**2020-2021 第1学期**

**计算机组成原理实验报告**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | **计算机与通信工程学院** | | | | | | | |
| **专业班级** | **计算机科学与技术1802** | | | | | | | |
| **班级序号** | **35** | | | | | | | |
| **学号** | **20188068** | | | | | | | |
| **姓名** | **孔天欣** | | | | | | | |
| **指导教师** | **张旭** | | | | | | | |
| **成绩** |  |  |  |  |  |  |  |  |

**计 算 机 组 成 原 理 实 验 报 告**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**班级： 计科1802 姓名： 孔天欣 班级序号： 35 学号：20188068**

**实验日期：2020.10.19**

**学院： 计算机与通信工程学院 专业：计算机科学与技术**

**实验顺序：31 实验名称：数字集成电路设计方法 指导教师：张旭**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验目的**

1. 了解数字集成电路设计方法。

2. 熟悉并运用 Verilog 语言进行电路设计。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验环境**

装有 Xilinx Vivado 的计算机一台。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验设计图**

1. 裁判表决器

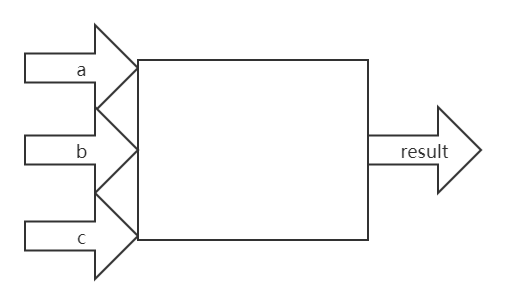


图1.1 裁判表决器

2. 基本D 触发器

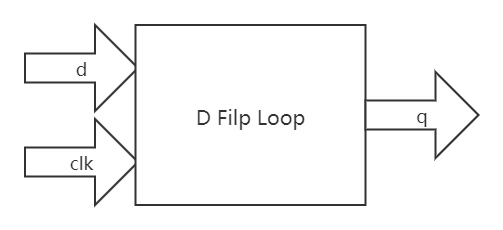


图1.2 D触发器

3. 带低电平有效异步复位端的触发器

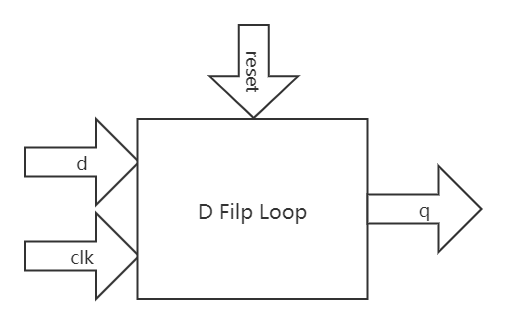


图1.3 带低电平有效异步复位端的触发器

4. 带同步复位端的D触发器

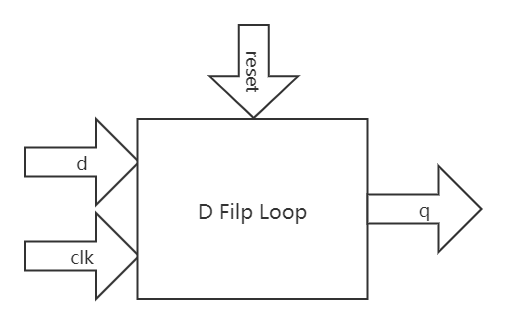


图1.4 带同步复位端的D触发器

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验代码**

一、裁判表决器

1. 裁判表决器电路代码——设计模块

**module** referee(

**input** wire r1,

**input** wire r2,

**input** wire rm,

**output** reg res

);

**always** @(r1 **or** r2 **or** rm) **begin**

res = rm & (r1 | r2);

**end**

**endmodule**

2. 裁判表决器电路代码——测试模块

**module** referee\_test();

reg a;

reg b;

reg c;

wire result;

referee referee1(a,b,c,result);

**initial** **begin**

a=0;b=0;c=0;

#10 a=0;b=0;c=1;

#10 a=0;b=1;c=0;

#10 a=0;b=1;c=1;

#10 a=1;b=0;c=0;

#10 a=1;b=0;c=1;

#10 a=1;b=1;c=0;

#10 a=1;b=1;c=1;

#10 $finish;

**end**

**initial** **begin**

$monitor($time," , a=%b ,b=%b ,c=%b ,result=%b",a,b,c,result);

**end**

**endmodule**

二、D触发器

1. 基本D 触发器电路代码——设计模块

**module** simple\_d(

**input** wire clk,

**input** wire d,

**output** reg q

);

**always** @(**posedge** clk) **begin**

q <= d;

**end**

**endmodule**

2. 基本D 触发器电路代码——测试模块

**module** d\_test();

reg d;

reg clk;

wire q;

simple\_d simpled(clk,d,q);

**initial** **begin**

d = 1;

clk = 0;

**forever** #100 clk = ~clk;

**end**

**always** #200 d = ~d;

**endmodule**

3. 带低电平有效异步复位端的触发器——设计模块

**module** plus\_d(

**input** wire clk,

**input** wire d,

**input** wire reset,

**output** reg q

);

**always** @(**posedge** clk **or** **negedge** reset) **begin**

**if**(!reset)

**begin**

q <= 0;

**end**

**else**

**begin**

q <= d;

**end**

**end**

**endmodule**

4. 带低电平有效异步复位端的触发器——测试模块

**module** d\_test();

reg d;

reg clk;

reg reset;

wire q;

plus\_d plusd(clk,d,reset,q);

**initial** **begin**

d = 1;

clk = 0;

reset = 1;

**forever** #10 clk = ~clk;

**end**

**initial** **begin**

#15 reset = 0;

#10 reset = 1;

**end**

**always** #20 d = ~d;

**initial** **begin**

$monitor($time," %b %b %b %b",d,clk,reset,q);

**end**

**endmodule**

5. 带同步复位端的D触发器——设计模块

**module** plus\_d2(

**input** wire clk,

**input** wire d,

**input** wire reset,

**output** reg q

);

**always** @(**posedge** clk) **begin**

**if**(!reset)

**begin**

q <= 0;

**end**

**else**

**begin**

q <= d;

**end**

**end**

**endmodule**

6. 带同步复位端的D触发器——测试模块

**module** d\_test();

reg d;

reg clk;

reg reset;

wire q;

plus\_d2 plusd2(clk,d,reset,q);

**initial** **begin**

d = 1;

clk = 0;

reset = 1;

**forever** #10 clk = ~clk;

**end**

**initial** **begin**

#30 reset = 0;

#40 reset = 1;

**end**

**always** #20 d = ~d;

**initial** **begin**

$monitor($time," %b %b %b %b",d,clk,reset,q);

**end**

**endmodule**

1. **仿真波形及说明**

1. 裁判表决器的仿真波形图

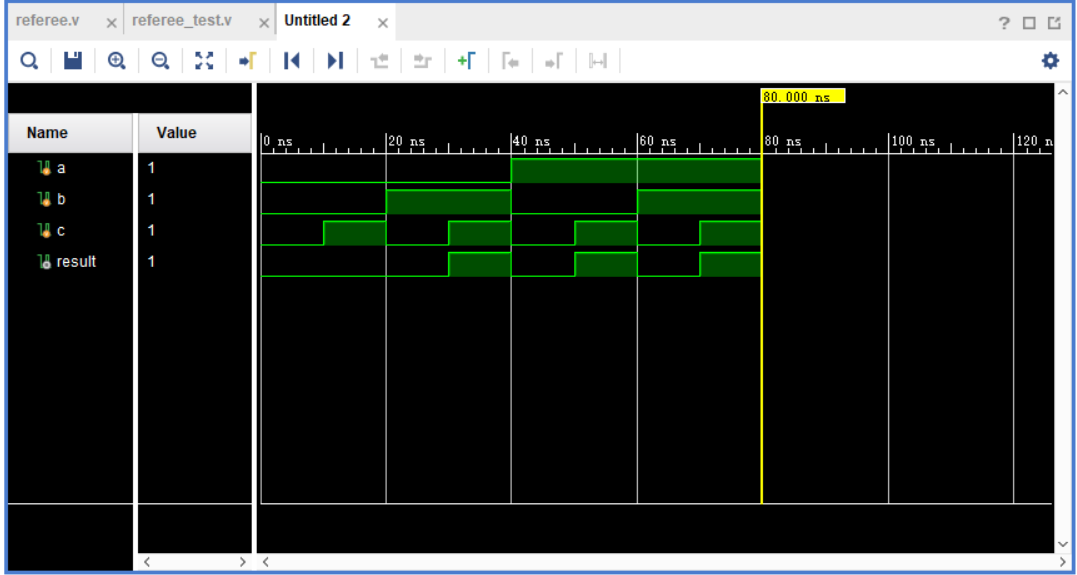


图1.5 裁判表决器波形图

说明：c是主裁判，a,b是另外两个裁判。仅有c为1且a,b有一个或以上为1时，result是1；在30 ns时，b和c都为1，因此result是1；在45 ns时，a和c都为1，因此result是1。

电路图：

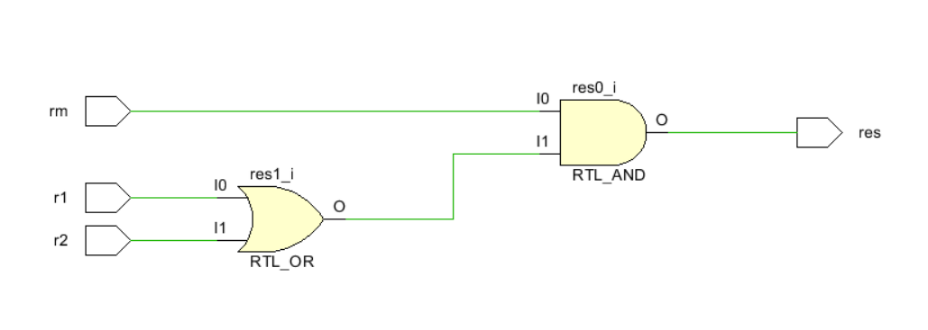


图1.6 裁判表决器电路图

2. D 触发器波形图

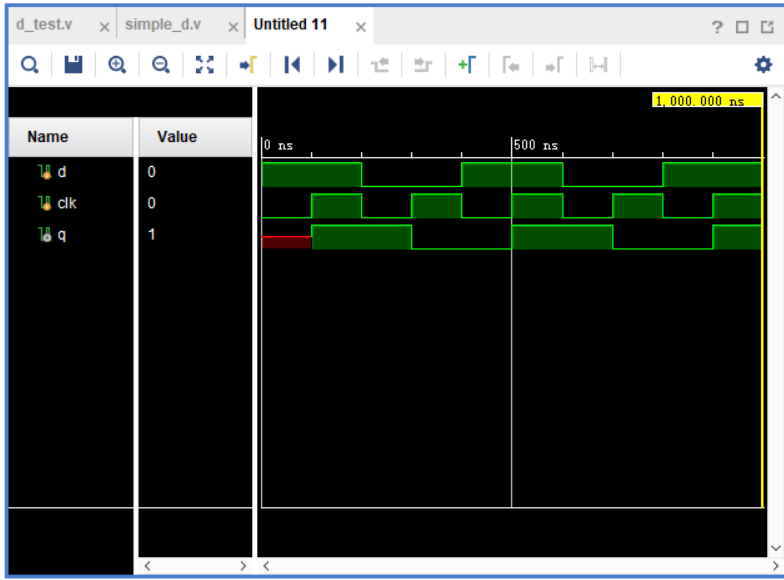


图1.7 D 触发器波形图

说明：波形图中clk初始化为0，每隔100 ns翻转一次，d初始化为1，每隔200 ns翻转一次。由于D触发器的输出q在clk处于上升沿时，变成d的电位，因此在100 ns时，由于d = 1，q置1，在300 ns时，由于d = 0，q置0。

电路图：

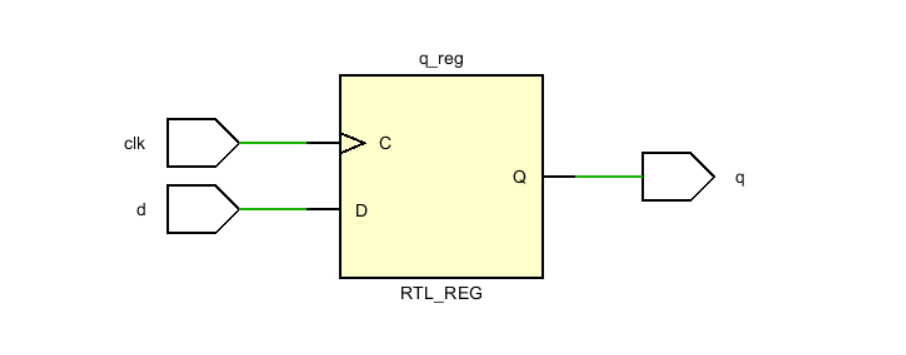


图1.8 D触发器电路图

3. 带低电平有效异步复位端的D触发器波形图

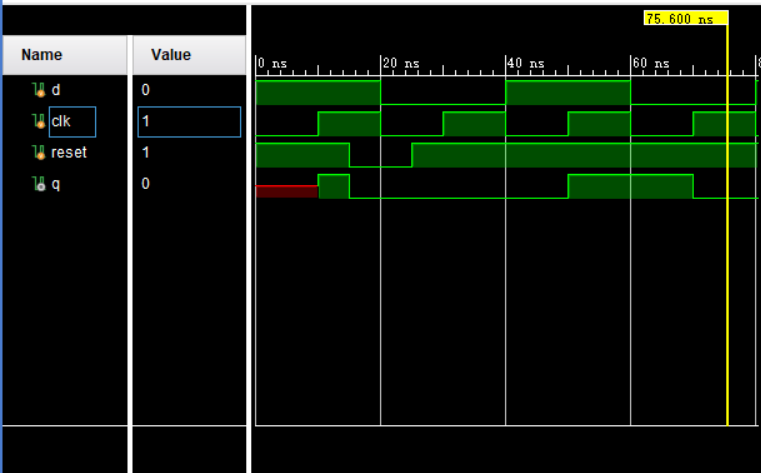


图1.9 带低电平有效异步复位端的D触发器波形图

说明：代码中clk初始化为0，每隔10 ns翻转一次，reset初始化为1，在15 ns设置为0，再过10 ns后再设置为1.d初始化为1，每隔20 ns反转一次。波形图中可见，在10 ns时，时钟处于上升沿且d为1，因此q为1.在15 ns时，reset = 0,因此q被复位为0。

