操作码和操作数进行运算，传递写地址和计算，以及访存的地址和信号，同时该模块也会针对不同的中断阶段给出相应的处理。

输入信号表

信号名称	类型	来源	功能描述
rst	STD_LOGIC	mcpu	异步置零
aluop_i	STD_LOGIC_VECTOR(2 down to 0)	Id_ex	将不同指令划分到大类中的小类，判读是属于某个小类,寄存器根据分类对应到不同的指令，对操作数进行运算
aluse_il	STD_LOGIC_VECTOR(2 down to 0)	Id_ex	将所有指令分成 8 个大类，判断是属于某个大类
reg0_i	STD_LOGIC_VECTOR(15 DOWNTO 0)	Id_ex	传入的第一个操作数

reg1_i	STD_LOGIC_VECTOR(15 DOWNTO 0)	Id_ex	传入的第二个操作数
we_i	STD_LOGIC	Id_ex	是否写寄存器信号
waddr_i	STD_LOGIC_VECTOR(3 DOWNTO 0)	Id_ex	写寄存器地址

输出信号表

信号名称	类型	去向	功能描述
we_o	STD_LOGIC	Mem_exid	是否写寄存器信号
waddr_o	STD_LOGIC_VECTOR(3 DOWNTO 0)	Mem_exid	写寄存器地址
Wdata_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	Mem_exid	计算出的写入寄存器中的值
memrw_o	STD_LOGIC_VECTOR(1 DOWNTO 0)	Mem_exid	计算出的是否需要读写内存的标志位
Memdata_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	Mem_ex	计算而得可能需要写入内存中的数
Memaddr_o	STD_LOGIC_VECTOR(15 DOWNTO 0)	Mem_ex	访存需要用到的内存地址

6. ex_mem 模块

基本功能为接受 ex 的输入，并在时钟上升沿时将输入端数据传送给输出端。同时可处理 rst 信号并将所有输出端数据置零。

输入信号表：

信号名称	类型	来源	功能描述
Clk	STD_LOGIC	Mcpu	时钟
rst	STD_LOGIC	Mcpu	Rst
Ex_we	STD_LOGIC	ex	是否写寄存器信号
Ex_waddr	RegAddrBus	ex	写寄存器地址
Ex_wdata	DataBus	ex	计算出的写入寄存器中的值
Ex_memrw	MemRWBus	ex	计算出的是否需要读写内存的标志位
Ex_memdata	DataBus	ex	需要写入内存中的数
Ex_memaddr	DataAddrBus	ex	访存需要用到的内存地址

输出信号表：

信号名称	类型	去向	功能描述
mem_we	STD_LOGIC	Mem	是否写寄存器信号
mem_waddr	RegAddrBus	Mem	写寄存器地址
mem_wdata	DataBus	Mem	计算出的写入寄存器中的值
mem_memrw	MemRWBus	Mem	计算出的是否需要读写内存的标志位

mem_memdata	DataBus	Mem	需要写入内存中的数
mem_memaddr	DataAddrBus	Mem	访存需要用到的内存地址

7. mem 模块

将 ex_mem 传入的写寄存器信号向后传，并依据通过 ex_mem 模块传入的信号决定是否读写内存，访问哪个 ram，并据此将读到的数据向后传送，以供写寄存器。特别的，在需要访问 ram2 时需要发出 stall_req 来插入气泡。

输入信号表

信号名称	类型	来源	功能描述
rst	STD_LOGIC	mcpu	rst
we_i	STD_LOGIC	ex_em	是否写寄存器信号
waddr_i	RegAddrBus	ex_em	写寄存器地址
wdata_i	DataBus	ex_mem	计算出的写入寄存器中的值
memrw_i	MemRWBus	ex_mem	计算出的是否需要读写内存的标志位
memdata_i	DataBus	ex_mem	需要写入内存中的数
memaddr_i	DataAddrBus	ex_mem	访存需要用到的内存地址
ram1_data_i	DataBus	Ram1_ctrl	数据区或串口读到的数据
ram2_data_i	DataBus	Ram2_ctrl	指令区读到的数据

