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 4](#_Toc468450724)

[1. pc模块 4](#_Toc468450725)

[2. if-id 段间寄存器 5](#_Toc468450726)

[3. id 6](#_Toc468450727)

[4. id-ex段间寄存器 6](#_Toc468450728)

[5. ex 模块 8](#_Toc468450729)

[6. ex\_mem模块 10](#_Toc468450730)

[7. mem模块 11](#_Toc468450731)

[8. mem\_wb模块 12](#_Toc468450732)

[9. stall\_ctrl模块 13](#_Toc468450733)

[10. mcpu模块 14](#_Toc468450734)

[二． 存储单元模块设计 16](#_Toc468450735)

[1. ram1\_ctrl模块 16](#_Toc468450736)

[2. ram2\_ctrl模块 16](#_Toc468450737)

[3. RegisterFile 模块 18](#_Toc468450738)

[三． 扩展功能 19](#_Toc468450739)

[1. Int软件中断 19](#_Toc468450740)

[2. PS2键盘 20](#_Toc468450741)

[3. VGA 20](#_Toc468450742)

[4. Flash 22](#_Toc468450743)

[四． 实验结果 23](#_Toc468450744)

[五． 实验小结与感想 23](#_Toc468450745)

[六． 小组分工 24](#_Toc468450746)

# 流水线模块设计

## 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 |

## 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 |

## 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 down to 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 |

## 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 |

## 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\_ex  id | 是否写寄存器信号 |
| waddr\_o | STD\_LOGIC\_VECTOR(3 DOWNTO 0) | Mem\_ex  id | 写寄存器地址 |
| Wdata\_o | STD\_LOGIC\_VECTOR(15 DOWNTO 0) | Mem\_ex  id | 计算出的写入寄存器中的值 |
| memrw\_o | STD\_LOGIC\_VECTOR(1 DOWNTO 0) | Mem\_ex  id | 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## 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 |

# 存储单元模块设计

## 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 |

## 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 |

## 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 |

# 扩展功能

## 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。

## 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 |

## 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 |

## 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; |

# 实验结果

# 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。

1. 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。

1. 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。

# 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 |