电路图：

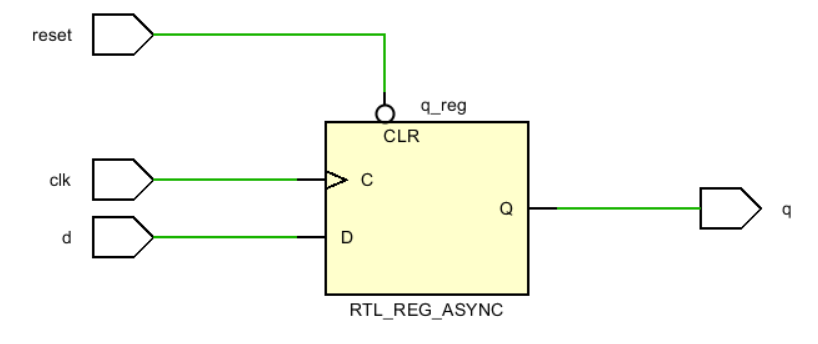


图1.10 带低电平有效异步复位端的D触发器电路图

4. 带同步复位端的D触发器

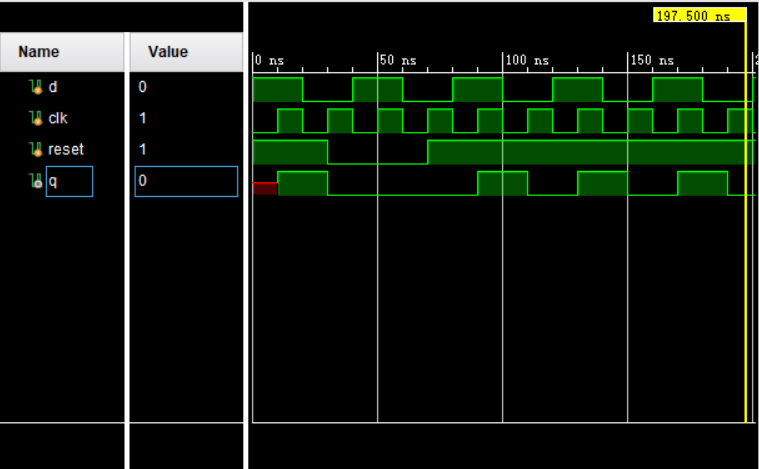


图1.11 带同步复位端的D触发器波形图

说明：因为在clk时钟上升沿和复位信号reset同时有效，才可以复位，所以在50 ns时发生了复位。

电路图：

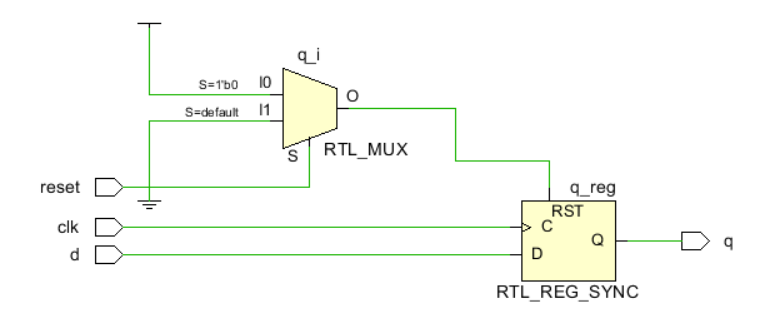


图1.12 带同步复位端的D触发器电路图

1. **实验心得体会**

通过本次实验，本人初步理解并掌握了Verilog语言的基础语法及其功能特性，此外还初步学会了vivado的使用方法，并成功通过硬件描述的方式实现了裁判表决电路、D触发器等，并能够通过波形图分析其对应的实现效果。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**计 算 机 组 成 原 理 实 验 报 告**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**班级： 计科1802 姓名：孔天欣 班级序号：180235 学号：20188068**

**实验日期：2020.10.26**

**学院： 计算机与通信工程学院 专业： 计算机科学与技术**

**实验顺序：31 实验名称：寄存器实验 指导教师：张旭**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验目的**

1.熟悉并掌握MIPS计算机中寄存器堆的原理和设计方法。

2.初步了解MIPS指令结构和源操作数/目的操作数的概念。

3.熟悉并运用verilog语言进行电路设计。

4.为后续设计cpu的实验打下基础。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验环境**

装有 vivado 软件的计算机一台。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验设计图**

一、程序计数器PC

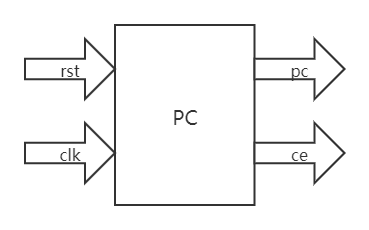


图2.1 程序计数器设计框图

表2.1 pc模块的接口描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度（bit） | 输入/输出 | 作用 |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | pc | 32 | 输出 | 要读取的指令地址 |
| 4 | ce | 1 | 输出 | 指令存储器使能信号 |

二、寄存器堆

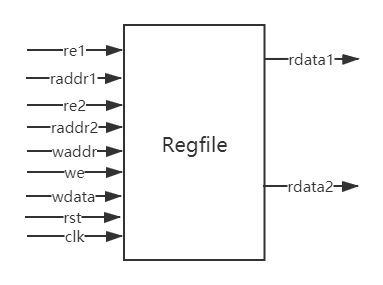


图2.2 寄存器堆设计框图

表2.2 Regfile模块的接口描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度（bit） | 输入/输出 | 作用 |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | re1 | 1 | 输入 | 读使能信号1 |
| 4 | raddr1 | 5 | 输入 | 读取的寄存器地址1 |
| 5 | re2 | 1 | 输入 | 读使能信号2 |
| 6 | raddr2 | 5 | 输入 | 读取的寄存器地址2 |
| 7 | we | 1 | 输入 | 写使能信号 |
| 8 | waddr | 5 | 输入 | 写入的寄存器地址 |
| 9 | wdata | 32 | 输入 | 写入的数据 |
| 10 | rdata1 | 32 | 输出 | 读出的32位数据1 |
| 11 | rdata2 | 32 | 输出 | 读出的32位数据2 |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验代码**

一、全局定义模块 define.v

// 复位

`define **RstEnable** 1**'**b1

`define **RstDisable** 1**'**b0

// 0

`define **ZeroWord** 32**'**h00000000

// 可写

`define **WriteEnable** 1**'**b1

`define **WriteDisable** 1**'**b0

// 可读

`define **ReadEnable** 1**'**b1

`define **ReadDisable** 1**'**b0

// 寄存器地址

`define **RegAddrBus** 4:0

// 32位数据

`define **RegBus** 31:0

`define **RegWidth** 32

`define **DoubleRegWidth** 64

`define **DoubleRegBus** 63:0

// 寄存器个数

`define **RegNum** 32

`define **RegNumLog2** 5

`define **NOPRegAddr** 5**'**

二、程序计数器PC

1. 设计模块 pc\_reg.v

`timescale 1ns / 1ns

module pc\_reg(

input wire rst,

input wire clk,

output reg[31:0] pc,

output reg ce

);

always@(posedge clk) begin

**if**(rst==1) begin

ce<=0;

**end** **else** begin

ce<=1;

**end**

**end**

always@(posedge clk) begin

**if**(ce==0) begin

pc<=32**'**h0000\_0000;

**end** **else** begin

pc <= pc + 32**'**h4;

**end**

**end**

endmodule

2. 测试模块 pc\_reg\_tb.v

`timescale 1ns / 1ns

module pc\_reg(

input wire rst,

input wire clk,

output reg[31:0] pc,

output reg ce

);

always@(posedge clk) begin

**if**(rst==1) begin

ce<=0;

**end** **else** begin

ce<=1;

**end**

**end**

always@(posedge clk) begin

**if**(ce==0) begin

pc<=32**'**h0000\_0000;

**end** **else** begin

pc <= pc + 32**'**h4;

**end**

**end**

endmodule

三、寄存器堆

1. 设计模块 regfile.v

`timescale 1ns / 1ns

module regfile(

input wire clk,

input wire rst,

// write

input wire we,

input wire[`**RegAddrBus**] waddr,

input wire[`**RegBus**] wdata,

//read 1

input wire re1,

input wire[`**RegAddrBus**] raddr1,

output reg[`**RegBus**] rdata1,

//read 2

input wire re2,

input wire[`**RegAddrBus**] raddr2,

output reg[`**RegBus**] rdata2

);

reg[`**RegBus**] regs[0:`**RegNum**-1];

always @ (posedge clk) begin

**if**(rst == `**RstDisable**) begin

**if**((we == `**WriteEnable**) && (waddr != `**RegNumLog2'h0**)) begin

regs[waddr] <= wdata;

**end**

**end**

**end**

always @(\*) begin

**if**(rst == `**RstEnable**) begin

rdata1 <= `**ZeroWord**;

**end** **else** **if** (raddr1 == `**RegNumLog2'h0**) begin

rdata1 <= `**ZeroWord**;

// read **and** write

**end** **else** **if**((raddr1 == waddr) && (we == `**WriteEnable**)

&& (re1 == `**ReadEnable**)) begin

rdata1 <= wdata;

**end** **else** **if** (re1 == `**ReadEnable**) begin

rdata1 <= regs[raddr1];

**end** **else** begin

rdata1 <= `**ZeroWord**;

**end**

**end**

always @(\*) begin

**if**(rst == `**RstEnable**) begin

rdata2 <= `**ZeroWord**;

**end** **else** **if** (raddr2 == `**RegNumLog2'h0**) begin

rdata2 <= `**ZeroWord**;

**end** **else** **if**((raddr2 == waddr) && (we == `**WriteEnable**)

&& (re2 == `**ReadEnable**)) begin

rdata2 <= wdata;

**end** **else** **if** (re2 == `**ReadEnable**) begin

rdata2 <= regs[raddr2];

**end** **else** begin

rdata2 <= `**ZeroWord**;

**end**

**end**

endmodule

2. 测试模块 regfile\_tb.v

`timescale 1ns / 1ns

module regfile\_tb();

reg clk;

reg rst;

reg we;

reg[`**RegAddrBus**] waddr;

reg[`**RegBus**] wdata;

reg re1;

reg[`**RegAddrBus**] raddr1;

wire[`**RegBus**] rdata1;

//read 2

reg re2;

reg[`**RegAddrBus**] raddr2;

wire[`**RegBus**] rdata2;

regfile regfile1(clk,rst,we,waddr,wdata,

re1,raddr1,rdata1,re2,raddr2,rdata2);

integer i;

integer j;

integer k;

initial clk = 1;

always #10 clk = ~clk;

initial begin

// 开启复位操作

rst = `**RstEnable**;

we = `**WriteDisable**;

re1 = `**ReadDisable**;

re2 = `**ReadDisable**;

waddr = `**ZeroWord**;

raddr1 = `**ZeroWord**;

raddr2 = `**ZeroWord**;

#5

// 开启写操作

rst = `**RstDisable**;

we = `**WriteEnable**;

**for**(j = 1;j < 32;j = j+1) begin

waddr = j;

wdata = j;

#50;

**end**

we = `**WriteDisable**;

#5;

// 开启读 1 操作

re1 = `**ReadEnable**;

**for**(i = 1;i < 32;i = i+1) begin

raddr1 = i;

#50;

**end**

re1 = `**ReadDisable**;

#5;

// 开启读 2 操作

re2 = `**ReadEnable**;

**for**(k = 1;k < 32;k = k+1) begin

raddr2 = k;

#50;

**end**

re2 = `**ReadDisable**;

$finish;

**end**

initial begin

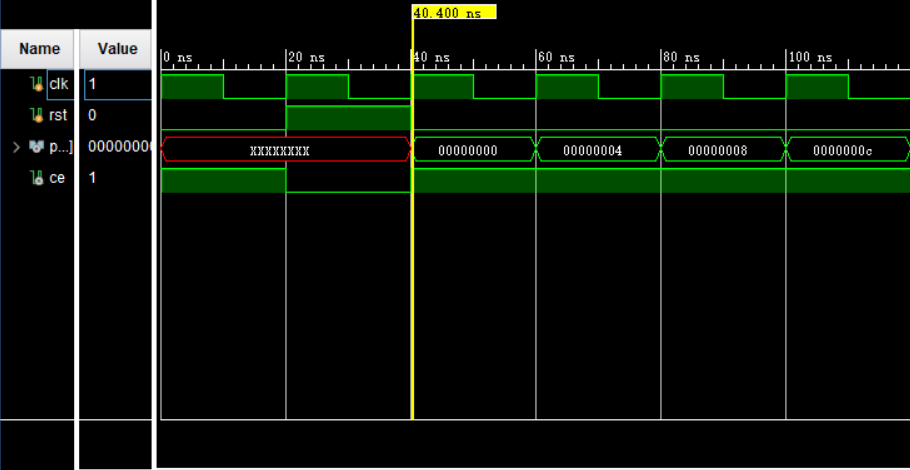
$monitor($time," rdata1= %10,rdata2 = %h",rdata1,rdata2);

**end**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

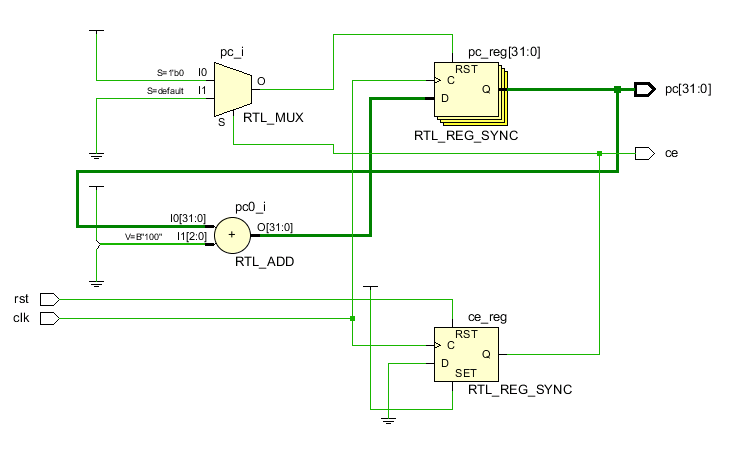
**四. 仿真波形及说明**

一、程序计数器

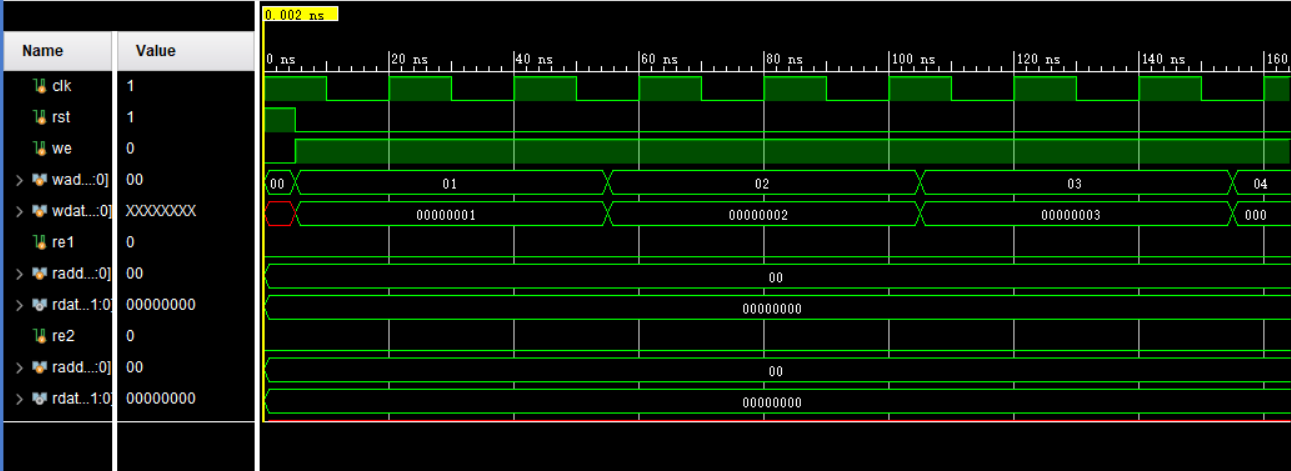


说明：图中时钟clk每隔10 ns翻转一次，同时复位信号rst在一开始置0，此时指令存储器使能信号ce为1，说明允许指令存储器工作。在20 ns时，复位信号rst置1，导致ce变为0，即暂停指令存储器工作。从40 ns开始，每隔20ns，在clk时钟上升沿的情况下，指令地址计数pc自增4，跳转到下一指令地址，因此实现了程序计数器的效果。