输出信号表

信号名称	类型	去向	功能描述
we_o	STD_LOGIC	mem_wb	是否写寄存器信号
waddr_o	RegAddrBus	mem_wb	写寄存器地址
wdata_o	DataBus	mem_wb	计算出的写入寄存器中的值
ram1_data_o	DataBus	Ram1_ctrl	需要写入内存的数据
ram1_addr_o	DataAddrBus	Ram1_ctrl	需要写入内存的地址
ram1_re_o	STD_LOGIC	Ram1_ctrl	是否要读 ram1
ram1_we_o	STD_LOGIC	Ram1_ctrl	是否要写 ram1
ram1_ce_o	STD_LOGIC	Ram1_ctrl	Ram1 使能信号
ram2_data_o	DataBus	Ram2_ctrl	需要写入内存的数据
ram2_addr_o	DataAddrBus	Ram2_ctrl	需要写入内存的地址
ram2_re_o	STD_LOGIC	Ram2_ctrl	是否要读 ram2
ram2_we_o	STD_LOGIC	Ram2_ctrl	是否要写 ram2
ram2_ce_o	STD_LOGIC	Ram2_ctrl	Ram1 使能信号
Stall_req	STD_LOGIC	mcpu	是否有结构冲突

8. mem_wb 模块

基本功能为接受 mem 的输入，并在时钟上升沿时将输入端数据传送给输出端。同时可处理 rst 信号并将所有输出端数据置零。

输入信号表

信号名称	类型	来源	功能描述
clk	STD_LOGIC	mcpu	控制时钟
rst	STD_LOGIC	mcpu	rst
mem_we	STD_LOGIC	mem	是否写寄存器信号
mem_waddr	RegAddrBus	mem	写寄存器地址
mem_wdata	DataBus	mem	计算出或读入的写入寄存器中的值

输出信号表

信号名称	类型	来源	功能描述
wb_we	STD_LOGIC	RegisterFile	是否写寄存器信号
wb_waddr	RegAddrBus	RegisterFile	写寄存器地址
wb_wdata	DataBus	RegisterFile	计算出或读入的写入寄存器中的值

9. stall_ctrl 模块

主要根据不用的暂停请求，判断是哪一种类型的暂停，给出不同优先级的暂停信号，方便不同模块做出相应。

输入信号表

信号名称	类型	来源	功能描述
rst	STD_LOGIC	Mcpu	异步置零
Stallreq_id	STD_LOGIC	Id	从 id 收到的暂停请求，此时是遇到了 lw 类型的冲突。
Stallreq_mem	STD_LOGIC	Mem	从 mem 收到的暂停请求 ,此时是遇到了要访问指令寄存器，不得不暂停 pc 的冲突。
Stallreq_int	STD_LOGIC	id	从 id 收到的暂停请求，此时需要进行中断处理而暂停流水线

输出信号表

信号名称	类型	去向	功能描述
Stall_pc	STD_LOGIC	Pc, if_id, id_ex	优先级最低的暂停请求，表示访问指令存储器冲突
Stall_id	STD_LOGIC	If_id, id_ex	优先级其次的暂停请求，表示 lw 类型的冲突
Stallreq_int	STD_LOGIC	If_id, Id_ex	优先级最高的暂停请求，表示需要进行中断处理

10. mcpu 模块

顶层模块，负责将不同模块部件的连线，以及与板子的 sram,flash 和串口的通信。

时钟与复位

名称	数据类型	输入/输出	功能描述
Rst_in	STD_LOGIC	in	板子所给的复位信号，在连接到各个模块时，因为要先运行 flash 模块，其他所有部件是 rst 输入都是该信号与 flash 完成信号做与的结果。
Clk_50_in	STD_LOGIC	in	50M 时钟信号，模块内部进行分频处理，产生 12.5M 和 1M 的时钟。