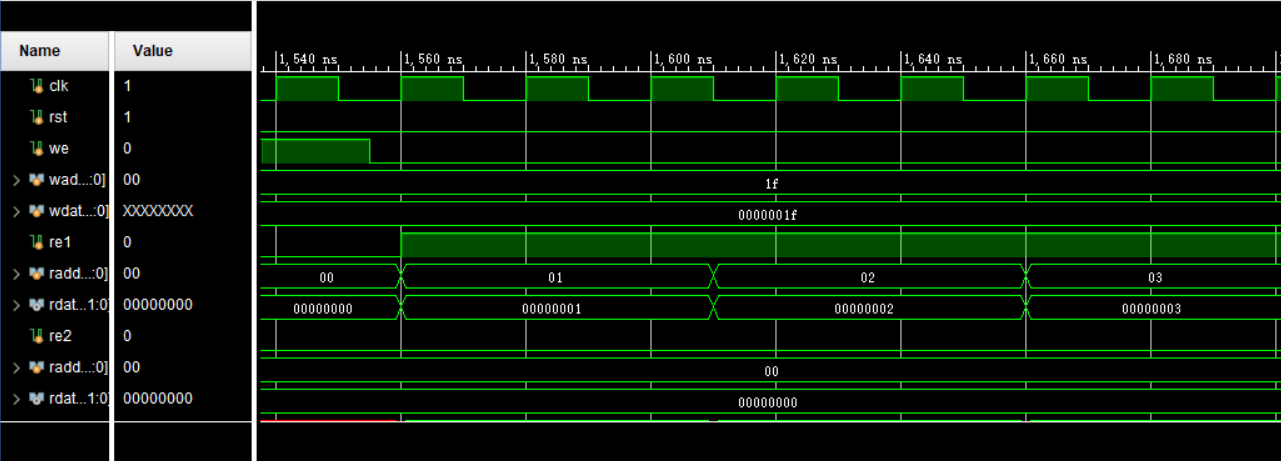
电路图：



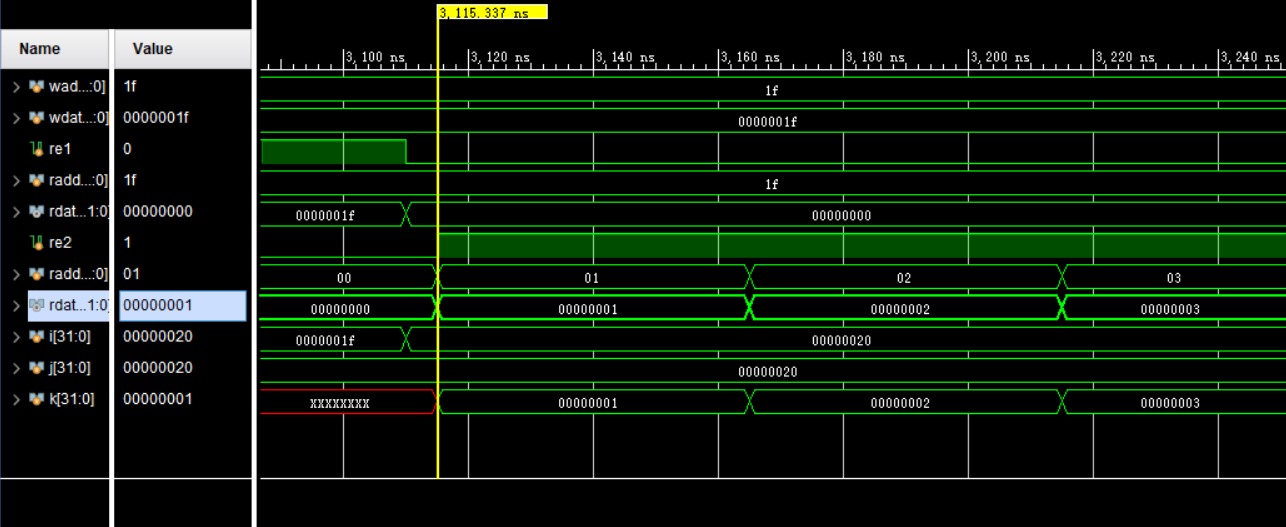
二、寄存器堆



说明：波形图中，刚开始时钟信号clk设置为每隔10 ns翻转一次，且复位信号置1，写使能信号、第一个和第二个端口的读使能信号均置0，同时读取的寄存器地址也都初始化为ZeroWord ，即0。从5 ns开始，复位信号置0，寄存器堆开始工作。we置1，寄存器堆开始写入数据，每隔50 ns，就往第i个（0除外）寄存器地址对应的寄存器写入32位数据 i.（1≤i≤31）。

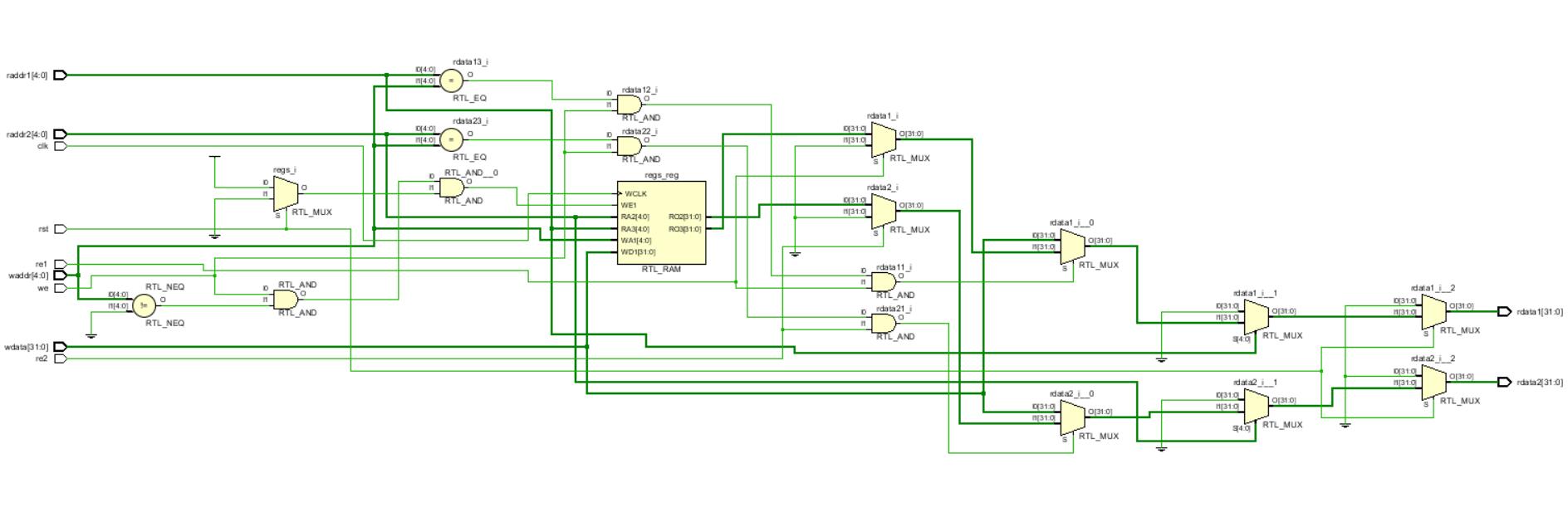


说明：在写端口写入32位数据1f（即31）完毕后，写使能信号we置0，然后第一个读端口开始工作，读使能信号re1置1。寄存器堆开始读出数据，每隔50 ns，就读出第i个（0除外）寄存器地址存储的32位数据 i（1≤i≤31）到输出端rdata1中。



说明：在读端口读出32位数据1f（即31）完毕后，读使能信号re1置0，然后第二个读端口开始工作，读使能信号re2置1。寄存器堆开始继续读出数据，每隔50 ns，就读出第i个（0除外）寄存器地址存储的32位数据 i（1≤i≤31）到输出端rdata2中。

电路图：



1. **实验心得体会**

通过本次实验，本人初步掌握了程序计数器和寄存器堆的原理，并能够根据Verilog语言设计出相关的电路，最后成功输出了预期的波形图。经过本次实验，本人还加深了对程序计数器和寄存器在中央处理器设计使用范围的理解，并提高了Verilog语言的运用能力。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**计 算 机 组 成 原 理 实 验 报 告**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**班级：计科1802 姓名：孔天欣 班级序号：180235 学号：20188068**

**实验日期：2020.11.2**

**学院： 计算机与通信工程学院 专业： 计算机科学与技术**

**实验顺序：31 实验名称：ROM实验 指导教师：张旭**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验目的**

1. 了解只读存储器ROM原理。

2. 理解ROM读取数据的过程。

3. 理解取指过程。

4. 熟悉并运用verilog语言进行电路设计。

5. 为后续设计cpu的实验打下基础。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验环境**

装有 vivado 软件的计算机一台。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验设计图**

一、指令存储器 inst\_rom

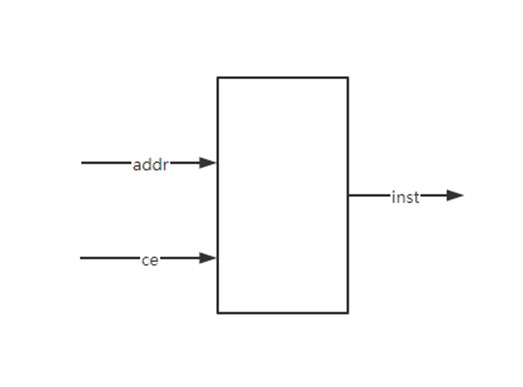


图3.1 指令存储器设计框图

表3.1 inst\_rom模块的接口描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度（bit） | 输入/输出 | 作用 |
| 1 | ce | 1 | 输入 | 使能信号 |
| 2 | addr | 32 | 输入 | 要读取的指令地址 |
| 3 | inst | 32 | 输出 | 读出的指令 |

二、取指模块 inst\_fetch

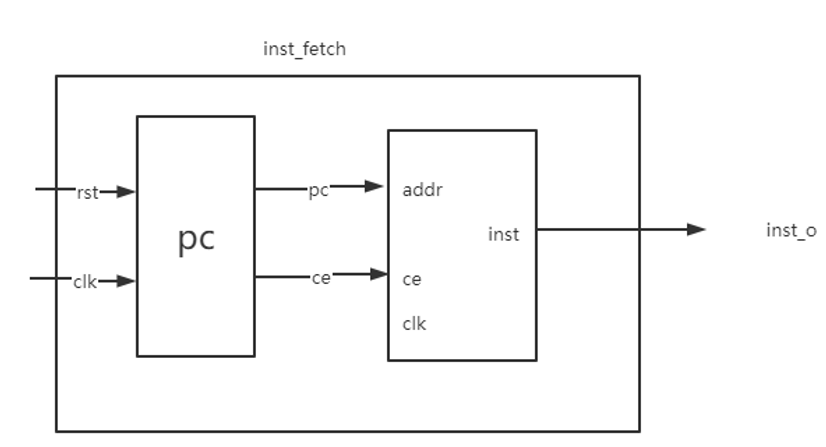


图3.2 取指模块设计框图

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验代码**

一、全局定义模块 define.v

`define RstEnable 1'b1

`define RstDisable 1'b0

`define ZeroWord 32'h00000000

`define WriteEnable 1'b1

`define WriteDisable 1'b0

`define ReadEnable 1'b1

`define ReadDisable 1'b0

`define RegAddrBus 4:0

`define RegBus 31:0

`define RegWidth 32

`define DoubleRegWidth 64

`define DoubleRegBus 63:0

`define RegNum 32

`define RegNumLog2 5

`define NOPRegAddr 5'b0000

`define InstAddrBus 31:0

`define InstBus 31:0

`define InstMemNum 131072

`define InstMemNumLog2 17

`define ChipEnable 1'b1

`define ChipDisable 1'b0

二、指令存储器 inst\_rom

1. 设计模块 inst\_rom.v

`timescale 1ns / 1ns

module inst\_rom(

input wire clk,

input wire ce,

input wire[`InstAddrBus] addr,

output reg[`InstBus] inst

);

reg[`InstBus] inst\_mem[0:`InstMemNum-1];

initial $readmemh ( "D:/inst\_rom.data ",inst\_mem);

always @(\*) begin

if(ce==`ChipDisable) begin

inst <= `ZeroWord;

end else begin

inst <= inst\_mem[addr[`InstMemNumLog2 + 1:2]];

end

end

endmodule

2. 测试模块 inst\_rom\_tb.v

`timescale 1ns / 1ns

module inst\_rom\_tb();

reg clk;

reg ce;

reg[31:0] addr;

wire[31:0] inst;

inst\_rom inst\_rom0(clk,ce,addr,inst);

initial clk = 1;

always #10 clk = ~clk;

integer i;

initial begin

ce <= 0;

#20;

ce <= 1;

**for**(i = 0;i<=40;i = i+1) begin

addr = i;

#20;

**end**

#1000 $finish;

**end**

initial begin

$monitor( " addr = %h,instdata = %h",addr,inst);

**end**

endmodule

三、取指模块

1. 设计模块 inst\_fetch.v

`timescale 1ns / 1ns

module inst\_fetch(

input wire clk,

input wire rst,

output wire[31:0] inst\_o

);

wire[31:0] pc;

wire rom\_ce;

pc\_reg pc0(rst,clk,pc,rom\_ce);

inst\_rom rom0(.ce(rom\_ce),.addr(pc),.inst(inst\_o));

endmodule

2. 测试模块 inst\_fetch\_tb.v

`timescale 1ns / 1ns

module inst\_fetch\_tb();

reg rst;

reg clk;

wire[31:0] inst\_o;

inst\_fetch inst\_fetch0(clk,rst,inst\_o);

initial clk = 1;

always #10 clk = ~clk;

initial begin

rst = 1;

#100 rst = 0;

#1000 $finish;

end

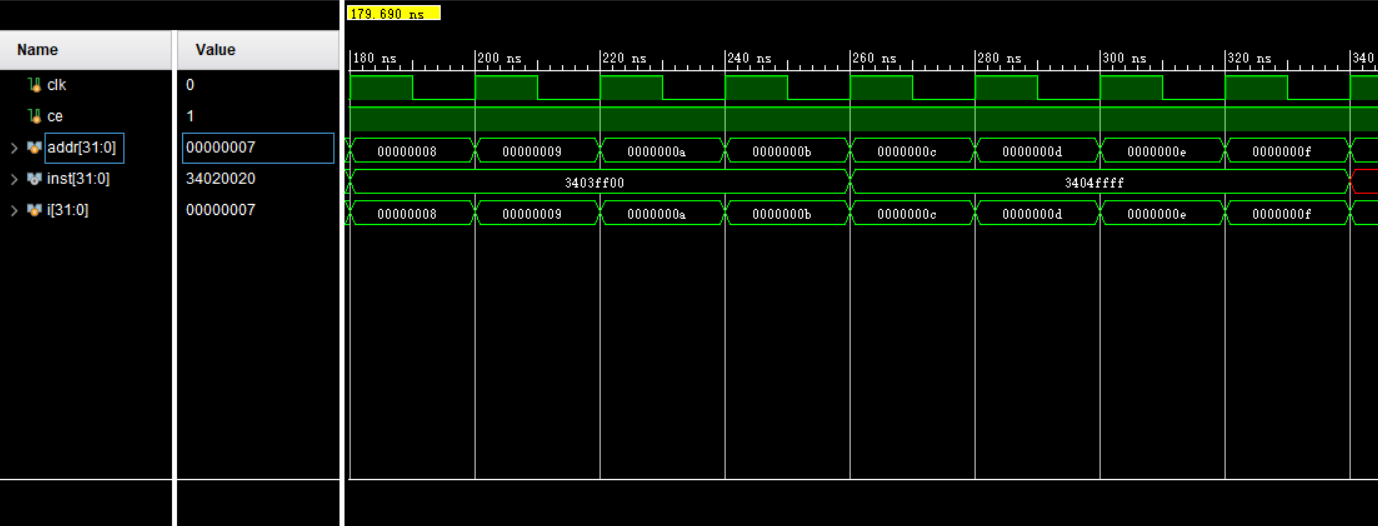
endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**四. 仿真波形及说明**

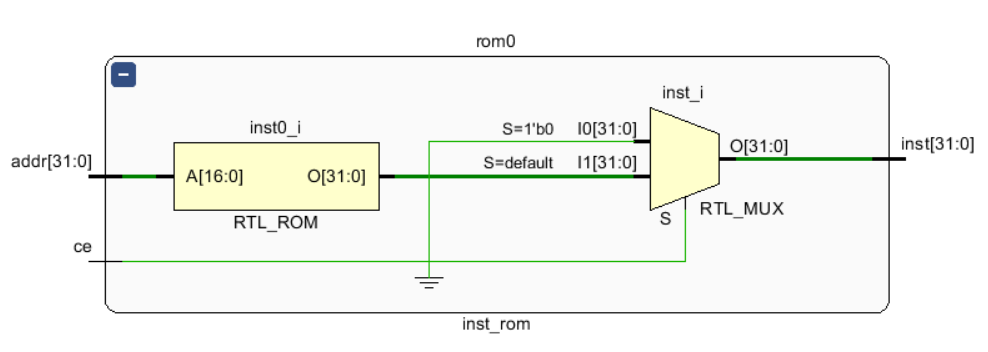
一、指令存储器 ROM



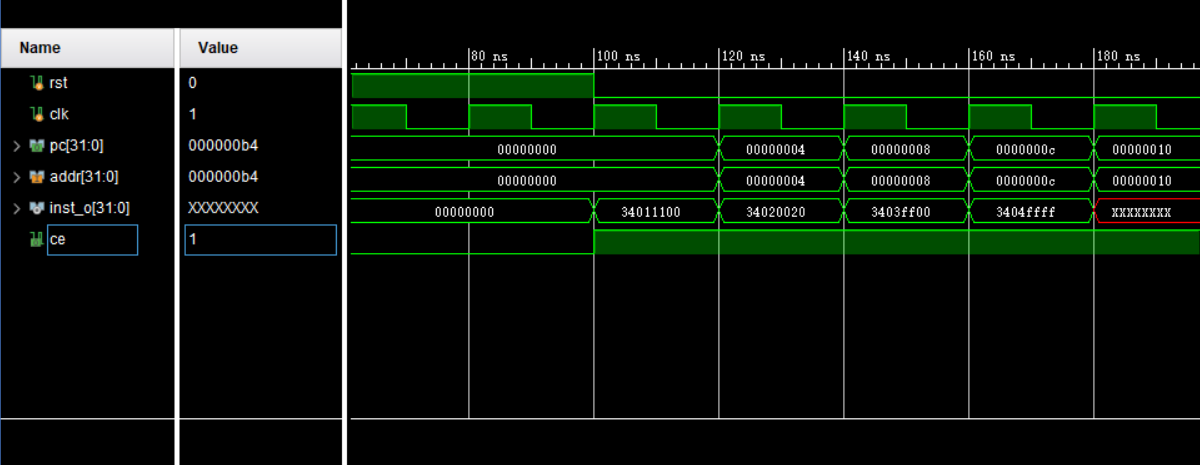


说明：在初始化时，存储器ROM从文件中读取并存储4个32位数据，并使能信号ce置0，20 ns后，ce置1，addr置0，并在每个时钟周期上升沿将addr + 1 (0≤addr＜40)，由于addr的地址在inst\_rom中被右移2位（除以4）才使用，因此图中可见，addr每隔4获取存储器ROM中的数据，且ROM每个元素是32位。又可知，340ns时，addr为10f，inst中读不到数据，之前正好4个元素，说明存储器中的元素已经全部访问完毕。

电路图：

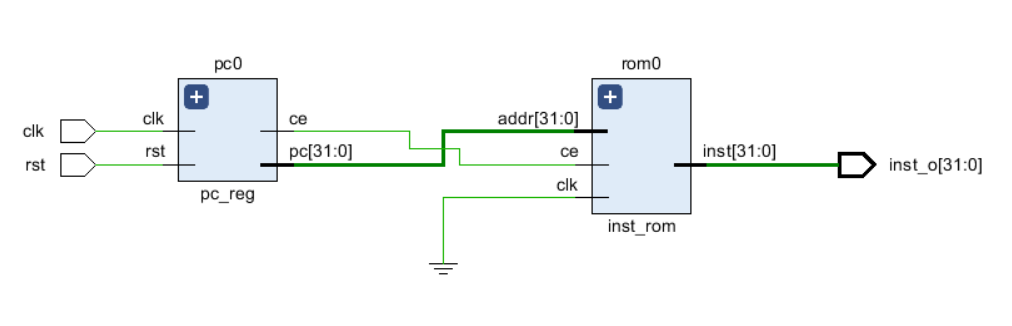


二、取指模块



说明： 图中可见，在前100 ns时，rst复位信号置0，因此ce为0。从100 ns之后开始，rst置1，ce也同时置1，PC计数器和指令存储器ROM均开始工作。PC计数器指令地址addr每增加4，指令存储器ROM就从该指令地址 / 4的的存储器地址中获得一个32位数据并输出inst\_o，并总共访问了4个数据。

电路图：



1. **实验心得体会**

问题思考：addr为什么取[18:2]？

因为PC计数器是按字节寻址，即定义8位一个字，它在每一个时钟周期增加4个字。而指令存储器ROM中定义32位一个字，如果不将指令地址除以4，会导致ROM中每4个字就有3个被跳过。因此需要将PC计数器给出的指令地址除以4，即把指令地址整体右移两位后再使用，因此取[18:2]。18是存储器地址宽度，可以存2的18次个字即131072。

通过本次实验，本人初步掌握了指令存储器ROM和取指模块的原理，并能够根据Verilog语言设计出相关的电路，最后成功输出了预期的波形图。经过本次实验，本人还加深了对指令存储器ROM和PC计数器之间的协作构成取指模块的理解，并再次提高了Verilog语言的运用能力。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**计 算 机 组 成 原 理 实 验 报 告**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**班级：计科1802 姓名：孔天欣 班级序号：180235 学号：20188068**

**实验日期：2020.11.09**

**学院： 计算机与通信工程学院 专业： 计算机科学与技术**

**实验顺序：31 实验名称：RAM实验 指导教师：张旭**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验目的**

1. 了解随机存取存储器RAM的原理。

2. 理解 RAM 读取、写入数据的过程。

3. 理解计算机中存储器地址编址和数据索引方法。

4. 理解同步RAM和异步RAM 的区别。 **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验环境**

装有 vivado 软件的计算机一台。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验设计图**

一、数据存储器 data\_ram

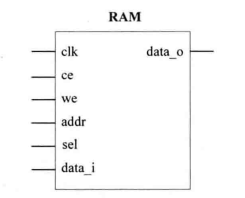


图4.1 数据存储器设计框图

表4.1 inst\_ram模块的接口描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度（bit） | 输入/输出 | 作用 |
| 1 | ce | 1 | 输入 | 使能信号 |
| 2 | clk | 1 | 输入 | 时钟信号 |
| 3 | data\_i | 32 | 输入 | 要写入的数据 |
| 4 | addr | 32 | 输入 | 要读取的地址 |
| 5 | we | 1 | 输入 | 是否是写操作 |
| 6 | sel | 4 | 输入 | 字节选择信号 |
| 7 | data\_o | 32 | 输出 | 读出的数据 |

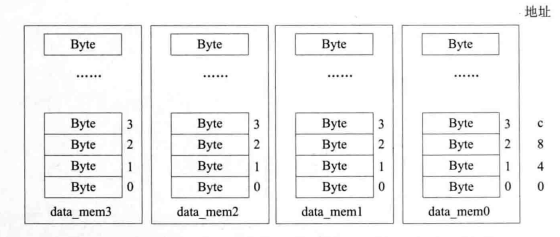


图4.2 32位数据存储器由4个8位数据存储器构成

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验代码**

一、全局定义模块 define.v

`define **ChipEnable** 1**'**b1

`define **ChipDisable** 1**'**b0

`define **ZeroWord** 32**'**h00000000

`define **WriteEnable** 1**'**b1

`define **WriteDisable** 1**'**b0

`define **DataAddrBus** 31:0

`define **DataBus** 31:0

`define **DataMemNum** 131071

`define **DataMemNumLog2** 17

`define **ByteWidth** 7:0

二、数据存储器 data\_ram（同步写，异步读）

1. 设计模块 data\_ram.v

`timescale 1ns / 1ns

**module** data\_ram(

**input** wire clk,

**input** wire ce,

**input** wire we,

**input** wire[`DataAddrBus] addr,

**input** wire[3:0] sel,

**input** wire[`DataBus] data\_i,

**output** reg[`DataBus] data\_o

);

reg[`ByteWidth] data\_mem0[0:`DataMemNum-1];

reg[`ByteWidth] data\_mem1[0:`DataMemNum-1];

reg[`ByteWidth] data\_mem2[0:`DataMemNum-1];

reg[`ByteWidth] data\_mem3[0:`DataMemNum-1];

**always** @ (**posedge** clk) **begin**

**if**(ce == `ChipDisable) **begin**

data\_o <= `ZeroWord;

**end**

**else** **if**(we == `WriteEnable) **begin**

**if**(sel[3] ==1'b1) **begin**

data\_mem3[addr[`DataMemNumLog2+1:2]] <= data\_i[31:24];

**end**

**if**(sel[2] == 1'b1) **begin**

data\_mem2[addr[`DataMemNumLog2+1:2]] <= data\_i[23:16];

**end**

**if**(sel[1] == 1'b1) **begin**

data\_mem1[addr[`DataMemNumLog2+1:2]] <= data\_i[15:8];

**end**

**if**(sel[0] == 1'b1) **begin**

data\_mem0[addr[`DataMemNumLog2+1:2]] <= data\_i[7:0];

**end**

**end**

**end**

**always** @ (\*) **begin**

**if**(ce == `ChipDisable) **begin**

data\_o <= `ZeroWord;

**end**

**else** **if** (we == `WriteDisable) **begin**

data\_o <= {data\_mem3[addr[`DataMemNumLog2+1:2]],

data\_mem2[addr[`DataMemNumLog2+1:2]],

data\_mem1[addr[`DataMemNumLog2+1:2]],

data\_mem0[addr[`DataMemNumLog2+1:2]]};

**end**

**else** **begin**

data\_o <= `ZeroWord;

**end**

**end**

**endmodule**

2. 测试模块 data\_ram\_tb.v

`timescale 1ns / 1ps

**module** data\_ram\_tb();

reg clk;

reg ce;

reg we;

reg[31:0] addr;

reg[3:0] sel;

reg[31:0] data\_i;

wire[31:0] data\_o;

data\_ram data\_ram0(clk,ce,we,addr,sel,data\_i,data\_o);

**integer** i,j,k;

**initial** clk = 1;

**always** #10 clk = ~clk;

**initial** **begin**

ce = 0;

we = 0;

sel = 4'b0001;

data\_i = 32'hfedcba98;

addr = 0;

#100 ce = 1;

we = 1;

**for**(j = 0;j<10;j = j+1) **begin**

sel = 4'b0001;

**for**(i = 0;i<4;i = i+1) **begin**

#40

addr = addr + 1;

sel = sel<<1;

data\_i = data\_i -32'h01010101;

**end**

**end**

#30;

we = 0;

**for**(k = 0;k<40;k = k+1) **begin**

addr = k;

#20;

**end**

**end**

**endmodule**

三、数据存储器 data\_ram（同步写，同步读）

1. 设计模块 data\_ram.v

`timescale 1ns / 1ps

**module** data\_ram(

**input** wire clk,

**input** wire ce,

**input** wire we,

**input** wire[`DataAddrBus] addr,

**input** wire[3:0] sel,

**input** wire[`DataBus] data\_i,

**output** reg[`DataBus] data\_o

);

reg[`ByteWidth] data\_mem0[0:`DataMemNum-1];

reg[`ByteWidth] data\_mem1[0:`DataMemNum-1];

reg[`ByteWidth] data\_mem2[0:`DataMemNum-1];

reg[`ByteWidth] data\_mem3[0:`DataMemNum-1];

**always** @ (**posedge** clk) **begin**

**if**(ce == `ChipDisable) **begin**

data\_o <= `ZeroWord;

**end**

**else** **if**(we == `WriteEnable) **begin**

**if**(sel[3] ==1'b1) **begin**

data\_mem3[addr[`DataMemNumLog2+1:2]] <= data\_i[31:24];

**end**

**if**(sel[2] == 1'b1) **begin**

data\_mem2[addr[`DataMemNumLog2+1:2]] <= data\_i[23:16];

**end**

**if**(sel[1] == 1'b1) **begin**

data\_mem1[addr[`DataMemNumLog2+1:2]] <= data\_i[15:8];

**end**

**if**(sel[0] == 1'b1) **begin**

data\_mem0[addr[`DataMemNumLog2+1:2]] <= data\_i[7:0];

**end**

**end**

**end**

**always** @ (**posedge** clk) **begin**

**if**(ce == `ChipDisable) **begin**

data\_o <= `ZeroWord;

**end**

**else** **if** (we == `WriteDisable) **begin**

data\_o <= {data\_mem3[addr[`DataMemNumLog2+1:2]],

data\_mem2[addr[`DataMemNumLog2+1:2]],

data\_mem1[addr[`DataMemNumLog2+1:2]],

data\_mem0[addr[`DataMemNumLog2+1:2]]};

**end**

**else** **begin**

data\_o <= `ZeroWord;

**end**

**end**

**endmodule**

2. 测试模块 data\_ram\_tb.v

`timescale 1ns / 1ns

**module** data\_ram\_tb();

reg clk;

reg ce;

reg we;

reg[31:0] addr;

reg[3:0] sel;

reg[31:0] data\_i;

wire[31:0] data\_o;

data\_ram data\_ram0(clk,ce,we,addr,sel,data\_i,data\_o);

**integer** i,j,k;

**initial** clk = 1;

**always** #10 clk = ~clk;

**initial** **begin**

ce = 0;

we = 0;

sel = 4'b0001;

data\_i = 32'hfedcba98;

addr = 0;

#100 ce = 1;

we = 1;

**for**(j = 0;j<10;j = j+1) **begin**

sel = 4'b0001;

**for**(i = 0;i<4;i = i+1) **begin**

#40

addr = addr + 1;

sel = sel<<1;

data\_i = data\_i -32'h01010101;

**end**

**end**

#30;

we = 0;

**for**(k = 0;k<40;k = k+1) **begin**

addr = k;

#20;