Ram1

名称	数据类型	输入/输出	功能描述
dev_ram1_data_ready_i	STD_LOGIC	in	串口准备信号
dev_ram1_tbre_i	STD_LOGIC	in	发送数据标志
dev_ram1_tsre_i	STD_LOGIC	In	数据发送完毕标志
dev_ram1_data_bi	STD_LOGIC_VECTOR(15 DOWNTO 0)	inout	Ram1 数据总线
dev_ram1_oe_o	STD_LOGIC	out	Ram1 输出使能
dev_ram1_en_o	STD_LOGIC	out	Ram1 使能
dev_ram1_we_o	STD_LOGIC	out	Ram1 写使能
dev_ram1_addr_o	STD_LOGIC_VECTOR(17 DOWNTO 0)	out	Ram1 地址
dev_ram1_wrn_o	STD_LOGIC	out	写串口
dev_ram1_rdn_o	STD_LOGIC	out	读串口

Ram2

名称	数据类型	输入/输出	功能描述
dev_ram2_data	STD_LOGIC_VECTOR(15 DOWNTO 0)	inout	Ram2 数据总线
dev_ram2_oe_o	STD_LOGIC	out	Ram2 输出使能
dev_ram2_en_o	STD_LOGIC	out	Ram2 使能
dev_ram2_we_o	STD_LOGIC	out	Ram2 写使能

dev_ram2_addr_o	STD_LOGIC_VECTOR(17 DOWNTO 0)	out	Ram2 地址

Flash

名称	数据类型	输入/输出	功能描述
flash_data	STD_LOGIC_VECTOR(15 DOWNTO 0)	inout	flash 数据总线
flash_byte	STD_LOGIC	out	操作模式
flash_vpen	STD_LOGIC	out	写保护
flash_ce	STD_LOGIC	out	使能
flash_oe	STD_LOGIC	out	输出使能
flash_we	STD_LOGIC	out	写使能
flash_rp	STD_LOGIC	out	0 重置
flash_addr	STD_LOGIC_VECTOR(22 DOWNTO 1)	out	flash 地址

二． 存储单元模块设计

1. ram1_ctrl 模块

sram1 和串口 uart 控制器，接收来自 mem 模块的读写请求，并操作实际硬件设备的控制信号来读取对应的数据

具体来说，需要根据读写地址来分别进行 ram，串口数据，串口状态读写

输入信号表

信号名称	类型	来源	功能描述
Clk	STD_LOGIC	mcpu	时钟信号，不进行上升下降检测，只用于写信号的拉低以及拉高。
Rst	STD_LOGIC	mcpu	异步至零
mem_data_i	DataBus	Ram1_ctrl	需要写入内存的数据
mem_addr	DataAddrBus	Ram1_ctrl	需要访问内存的地址
mem_re	STD_LOGIC	Ram1_ctrl	是否要读 ram2
mem_we	STD_LOGIC	Ram1_ctrl	是否要写 ram2
mem_ce	STD_LOGIC	Ram1_ctrl	Ram2 使能信号
Ram_data_ready_i	STD_LOGIC	Ram1_ctrl	Uart 的读寄存器信号
Ram_tbre_i	STD_LOGIC	Ram1_ctrl	Uart 的写寄存器信号
Ram_tsre_i	STD_LOGIC	Ram1_ctrl	Uart 的写寄存器信号

输出信号表

信号名称	类型	去向	功能描述

Ram_wrn	STD_LOGIC	Ram1_ctrl	Uart 的写控制信号
Ram_rdn	STD_LOGIC	Ram1_ctrl	Uart 的读控制信号
mem_data_o	DataBus	mem_wb	返回给 mem 的读到的数据
ram_data	DataBus	mcpu	连接 ram2 的 inout 线
ram_addr_o	RamAddrBus	mcpu	需要访问内存的地址
ram_oe_o	STD_LOGIC	mcpu	是否要读 ram2
ram_we_o	STD_LOGIC	mcpu	是否要写 ram2
ram_en_p	STD_LOGIC	mcpu	Ram2 使能信号

2. ram2_ctrl 模块

有优先级地（优先 mem）处理 pc 和 mem 的内存读写请求。控制 ram2 的读写控制信号和地址数据，并将 inout 线读到的数据转换成 out 类型返回给所需部件。