**end**

**end**

**endmodule**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**四. 仿真波形及说明**

一、数据存储器 RAM（同步写，异步读）

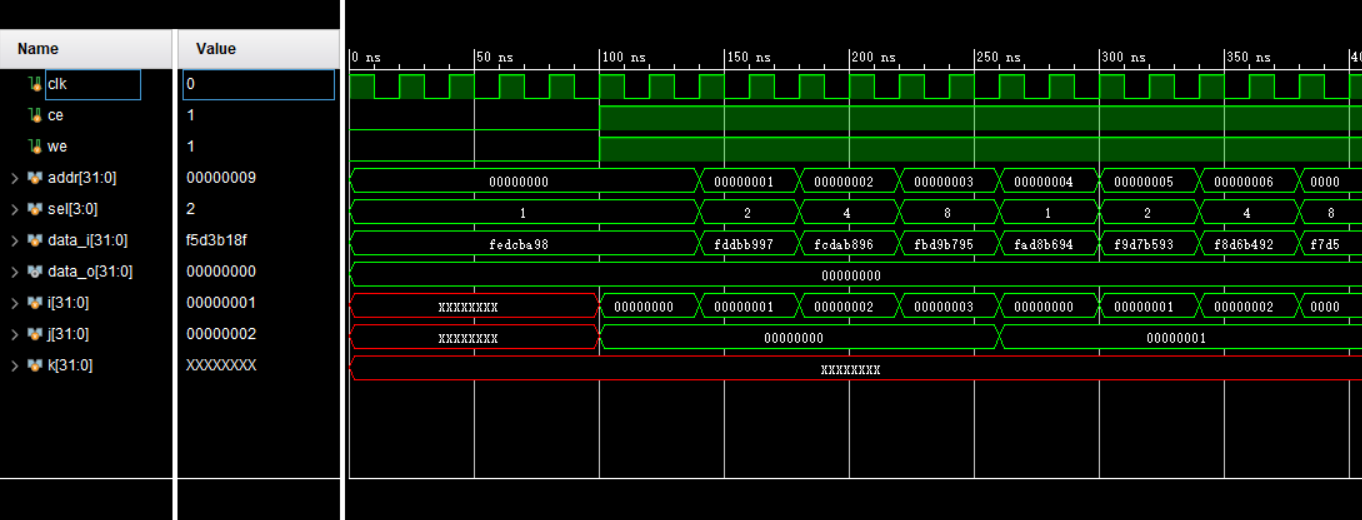


图4.3 RAM写波形图

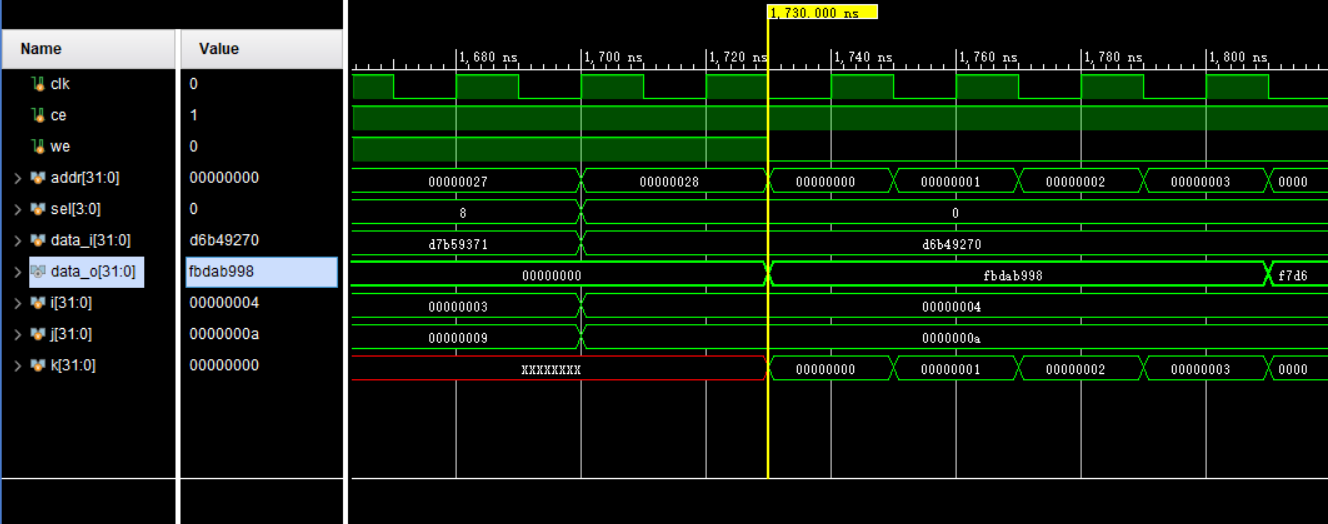


图4.4 RAM读波形图

说明：在测试代码中，RAM写入完毕后的等待时间由40 ns改成30 ns，这样可以更好地看出同步读和异步读的区别。由图4.3可知，时钟clk每隔10 ns翻转一次，同时使能信号ce和写信号we初始化为0，字节选择信号sel初始化为1，d要写入的数据data\_i则初始化为32位数据hfedcda98，写入的地址addr初始化为0。从100 ns开始，ce和we置1，开始写数据操作：首先从地址0开始依次递增，并向第一个存储器的第一个字节写入数据data\_i，然后左移一次sel以更换下一个存储器，接着向第二个存储器的第一个字节写入数据，总共移动3次后切换回第一个存储器，并从它的第二个字节继续写入数据（因为32位存储器由4个8位存储器构成，一次存取32位），依次类推完成写入400个地址操作。

由图4.4可知，从1730 ns开始读取数据操作，从地址0开始依次读取之前写入的部分数据，可以看出，读取的数据和之前写入的数据一致。由于是异步读的方式，可以看到，开始读的时候时钟并没有处在上升沿，所以并不需要等待到上升沿才能读。

电路图：

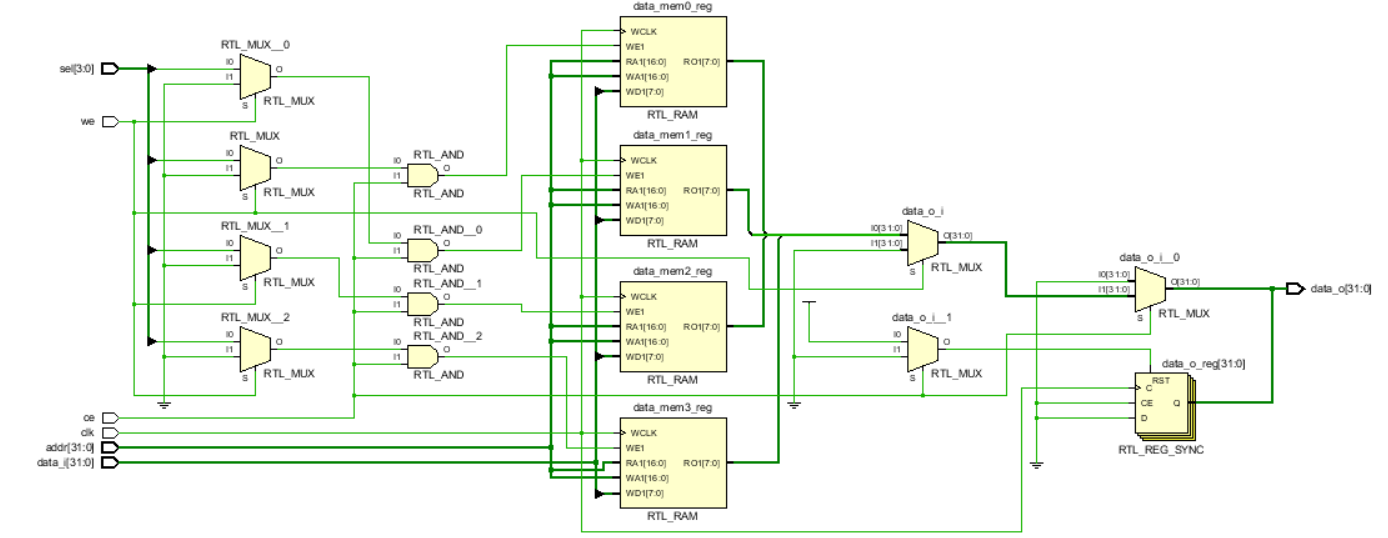


图4.5 同步写异步读RAM电路图

二、数据存储器 RAM（同步写，同步读）

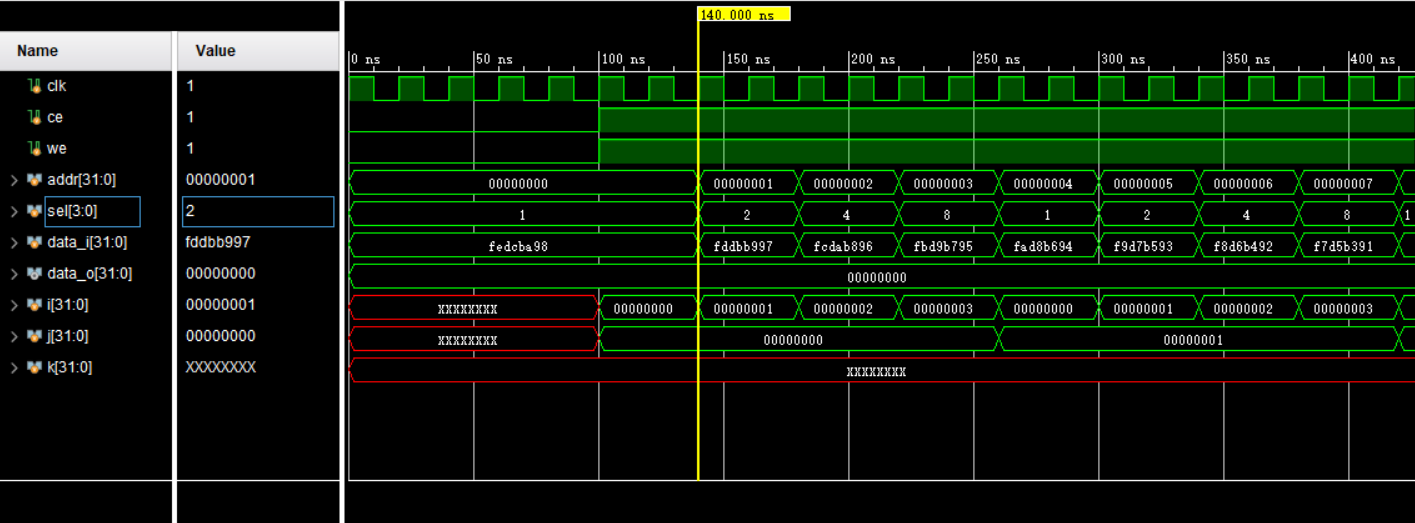


图4.6 RAM写波形图

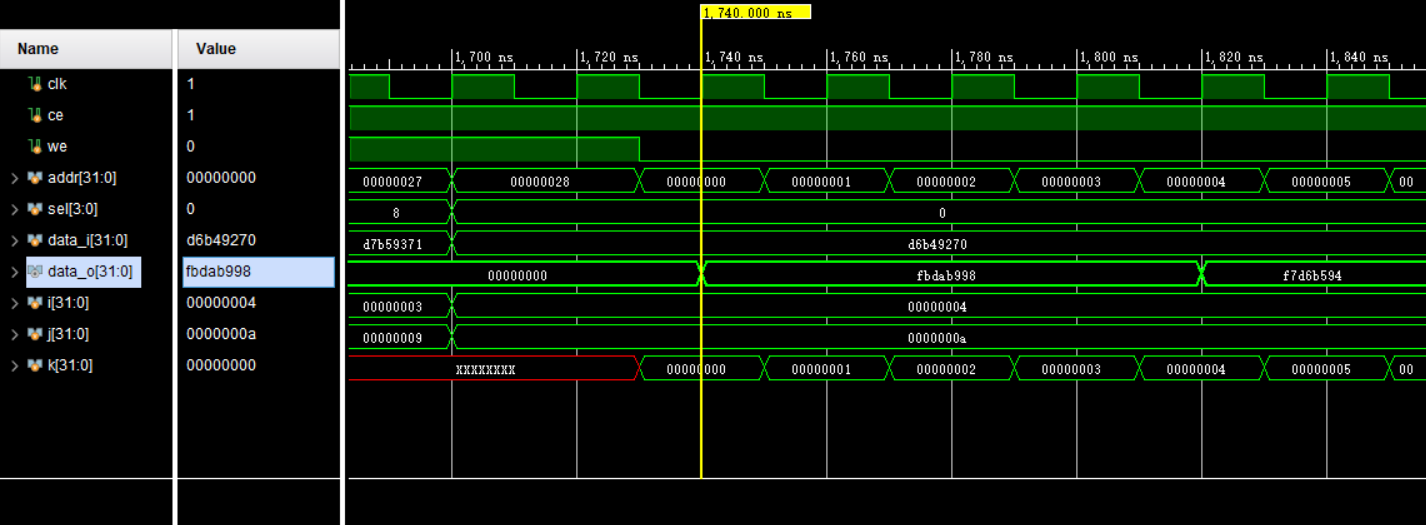


图4.7 RAM读波形图

说明：在写入数据的部分中，和前文是一致的。但在读数据的过程中，可以看到在1730 ns时，addr变为0，但是data\_o的数据并没有读出来，直到1740 ns 时，时钟clk处于上升沿的时候，这时才读到了数据 fbda998，因此可以得出结论：这是同步读的方式，此后的读取数据都是在时钟处于上升沿时才能读，读取的数据内容和前文一致。

电路图：

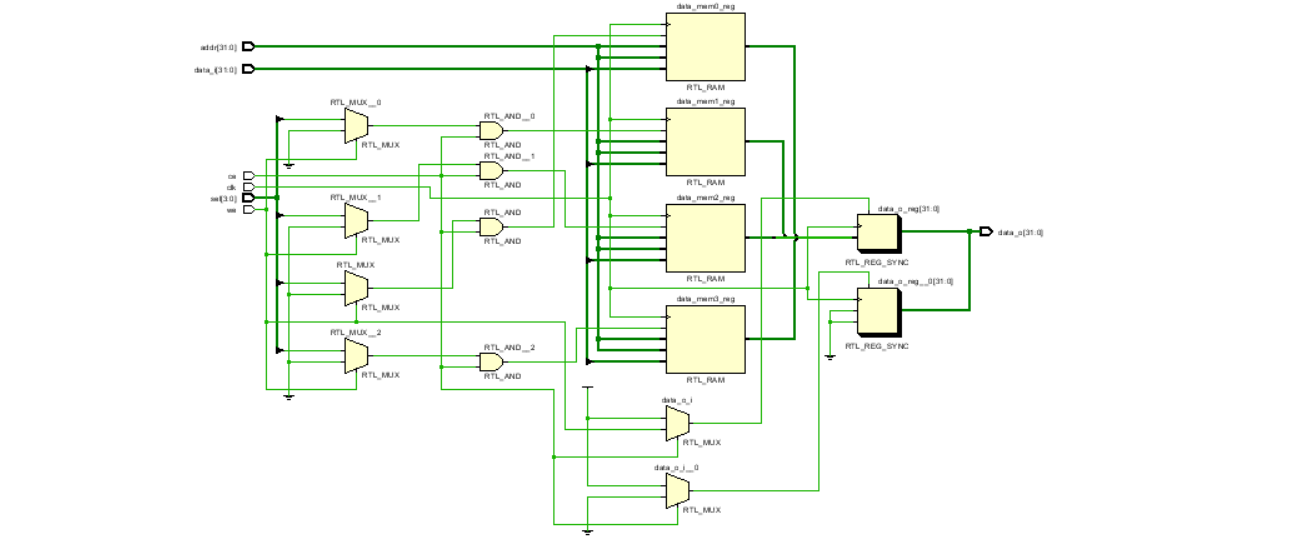


图4.8 同步写同步读RAM电路图

1. **实验心得体会**

问题思考：

1. 在本次实验 addr为什么取[18:2]？

因为RAM一次存取4个字节，而系统设计按字节寻址，因此需要将实际地址除以4才能转换为RAM地址，故舍去低2位以达到除以4的效果。同时RAM总共能存储2的17次个字的数据，故取[18:2]。

2. 画出data\_mem0，data\_mem1，data\_mem2,data\_mem3这4个存储区域的地址引脚和数据引脚往系统数据总线，地址总线上的连接示意图。

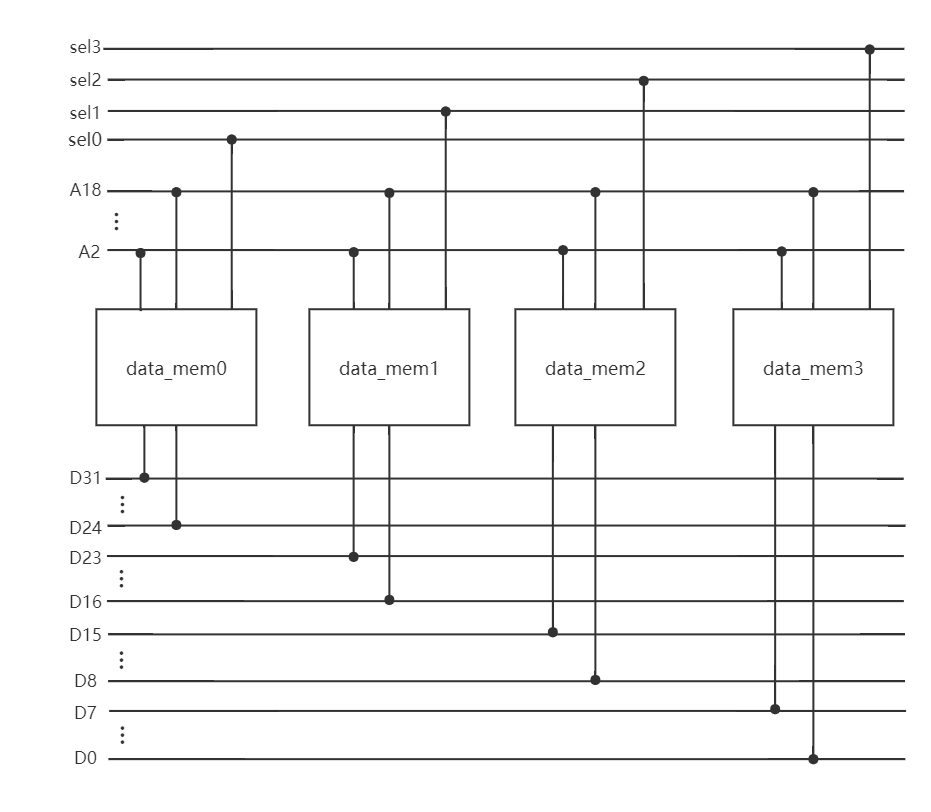


图4.9 连接示意图

心得体会：

通过本次实验，本人初步掌握了数据存储器RAM原理，并能够根据Verilog语言设计出相关的电路，最后成功输出了预期的波形图。经过本次实验，本人还加深了对数据存储器RAM和组合方法和同步、异步读写的理解，并再次提高了计算机组成原理相关知识的运用能力。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**计 算 机 组 成 原 理 实 验 报 告**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**班级：计科1802 姓名：孔天欣 班级序号：180235 学号：20188068**