输入信号表

信号名称	类型	来源	功能描述
Clk	STD_LOGIC	mcpu	时钟信号，不进行上升下降检测，只用于写信号的拉低以及拉高。
pc_addr	DataAddrBus	pc	读指令的 pc 值
mem_data_i	DataBus	Ram1_ctrl	需要写入内存的数据
mem_addr	DataAddrBus	Ram1_ctrl	需要访问内存的地址
mem_re	STD_LOGIC	Ram1_ctrl	是否要读 ram2
mem_we	STD_LOGIC	Ram1_ctrl	是否要写 ram2
mem_ce	STD_LOGIC	Ram1_ctrl	Ram2 使能信号

输出信号表

信号名称	类型	去向	功能描述
Inst	InstBus	pc_id	读到的指令
mem_data_o	DataBus	mem_wb	返回给 mem 的读到的数据
ram_data	DataBus	mcpu	连接 ram2 的 inout 线
ram_addr_o	RamAddrBus	mcpu	需要访问内存的地址
ram_oe_o	STD_LOGIC	mcpu	是否要读 ram2
ram_we_o	STD_LOGIC	mcpu	是否要写 ram2
ram_en_p	STD_LOGIC	mcpu	Ram2 使能信号

3. RegisterFile 模块

基本功能为通用寄存器组模块，包括除了 R0~R7 8 个通用寄存器以外还有 IH, SP, T 三个寄存器，用 4 位地址去索引。主要完成的就是读写暂存功能。

输入信号表

信号名称	类型	来源	功能描述
clk	STD_LOGIC	mcpu	时钟信号，检测下降沿，下降沿写入
rst	STD_LOGIC	mcpu	异步置零
re_0	STD_LOGIC	Id	是否读寄存器
re_1	STD_LOGIC	Id	是否读寄存器

raddr0	STD_LOGIC_VECTOR (3 DOWNTO 0)	Id	读寄存器地址
raddr1	STD_LOGIC_VECTOR (3 DOWNTO 0)	Id	读寄存器地址
we	STD_LOGIC	Mem_wb	是否写寄存器信号
waddr	STD_LOGIC_VECTOR (3 DOWNTO 0)	Mem_wb	写寄存器地址
wdata	STD_LOGIC_VECTOR (3 DOWNTO 0)	Mem_wb	要写入寄存器的值

输出信号表

信号名称	类型	去向	功能描述
rdata0	STD_LOGIC_VECTOR (15 DOWNTO 0)	Id	寄存器中读取的数
rdata1	STD_LOGIC_VECTOR (15 DOWNTO 0)	Id	寄存器中读取的数

三． 扩展功能

1. Int 软件中断

实现思路：阅读 kernel 的代码，不难发现监控程序中有专门的中断处理模块，我们需要做的是解析 int 指令，将对应的中断号和用户程序返回地址存入栈中相应的位置。相当于实现如下四条汇编指令：

ADDSP FF
SW_SP PC
ADDSP FF
SW_SP immint(其中 immint 表示中断号)

因为一条指令相当于四条汇编语句 ,所以一个周期肯定是做不完的 ,流水线需要相应的暂停，并且在 if_id 阶段记录中断处理的状态，id 阶段根据不同的状态给出操作码和操作数。

虽然相当于四条汇编指令 , 但如果两两同时执行的话 , 只用暂停一个时钟周期 , 加快处理速度，且不难实现。相应的，将中断处理划分成两个阶段：第一阶段，id 阶段获取 sp 寄存器的值，以及 pc 值，写寄存器信号和地址，同时给出暂停流水线的请求给 stall_ctrl。将其传给 ex，ex 检测到此时是中断处理第一阶段，给出写内存信号，地址和数，将写入寄存器的数附为 sp-1,将写内存的地址给成 sp-1,将 pc 值作为写入内存的数，传给下一模块。If_id 收到因为 Int 而产生的暂停流水线的请求，把对应的状态标记置为 1，提示进入中断处理的第二阶段，与第一阶段类似，id 给出 sp 寄存器的值，中断号，写寄存器信号和地址，并且要将跳转信号拉高，将跳转地址传给 pc，让 pc 跳转到中断处理程序的位置，将暂停流水线的请求恢复。Ex 模块检测到进入中断处理的第二阶段，将写入寄存器的值给成 sp-1 给出写内存信号，将写内存的地址置为 sp-1,将写入的数置为中断号即可。