**实验日期：2020.11.16**

**学院： 计算机与通信工程学院 专业： 计算机科学与技术**

**实验顺序：31 实验名称：ALU实验 指导教师：张旭**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验目的**

1.了解MIPS指令集中的运算指令，学会对这些指令进行归纳分类。

2.熟悉并掌握 ALU 的原理、功能和设计。

3.进一步加强运用 verilog 语言进行电路设计的能力。

4.为后续设计 cpu 的实验打下基础。**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验环境**

装有 vivado 软件的计算机一台。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验设计图**

一、运算器 ALU

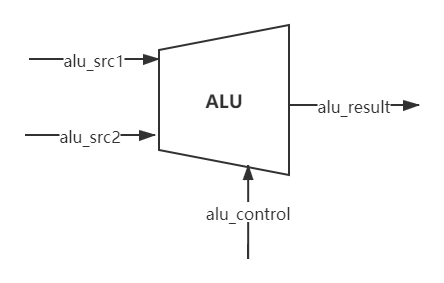


图5.1 运算器ALU设计框图

表5.1 ALU模块的接口描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度（bit） | 输入/输出 | 作用 |
| 1 | alu\_control | 12 | 输入 | 控制信号 |
| 2 | alu\_src1 | 32 | 输入 | 操作数1，补码 |
| 3 | alu\_src2 | 32 | 输入 | 操作数2，补码 |
| 4 | alu\_result | 32 | 输出 | 运算结果 |

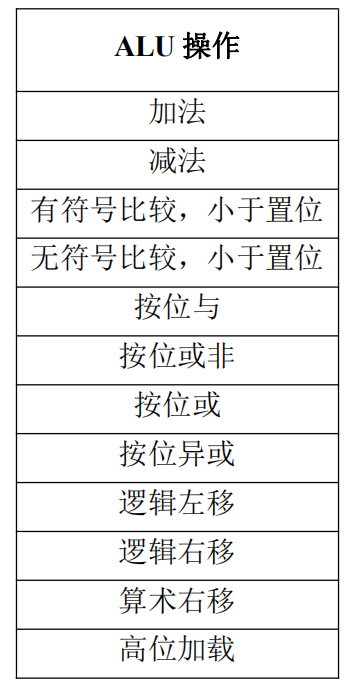


图5.2 ALU运算器实现框图

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验代码**

一、全局定义模块 define.v

// 加法

`define **ADD\_OP** 4**'**b0000

// 减法

`define **SUB\_OP** 4**'**b0001

// 有符号比较，a小于b置1

`define **SLT\_OP** 4**'**b0010

// 无符号比较，a小于b置1

`define **SLTU\_OP** 4**'**b0011

`define **AND\_OP** 4**'**b0100

`define **NOR\_OP** 4**'**b0101

`define **OR\_OP** 4**'**b0110

`define **XOR\_OP** 4**'**b0111

// 逻辑左移

`define **SLL\_OP** 4**'**b1000

// 逻辑右移

`define **SRL\_OP** 4**'**b1001

// 算数右移

`define **SRA\_OP** 4**'**b1010

// 高位加载

`define **LUI\_OP** 4'b1011

二、运算器 ALU（编码）

1. 设计模块 alu.v

`timescale 1ns / 1ps

**module** alu(

**input**[3:0] alu\_control,

**input**[31:0] alu\_src1,

**input**[31:0] alu\_src2,

**output** reg[31:0] alu\_result

);

wire[31:0] alu\_src2\_mux;

wire[31:0] result\_sum;

**assign** alu\_src2\_mux = (alu\_control==`SUB\_OP||alu\_control==`SLT\_OP)?(~alu\_src2)+1:alu\_src2;

**assign** result\_sum = alu\_src1+alu\_src2\_mux;

*// 比较结果*

**assign** src1\_lt\_src2 = ((alu\_control==`SLT\_OP))?

((alu\_src1[31]&&!alu\_src2[31])||

(!alu\_src1[31]&&!alu\_src2[31]&&result\_sum[31])||

(alu\_src1[31]&&alu\_src2[31]&&result\_sum[31])):(alu\_src1<alu\_src2);

**always** @(\*) **begin**

**case**(alu\_control)

`ADD\_OP,`SUB\_OP:**begin**

alu\_result = result\_sum;

**end**

`SLT\_OP,`SLTU\_OP:**begin**

alu\_result = src1\_lt\_src2;

**end**

`AND\_OP:**begin**

alu\_result = alu\_src1 & alu\_src2;

**end**

`NOR\_OP:**begin**

alu\_result = ~(alu\_src1|alu\_src2);

**end**

`OR\_OP:**begin**

alu\_result = alu\_src1 | alu\_src2;

**end**

`XOR\_OP:**begin**

alu\_result = alu\_src1 ^ alu\_src2;

**end**

`SLL\_OP:**begin**

alu\_result = alu\_src2 << alu\_src1[4:0];

**end**

`SRL\_OP:**begin**

alu\_result = alu\_src2 >> alu\_src1[4:0];

**end**

`SRA\_OP:**begin**

alu\_result = ({32{alu\_src2[31]}} << (6'd32-{1'b0,alu\_src1[4:0]}))

| alu\_src2 >> alu\_src1[4:0];

**end**

`LUI\_OP:**begin**

alu\_result = {alu\_src2[15:0],16'd0};

**end**

**default**:**begin**

alu\_result = 32'b0;

**end**

**endcase**

**end**

**endmodule**

2. 测试模块 alu\_tb.v

`timescale 1ns / 1ps

**module** alu\_tb();

reg[3:0] alu\_control;

reg[31:0] alu\_src1;

reg[31:0] alu\_src2;

wire[31:0] alu\_result;

**integer** i;

alu alu1(alu\_control,alu\_src1,alu\_src2,alu\_result);

**initial** **begin**

alu\_control = 4'b0000;

alu\_src1 = 32'h1257\_89Ab;

alu\_src2 = 32'hFEAB\_BC76;

#20;

**for**(i = 0;i<12;i = i+1) **begin**

$monitor("alusrc1 = %h , alu\_control = %b , alu\_src2 = %h , alu\_result = %h",

alu\_src1,alu\_control,alu\_src2,alu\_result);

#20;

alu\_control = alu\_control + 1;

**end**

#40 $finish;

**end**

**endmodule**

三、运算器 ALU（独热码）

1. 设计模块 alu.v

`timescale 1ns / 1ps

**module** alu(

**input** [11:0] alu\_control,

**input** [31:0] alu\_src1,

**input** [31:0] alu\_src2,

**output** [31:0] alu\_result

);

wire alu\_add;

wire alu\_sub;

wire alu\_sltu;

wire alu\_and;

wire alu\_nor;

wire alu\_or;

wire alu\_xor;

wire alu\_sll;

wire alu\_srl;

wire alu\_sra;

wire alu\_luo;

**assign** alu\_add = alu\_control[11];

**assign** alu\_sub = alu\_control[10];

**assign** alu\_slt = alu\_control[9];

**assign** alu\_sltu = alu\_control[8];

**assign** alu\_and = alu\_control[7];

**assign** alu\_nor = alu\_control[6];

**assign** alu\_or = alu\_control[5];

**assign** alu\_xor = alu\_control[4];

**assign** alu\_sll = alu\_control[3];

**assign** alu\_srl = alu\_control[2];

**assign** alu\_sra = alu\_control[1];

**assign** alu\_lui = alu\_control[0];

wire[31:0] add\_sub\_result;

wire[31:0] slt\_result;

wire[31:0] sltu\_result;

wire[31:0] and\_result;

wire[31:0] nor\_result;

wire[31:0] or\_result;

wire[31:0] xor\_result;

wire[31:0] sll\_result;

wire[31:0] srl\_result;

wire[31:0] sra\_result;

wire[31:0] lui\_result;

**assign** and\_result = alu\_src1 & alu\_src2;

**assign** or\_result = alu\_src1 | alu\_src2;

**assign** nor\_result = ~or\_result;

**assign** xor\_result = alu\_src1 ^ alu\_src2;

**assign** lui\_result = {alu\_src2[15:0],16'd0};

wire[31:0] adder\_operand1;

wire[31:0] adder\_operand2;

wire adder\_cin;

wire[31:0] adder\_result;

wire adder\_cout;

**assign** adder\_operand1 = alu\_src1;

**assign** adder\_operand2 = alu\_add ? alu\_src2:~alu\_src2;

**assign** adder\_cin = ~alu\_add;

adder adder\_module(

adder\_operand1,

adder\_operand2,

adder\_cin,

adder\_result,

adder\_cout);

**assign** add\_sub\_result = adder\_result;

**assign** slt\_result[31:1] = 31'd0;

**assign** slt\_result[0] = (alu\_src1[31] & ~alu\_src2[31])

| (~(alu\_src1[31] ^ alu\_src2[31]) & adder\_result[31]);

**assign** sltu\_result = {31'd0,~adder\_cout};

wire[4:0] shf;

**assign** shf = alu\_src1[4:0];

wire[1:0] shf\_1\_0;

wire[1:0] shf\_3\_2;

**assign** shf\_1\_0 = shf[1:0];

**assign** shf\_3\_2 = shf[3:2];

wire[31:0] sll\_step1;

wire[31:0] sll\_step2;

**assign** sll\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2

| {32{shf\_1\_0 == 2'b01}} & {alu\_src2[30:0],1'd0}

| {32{shf\_1\_0 == 2'b10}} & {alu\_src2[29:0],2'd0}

| {32{shf\_1\_0 == 2'b11}} & {alu\_src2[28:0],3'd0};

**assign** sll\_step2 = {32{shf\_3\_2 == 2'b00}} & sll\_step1

| {32{shf\_3\_2 == 2'b01}} & {sll\_step1[27:0],4'd0}

| {32{shf\_3\_2 == 2'b10}} & {sll\_step1[23:0],8'd0}

| {32{shf\_3\_2 == 2'b11}} & {sll\_step1[19:0],12'd0};

**assign** sll\_result = shf[4] ? {sll\_step2[15:0],16'd0}:sll\_step2;

wire [31:0] srl\_step1;

wire [31:0] srl\_step2;

**assign** srl\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2

| {32{shf\_1\_0 == 2'b01}} & {1'd0,alu\_src2[31:1]}

| {32{shf\_1\_0 == 2'b10}} & {2'd0,alu\_src2[31:2]}

| {32{shf\_1\_0 == 2'b11}} & {3'd0,alu\_src2[31:3]};

**assign** srl\_step2 = {32{shf\_3\_2 == 2'b00}} & srl\_step1

| {32{shf\_3\_2 == 2'b01}} & {4'd0,srl\_step1[31:4]}

| {32{shf\_3\_2 == 2'b10}} & {8'd0,srl\_step1[31:8]}

| {32{shf\_3\_2 == 2'b11}} & {12'd0,srl\_step1[31:12]};

**assign** srl\_result = shf[4] ? {16'd0,srl\_step2[31:16]}:srl\_step2;

wire [31:0] sra\_step1;

wire [31:0] sra\_step2;

**assign** sra\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2

| {32{shf\_1\_0 == 2'b01}} & {alu\_src2[31],alu\_src2[31:1]}

| {32{shf\_1\_0 == 2'b10}} & {{2{alu\_src2[31]}},alu\_src2[31:2]}

| {32{shf\_1\_0 == 2'b11}} & {{3{alu\_src2[31]}},alu\_src2[31:3]};

**assign** sra\_step2 = {32{shf\_3\_2 == 2'b00}} & sra\_step1

| {32{shf\_3\_2 == 2'b01}} & {{4{sra\_step1[31]}},sra\_step1[31:4]}

| {32{shf\_3\_2 == 2'b10}} & {{8{sra\_step1[31]}},sra\_step1[31:8]}

| {32{shf\_3\_2 == 2'b11}} & {{12{sra\_step1[31]}},sra\_step1[31:12]};

**assign** sra\_result = shf[4] ? {{16{sra\_step2[31]}},sra\_step2[31:16]}:sra\_step2;

**assign** alu\_result = (alu\_add|alu\_sub)?add\_sub\_result[31:0]:

alu\_slt?slt\_result:

alu\_sltu?sltu\_result:

alu\_and?and\_result:

alu\_nor?nor\_result:

alu\_or?or\_result:

alu\_xor?xor\_result:

alu\_sll?sll\_result:

alu\_srl?srl\_result:

alu\_sra?sra\_result:

alu\_lui?lui\_result:

32'd0;

**endmodule**

2. 设计模块 adder.v

`timescale 1ns / 1ps

**module** adder(

**input**[31:0] operand1,

**input**[31:0] operand2,

**input** cin,

**output**[31:0] result,

**output** cout

);

**assign** {cout,result} = operand1 + operand2 + cin;

**endmodule**

3. 测试模块 alu\_tb.v

`timescale 1ns / 1ps

**module** alu\_tb();

reg[11:0] alu\_control;

reg[31:0] alu\_src1;

reg[31:0] alu\_src2;

wire[31:0] alu\_result;

**integer** i;

alu alu2(alu\_control,alu\_src1,alu\_src2,alu\_result);

**initial** **begin**

alu\_control = 11'b0000\_0000\_0001;

alu\_src1 = 32'h1257\_89Ab;

alu\_src2 = 32'hFEAB\_BC76;

#20;

**for**(i = 0;i<12;i = i+1) **begin**

$monitor("alusrc = %h,alu\_control = %b,alu\_src = %h,alu\_result = %b"

,alu\_src1,alu\_control,alu\_src2,alu\_result);

#20;

alu\_control = alu\_control << 1;

**end**

#40;

$finish;

**end**

**endmodule**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**四. 仿真波形及说明**

一、运算器 ALU（编码）

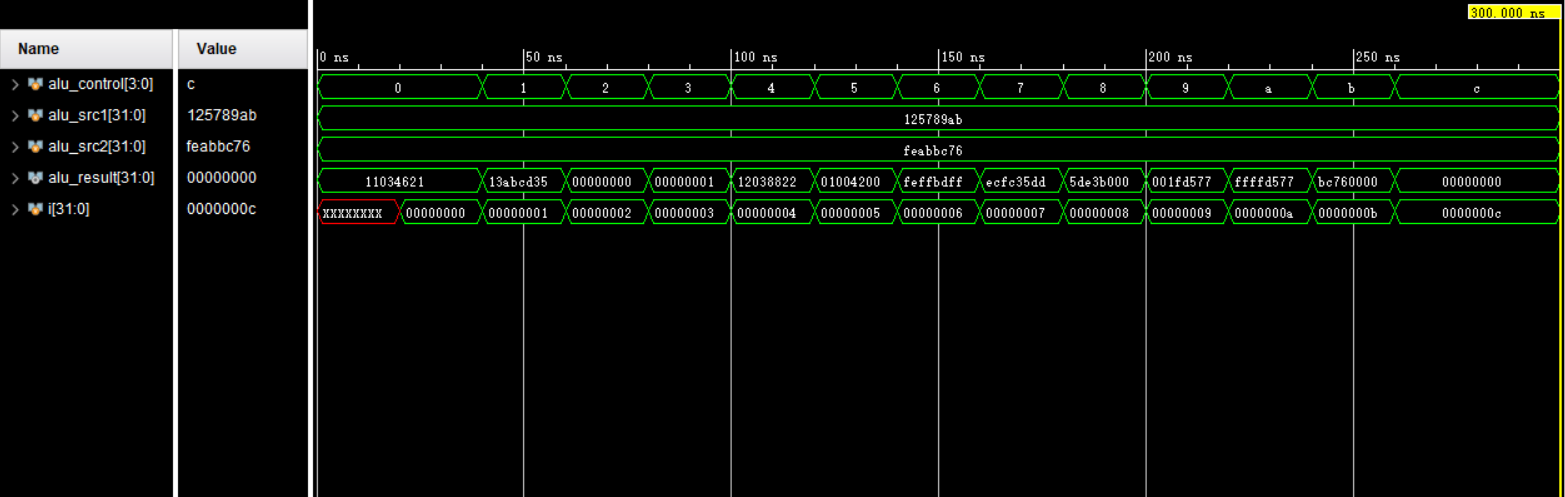


图4.3 运算器ALU波形图

说明： 这段波形图的输出可以对应下面的输出清单：

alusrc1 = 125789ab , alu\_control = 0000 , alu\_src2 = feabbc76 , alu\_result = 11034621

alusrc1 = 125789ab , alu\_control = 0001 , alu\_src2 = feabbc76 , alu\_result = 13abcd35

alusrc1 = 125789ab , alu\_control = 0010 , alu\_src2 = feabbc76 , alu\_result = 00000000

alusrc1 = 125789ab , alu\_control = 0011 , alu\_src2 = feabbc76 , alu\_result = 00000001

alusrc1 = 125789ab , alu\_control = 0100 , alu\_src2 = feabbc76 , alu\_result = 12038822

alusrc1 = 125789ab , alu\_control = 0101 , alu\_src2 = feabbc76 , alu\_result = 01004200

alusrc1 = 125789ab , alu\_control = 0110 , alu\_src2 = feabbc76 , alu\_result = feffbdff

alusrc1 = 125789ab , alu\_control = 0111 , alu\_src2 = feabbc76 , alu\_result = ecfc35dd

alusrc1 = 125789ab , alu\_control = 1000 , alu\_src2 = feabbc76 , alu\_result = 5de3b000

alusrc1 = 125789ab , alu\_control = 1001 , alu\_src2 = feabbc76 , alu\_result = 001fd577

alusrc1 = 125789ab , alu\_control = 1010 , alu\_src2 = feabbc76 , alu\_result = ffffd577

alusrc1 = 125789ab , alu\_control = 1011 , alu\_src2 = feabbc76 , alu\_result = bc760000

可以看到，每隔20 ns，通过改变alu\_control的数据，就会改变ALU的运算方式，但两个源操作数alu\_src1和alu\_src2都是不变的。从0 ns开始，两个操作数（补码形式）先完成了加法运算（ADD\_OP），然后输出到alu\_result，接着从40 ns开始，完成了减法运算（SUB\_OP），此后每隔 20 ns，依次完成了有符号比较（小于置位，SLT\_OP）、无符号比较（小于置位，SLTU\_OP）、按位与（AND\_OP）、按位或非（NOR\_OP）、按位或（OR\_OP）、按位异或（XOR\_OP）、逻辑左移（SLL\_OP）、逻辑右移（SRL\_OP）、算数右移（SRA\_OP)、高位加载（LUI\_OP）的运算，具体运算结果输出可以分别对应上述清单中的alu\_control值。

电路图：

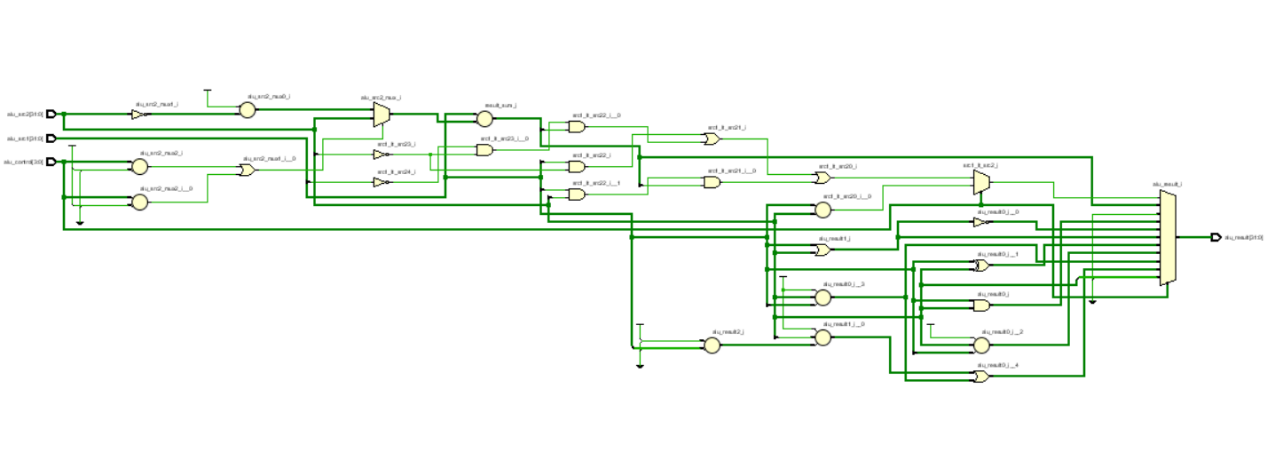


图4.4 运算器ALU（编码）电路图

二、运算器ALU（独热码）

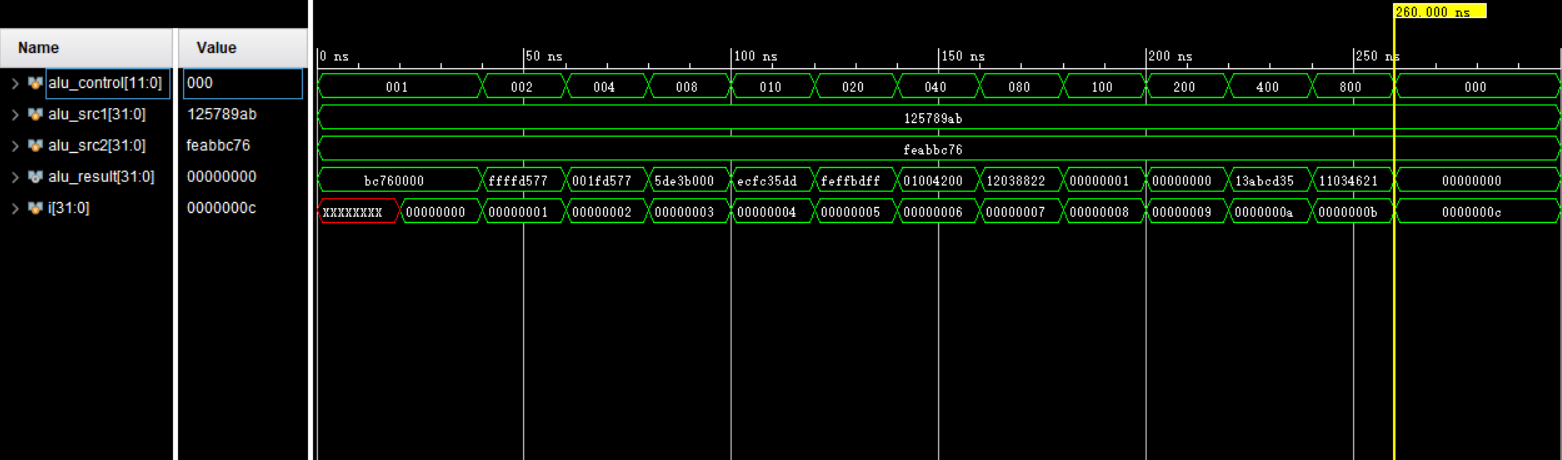


图4.5 运算器ALU（独热码）波形图

对于该波形图，运算结果的输出清单如下：



可以看出，对于相同的源操作数输入，对应运算结果的输出和编码形式也是一致的，因此也能够正确实现基本的运算功能。

电路图：

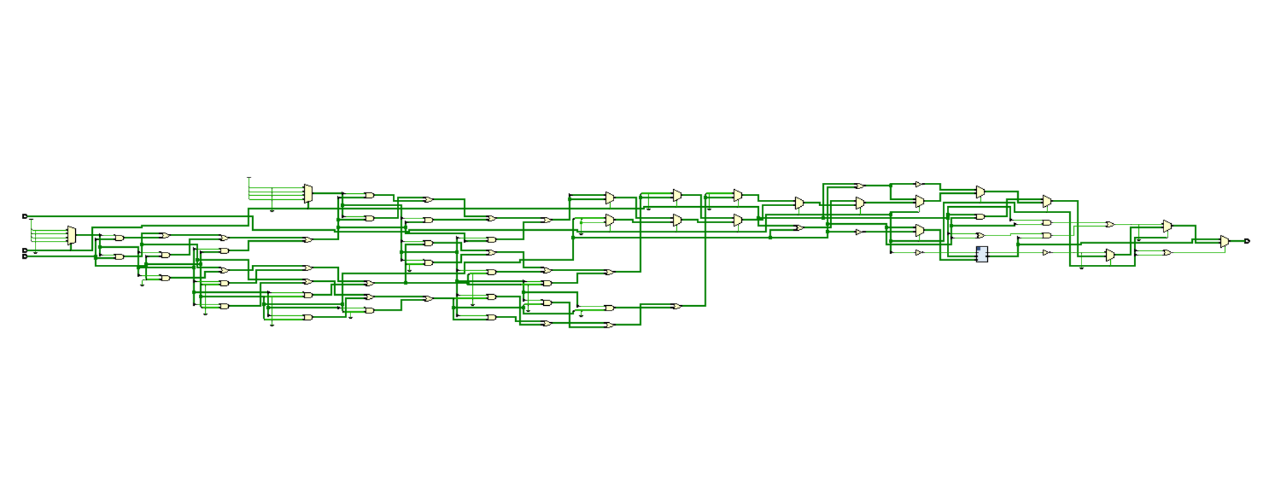


图4.6 运算器ALU（独热码）电路图

1. **实验心得体会**

通过本次实验，本人初步掌握了运算器ALU原理，并能够根据Verilog语言用独热码和编码两种译码方式实现出相关的电路，最后成功输出了预期的波形图。经过本次实验，本人还加深了对运算器ALU以及它能够实现的相关指令功能的理解，并再次提高了计算机组成原理相关知识的运用能力。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**计 算 机 组 成 原 理 实 验 报 告**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**班级：计科1802 姓名：孔天欣 班级序号：180235 学号：20188068**

**实验日期：2020.11.23**

**学院： 计算机与通信工程学院 专业： 计算机科学与技术**

**实验顺序：31 实验名称：译码器实验 指导教师：张旭**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验目的**

1.了解MIPS指令集中的运算指令，学会对这些指令进行归纳分类。

2.熟悉并掌握译码器的原理、功能和设计。

3.进一步加强运用 verilog 语言进行电路设计的能力。

4.为后续设计 cpu 的实验打下基础。**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验环境**

装有 vivado 软件的计算机一台。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验设计图**

一、译码器

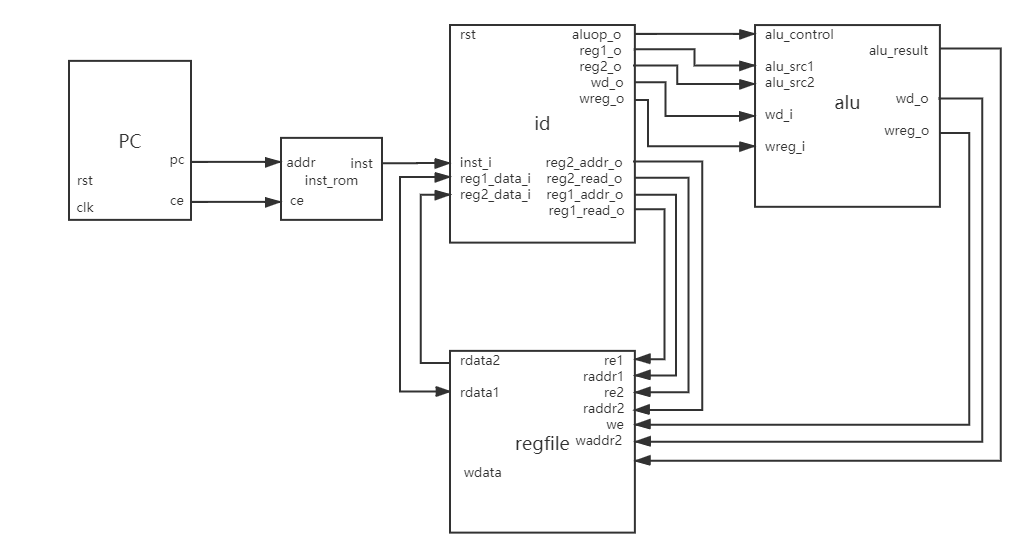


图6.1 单周期CPU实现细节框图

表6.1 译码器的接口描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
| 1 | rst | 1 | 输入 | 复位信号 |
| 2 | inst\_i | 32 | 输入 | 译码阶段的指令 |
| 3 | reg1\_data\_i | 32 | 输入 | 从regfile读入数据1 |
| 4 | reg2\_data\_i | 32 | 输入 | 从regfile读入数据2 |
| 5 | aluop\_o | 4 | 输出 | 译码阶段运算类型 |
| 6 | reg1\_o | 32 | 输出 | 译码阶段源操作数1 |
| 7 | reg2\_o | 32 | 输出 | 译码阶段源操作数2 |
| 8 | wd\_o | 5 | 输出 | 目的寄存器地址 |
| 9 | wreg\_o | 1 | 输出 | 是否要写入目的寄存器 |
| 10 | reg2\_addr\_o | 5 | 输出 | regfile第二个寄存器地址 |
| 11 | reg2\_read\_o | 1 | 输出 | regfile第二个寄存器读使能信号 |
| 12 | reg1\_addr\_o | 5 | 输出 | regfile第一个寄存器地址 |
| 13 | reg1\_read\_o | 1 | 输出 | regfile第一个寄存器读使能信号 |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验代码**

一、全局定义模块 define.v

*// 全局*

`define **RstEnable** 1**'**b1

`define **RstDisable** 1**'**b0

`define **ZeroWord** 32**'**h00000000

`define **WriteEnable** 1**'**b1

`define **WriteDisable** 1**'**b0

`define **ReadEnable** 1**'**b1

`define **ReadDisable** 1**'**b0

`define **DataAddrBus** 31:0

`define **DataBus** 31:0

`define **DataMemNum** 131071

`define **DataMemNumLog2** 17

`define **ByteWidth** 7:0

*// 通用寄存器regfile*

`define **RegAddrBus** 4:0

`define **RegBus** 31:0

`define **RegWidth** 32

`define **DoubleRegWidth** 64

`define **DoubleRegBus** 63:0

`define **RegNum** 32

`define **RegNumLog2** 5

`define **NOPRegAddr** 5**'**b00000

*//指令存储器inst\_rom*

`define **InstAddrBus** 31:0

`define **InstBus** 31:0

`define **InstMemNum** 131072

`define **InstMemNumLog2** 17

`define **ChipEnable** 1**'**b1

`define **ChipDisable** 1**'**b0

`define **InstValid** 1**'**b0

`define **InstInvalid** 1**'**b1

`define **AluOpBus** 3:0

*// ALU\_OP*

`define **ADD\_OP** 4**'**b0000

`define **SUB\_OP** 4**'**b0001

`define **SLT\_OP** 4**'**b0010

`define **SLTU\_OP** 4**'**b0011

`define **AND\_OP** 4**'**b0100

`define **NOR\_OP** 4**'**b0101

`define **OR\_OP** 4**'**b0110

`define **XOR\_OP** 4**'**b0111

`define **SLL\_OP** 4**'**b1000

`define **SRL\_OP** 4**'**b1001

`define **SRA\_OP** 4**'**b1010

`define **LUI\_OP** 4**'**b1011

`define **NOP\_OP** 4**'**b1111

*// 指令*

`define **EXE\_AND** 6**'**b100100

`define **EXE\_OR** 6**'**b100101

`define **EXE\_XOR** 6**'**b100110

`define **EXE\_NOR** 6**'**b100111

`define **EXE\_LUI** 6**'**b001111

`define **EXE\_SLL** 6**'**b000000

`define **EXE\_SRL** 6**'**b000010

`define **EXE\_SRA** 6**'**b000011

`define **EXE\_SLT** 6**'**b101010

`define **EXE\_SLTU** 6**'**b101011

`define **EXE\_ADD** 6**'**b100000

`define **EXE\_SUB** 6**'**b100010

`define **EXE\_SPECIAL\_INST** 6**'**b000000

二、译码器 ID

1. 设计模块 id.v

`timescale 1ns / 1ps

**module** id(

**input** wire rst,

**input** wire[`InstBus] inst\_i,

**input** wire[`RegBus] reg1\_data\_i,

**input** wire[`RegBus] reg2\_data\_i,

*// message to regfile*

**output** reg reg1\_read\_o,

**output** reg reg2\_read\_o,

**output** reg[`RegAddrBus] reg1\_addr\_o,

**output** reg[`RegAddrBus] reg2\_addr\_o,

*// message to run*

**output** reg[`AluOpBus] aluop\_o,

**output** reg[`RegBus] reg1\_o,

**output** reg[`RegBus] reg2\_o,

**output** reg[`RegAddrBus] wd\_o,

**output** reg wreg\_o

);

wire[5:0] op = inst\_i[31:26];

wire[4:0] op2 = inst\_i[10:6];

wire[5:0] op3 = inst\_i[5:0];

wire[4:0] op4 = inst\_i[20:16];

reg[`RegBus] imm;

reg instvalid;