最后实现的效果为，如果用户输出的程序中有 int 指令，则运行时会输出中断号，之后继续

运行用户程序。

2. PS2 键盘

Keybord 模块
用于实际 PS2 键盘的驱动，采用 50MHZ 采样键盘信号并进行防抖处理，将 ps2 键盘信号转换为更加稳定的使能和数据信号

输入信号表

信号名称	类型	来源	功能描述
Data_in	STD_LOGIC	ps2	键盘 ps2 数据信号
clkin	STD_LOGIC	ps2	键盘 ps2 时钟信号
Fclk	STD_LOGIC	mcpu	高频采样时钟
Rst	STD_LOGIC	mcpu	异步清零

输出信号表

信号名称	类型	去向	功能描述
scancode	STD_LOGIC_VECTOR(7 DOWNTO 0)	Keboard_ctr	键盘信号输出
Fok	STD_LOGIC	Keyboard_ctel	键盘使能输出

Keybord_ctrl 模块
用于将键盘信号转换为串口信号，使用串口协议连接 ram1_ctrl 和 vga

输入信号表

信号名称	类型	来源	功能描述
Rst	STD_LOGIC	ps2	异步清零
k_data	STD_LOGIC	ps2	键盘 ps2 数据信号
K_clk	STD_LOGIC	mcpu	键盘 ps2 时钟信号
Clk_50	STD_LOGIC	mcpu	高频采样时钟

输出信号表

信号名称	类型	去向	功能描述
Data_ready_out	STD_LOGIC	Ram1_ctrl	数据信号输出
Data_out	STD_LOGIC_VECTOR(7 DOWNTO 0)	Ram1_ctrl	数据输出
One_key_we	STD_LOGIC	Vga	数据信号输出
One_key_data	STD_LOGIC_VECTOR(7 DOWNTO 0)	vga	数据输出

Keybord_to_vga_decoder 模块
用于将键盘扫描码数据转换为 vga 显示所用码

输入信号表

信号名称	类型	来源	功能描述
Kdata	STD_LOGIC_VECTOR(7 downto 0)	Keyboard_ctrl	键盘扫描码

输出信号表

信号名称	类型	去向	功能描述
vdata	STD_LOGIC_VECTOR(7 downto 0)	vgal	Vga 显示码

3. VGA

Draw_char 模块
用于存储空格、0-9、A-Z 的显示信息。
输入信号表

信号名称	类型	来源	功能描述
subX	STD_LOGIC_VECTOR(3 DOWNTO 0)	Vga_char	单个字符内需要访问的相对 x 坐标
subY	STD_LOGIC_VECTOR(3 DOWNTO 0)	Vga_char	单个字符内需要访问的相对 y 坐标
chr	STD_LOGIC_VECTOR(5 DOWNTO 0)	Vga_char	此时需要显示的字符

输出信号表

信号名称	类型	去向	功能描述
Valid	STD_LOGIC	Vga_char	是否为显示部分

Vga_char 模块(vga.v)
根据键盘和 cpu 的输入信号分别在显示屏左右两部分显示相应数据，键盘部分分为两种模式，串口精灵模式下有效输入为两个十六进制数加一个空格，显示十六进制，写字板模式下可输入任意字母与数字的组合，可删除可换行/空格。
输入信号表

信号名称	类型	来源	功能描述
Clk_50	STD_LOGIC	Mcpu	25M 时钟
Rst_n	STD_LOGIC	Mcpu	Rst 信号
K_we	STD_LOGIC	keyboard	Keyboard 串口精灵模式下给的写信号
inChar	SerialDataBus	keyboard	Keyboard 的数据
One_key_we	STD_LOGIC	Keyboard	Keyboard 写字板模式下给的写信号
One_key_data	SerialDataBus	keyboard	Keyboard 传来的数据（需译码）
cpu_we	STD_LOGIC	Ram1_ctrl	Ram1 串口给的写信号

Cpu_inChar	SerialDataBus	Ram1_ctrl	Ram1 通过串口写的数据

输出信号表

信号名称	类型	去向	功能描述
hsync	STD_LOGIC	硬件 vga	行同步信号
vsync	STD_LOGIC	硬件 vga	场同步信号
Vga_rgb	STD_LOGIC_VECTOR(8 DOWNTO 0)	硬件 vga	此时所需 rgb 信息

4. Flash

Flahs_io 模块用于在 cpu 启动时从 flash 读入程序代码（也就是 kernal.bin）到 ram2，然后再运行流水线。相关的控制在 mcpu 中完成，flash 模块执行 flash 的读，mcpu 中控制 ram2_ctrl 执行 ram2 的写。

Flash_io 模块

输入信号表：

信号名称	类型	来源	功能描述
Clk	STD_LOGIC	mcpu	Flash 运行的时钟，与主频不同
Reset	STD_LOGIC	mcpu	Reset 后会重新读一遍

输出信号表：

信号名称	类型	去向	功能描述
data_out	DataBus	mcpu	读出的数据
addr_out	DataAddrBus	mcpu	要写入 ram2 的地址
booting	STD_LOGIC	mcpu	标识是否完成 flash 所有读取
flash_byte	STD_LOGIC	硬件 flash	Flash 模式信号，1-字模式
flash_vpen	STD_LOGIC	硬件 flash	Flash 写保护信号，常 1
flash_ce	STD_LOGIC	硬件 flash	Flash 使能信号
flash_oe	STD_LOGIC	硬件 flash	读使能
flash_we	STD_LOGIC	硬件 flash	写使能
flash_rp	STD_LOGIC	硬件 flash	Flash 控制信号，常 1
flash_addr	FlashAddrBus	硬件 flash	Flash 地址线
flash_data	DataBus	硬件 flash	Flash 数据线

Mcpu 中相关控制代码

```
rst <= rst_in and booting ;
clk <= clk_12_5 and booting;
ram2_we_m <= ram2_we_i or not booting;
ram2_ce_m <= ram2_ce_i or not booting;
process(booting,addr_out,data_out,ram2_addr_i,ram2_data_i)begin
    if( booting = '0') then
        ram2_addr_m<=addr_out;
        ram2_data_m<=data_out;
    else
        ram2_addr_m<=ram2_addr_i;
```

```
            ram2_data_m<=ram2_data_i;
        end if;
end process;
```

四． 实验结果

五． 实验小结与感想

1. 多语言问题

由于一开始打算用 verilog 写，实验三的时候发现 verilog 有很多坑，于是又改回 vhdl，所以最后的工程有的用 verilog 有的用 vhdl，虽然没什么问题，但在编写和调试过程中经常需要在两种语言中切换，也会带来一些语法写混带来的问题导致难以找出 bug。

2. Process 写法问题

多 process 思路上可能会比较清晰，但时序上可能会有奇怪的问题。推荐纯组合逻辑写法或者一个大的 process 来保证时序。

3. 仿真与调试

因为整个工程量较大，根本没有办法保证一次性写对，所以仿真是烧板子之前很重要的一步。我们组的仿真分成几个阶段，包括每个部件的单独仿真，单条指令仿真，以及整条流水线的时序仿真。因为仿真时无法使用 sram，所以只能写 ram_fake 来模拟。在这样的仿真之后，虽然烧板子的实际过程是还是遇到了问题，但这样很容易定位到是访存和串口的问题，方便了对问题的定位和调试。

六． 小组分工

Item	Detail	People
数据通路	数据通路设计	杨天龙
流水线 cpu	Pc	邵韵秋
	Pc_id	邵韵秋
	Id	杨天龙
	Id_ex	邵韵秋
	Ex	邵韵秋
	Ex_mem	王倩
	Mem	王倩
	Mem_wb	王倩
	RegisterFile	邵韵秋
	Stall_ctrl	邵韵秋
	Mcpu	邵韵秋
	Ram1_ctrl	杨天龙
	Ram2_ctrl	王倩
	整体仿真	邵韵秋 王倩
	多时钟	杨天龙
	提升频率与调试	邵韵秋 王倩
扩展	Keyboard	杨天龙
	Vga	王倩

	Flash	王倩
	Int	邵韵秋