**always** @(\*) **begin**

**if**(rst == `RstEnable) **begin**

aluop\_o <= `NOP\_OP;

wd\_o <= `NOPRegAddr;

wreg\_o <=`WriteDisable;

instvalid <= `InstValid;

reg1\_read\_o <=1'b0;

reg2\_read\_o <= 1'b0;

reg1\_addr\_o <= `NOPRegAddr;

reg2\_addr\_o <= `NOPRegAddr;

imm <= 32'h0;

**end** **else** **begin**

aluop\_o <= `NOP\_OP;

wd\_o <= inst\_i[15:11];

wreg\_o <= `WriteDisable;

instvalid <= `InstInvalid;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b0;

reg1\_addr\_o <= inst\_i[25:21];

reg2\_addr\_o <= inst\_i[20:16];

imm <= `ZeroWord;

**case** (op)

`EXE\_SPECIAL\_INST: **begin**

**case** (op2)

5'b00000: **begin**

**case**(op3)

`EXE\_OR: **begin**

wreg\_o <=`WriteEnable;

aluop\_o <= `OR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_AND: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `AND\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_XOR: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `XOR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_NOR: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `NOR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SLT: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLT\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SLTU: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLTU\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_ADD: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `ADD\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SUB: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SUB\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

**default**: **begin** **end**

**endcase**

**end**

**default**: **begin** **end**

**endcase**

**end**

`EXE\_LUI: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `LUI\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b0;

imm <= {inst\_i[15:0],16'h0};

wd\_o <= inst\_i[20:16];

instvalid <= `InstValid;

**end**

**default**: **begin** **end**

**endcase**

**if** (inst\_i[31:21] == 11'b00000000000) **begin**

**if**(op3 == `EXE\_SLL) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLL\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end** **else** **if** (op3 == `EXE\_SRL ) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SRL\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end** **else** **if** ( op3 == `EXE\_SRA ) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SRA\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end**

**end**

**end**

**end**

**always** @(\*) **begin**

**if** ( rst == `RstEnable) **begin**

reg1\_o <= `ZeroWord;

**end** **else** **if** (reg1\_read\_o == 1'b1) **begin**

reg1\_o <= reg1\_data\_i;

**end** **else** **if**(reg1\_read\_o == 1'b0) **begin**

reg1\_o <= imm;

**end** **else** **begin**

reg1\_o <= `ZeroWord;

**end**

**end**

**always** @(\*) **begin**

**if** ( rst == `RstEnable) **begin**

reg2\_o <= `ZeroWord;

**end** **else** **if** (reg2\_read\_o == 1'b1) **begin**

reg2\_o <= reg2\_data\_i;

**end** **else** **if**(reg2\_read\_o == 1'b0) **begin**

reg2\_o <= imm;

**end** **else** **begin**

reg2\_o <= `ZeroWord;

**end**

**end**

**endmodule**

2. 测试模块 id\_tb.v

`timescale 1ns / 1ps

**module** id\_tb();

reg rst;

reg[`InstBus] inst\_i;

reg[`RegBus] reg1\_data\_i;

reg[`RegBus] reg2\_data\_i;

wire reg1\_read\_o;

wire reg2\_read\_o;

wire[`RegAddrBus] reg1\_addr\_o;

wire[`RegAddrBus] reg2\_addr\_o;

wire[`AluOpBus] aluop\_o;

wire[`RegBus] reg1\_o;

wire[`RegBus] reg2\_o;

wire[`RegAddrBus] wd\_o;

wire wreg\_o;

reg[`InstBus] inst\_array[0:11];

**integer** i;

**initial** **begin**

$readmemh("D:/inst\_rom.data",inst\_array);

**end**

id id0(rst,inst\_i,reg1\_data\_i,reg2\_data\_i,reg1\_read\_o,reg2\_read\_o,

reg1\_addr\_o,reg2\_addr\_o,aluop\_o,reg1\_o,reg2\_o,wd\_o,wreg\_o);

**initial** **begin**

rst = `RstEnable;

#100;

rst = `RstDisable;

reg1\_data\_i = 32'h12345678;

reg2\_data\_i = 32'hfedcba98;

**for**(i = 0;i<12;i = i+1) **begin**

inst\_i = inst\_array[i];

#20;

**end**

#20 $stop;

**end**

**endmodule**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**四. 仿真波形及说明**

一、译码器 ID

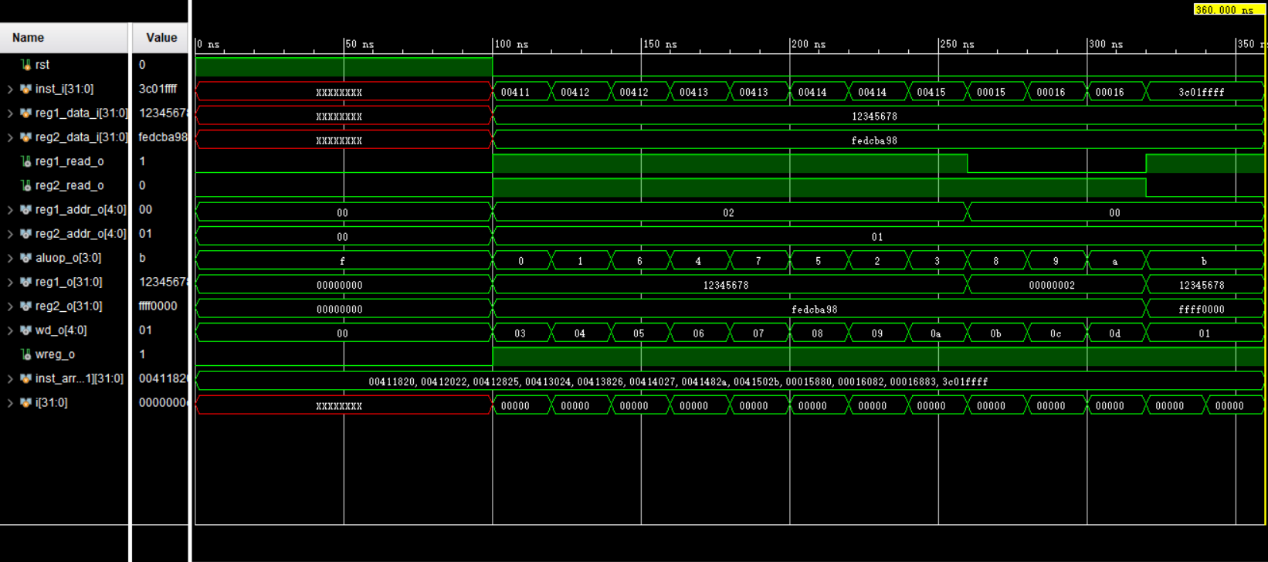


图6.2 译码器ID波形图

说明： 波形图中reg\_1\_data\_i和reg2\_data\_i都是不变的测试数值，inst\_i为读取inst\_rom.data文件中的机器码指令获得的数据，inst\_rom.data文件中对应的指令16进制机器码以及对应指令清单如下：

表6.2 机器码及其对应指令清单

|  |  |  |  |
| --- | --- | --- | --- |
| 机器码 | 对应指令 | 机器码 | 对应指令 |
| 00411820 | ADD $2,$1,$3 | 0041482a | SLT $2,$1,$9 |
| 00412022 | SUB $2,$1,$4 | 0041502b | SLTU $2,$1,$10 |
| 00412825 | OR $2,$1,$5 | 00015880 | SLL $1,$11,2 |
| 00413024 | AND $2,$1,$6 | 00016082 | SRL $1,$12,2 |
| 00413826 | XOR $2,$1,$7 | 00016883 | SRA $1,$13,2 |
| 00414027 | NOR $2,$1,$8 | 3c01ffff | LUI $1,65535 |

可以看到，波形图上的寄存器使用情况和表格中的指令是一一对应的。例如以指令ADD $2,$1,$3来说，在波形图中可以见到在100 ns ~ 120 ns处，读使能reg1\_read\_o置1，reg2\_read\_o置1，reg1\_addr\_o和reg2\_addr\_o分别是02和01，同时wd\_o是03，aluop\_o是0，对应defines.v中的*ADD\_OP 4'b0000*，意为将地址为2和1的寄存器值相加，运算结果送入地址为3的寄存器中，和翻译的结果一致，说明成功将机器码翻译成了对应的操作地址。而SUB,OR,AND,XOR,NOR,SLT,SLTU指令仅仅改变了目的寄存器的地址（为方便起见，将它依次设置为递增的形式，波形图上对应wd\_o）。除此以外，又例如指令SLL $1,$11,2，可以看到波形图260 ns ~ 280 ns处，读使能reg1\_read\_o置0（因为这个指令只需要一个寄存器），reg2\_read\_o置1，寄存器地址reg2\_addr\_o为01，译码阶段源操作数reg1\_o为2，同时送入寄存器地址wd\_o是0b，aluop\_o是8，对应defines.v的*SLL\_OP 4'b1000*，意为由立即数2指定位移量，对地址为01的寄存器进行逻辑左移，结果写入地址为11的寄存器中，也得到了成功的译码结果。其他指令例如SRL,SRA同理。

电路图：

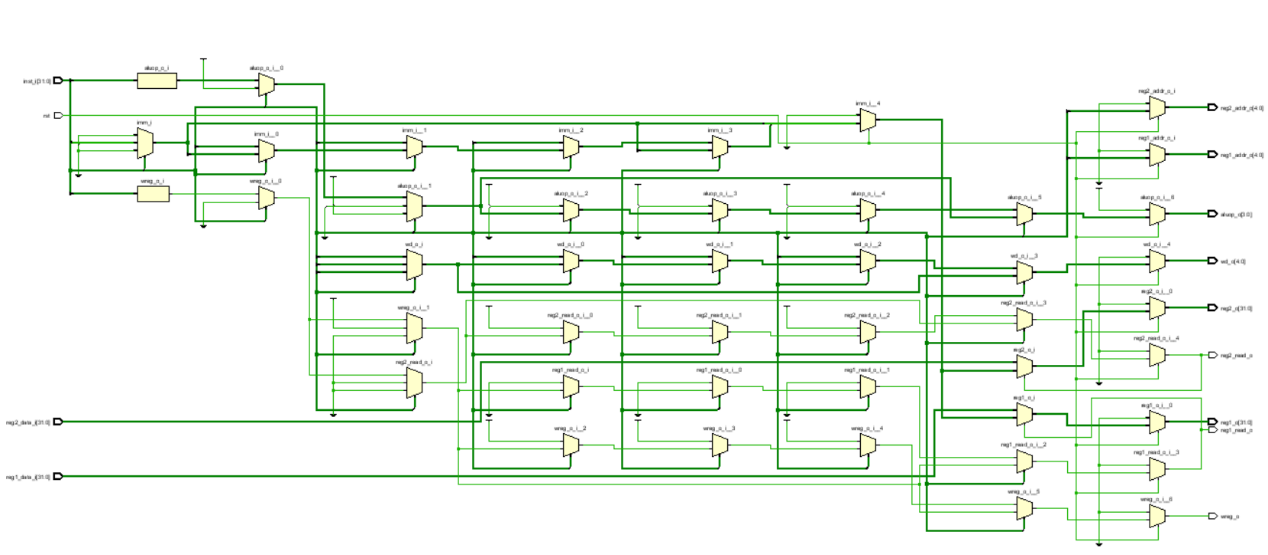


图6.3 译码器ID电路图

1. **实验心得体会**

通过本次实验，本人初步掌握了译码器ID原理，并能够根据Verilog语言实现出相关的电路，最后成功输出了预期的波形图。经过本次实验，本人还加深了对译码器ID以及它能够实现的相关指令功能的理解，并再次提高了计算机组成原理相关知识的运用能力。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**计 算 机 组 成 原 理 实 验 报 告**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**班级：计科1802 姓名：孔天欣 班级序号：180235 学号：20188068**

**实验日期：2020.11.30**

**学院： 计算机与通信工程学院 专业： 计算机科学与技术**

**实验顺序：31 实验名称：单周期CPU实验 指导教师：张旭**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验目的**

1.理解MIPS指令结构，理解MIPS指令集中常用指令的功能和编码，学会对这些指令进行归纳分类。

2.了解熟 MIPS体系的处理器结构，如哈佛结构的概念。

3.熟悉并掌握单周期CPU的原理和设计。

4.进一步加强运用verilog语言进行电路设计的能力。

5.为后续设计多周期cpu的实验打下基础。**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验环境**

装有 vivado 软件的计算机一台。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验设计图**

一、单周期CPU

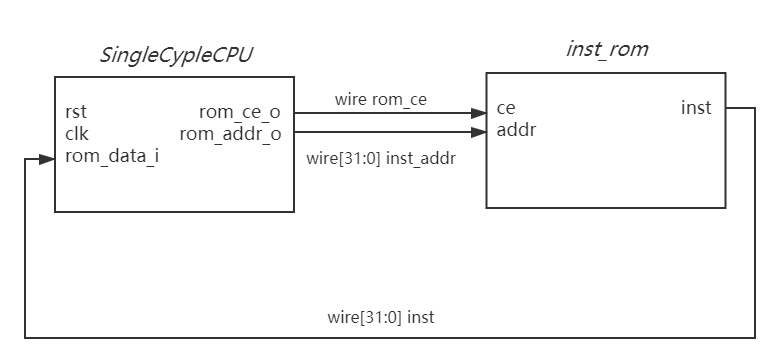


图7.1 单周期CPU实现框图

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验代码**

一、全局定义模块 define.v

*// 全局*

`define **RstEnable** 1**'**b1

`define **RstDisable** 1**'**b0

`define **ZeroWord** 32**'**h00000000

`define **WriteEnable** 1**'**b1

`define **WriteDisable** 1**'**b0

`define **ReadEnable** 1**'**b1

`define **ReadDisable** 1**'**b0

`define **DataAddrBus** 31:0

`define **DataBus** 31:0

`define **DataMemNum** 131071

`define **DataMemNumLog2** 17

`define **ByteWidth** 7:0

*// 通用寄存器regfile*

`define **RegAddrBus** 4:0

`define **RegBus** 31:0

`define **RegWidth** 32

`define **DoubleRegWidth** 64

`define **DoubleRegBus** 63:0

`define **RegNum** 32

`define **RegNumLog2** 5

`define **NOPRegAddr** 5**'**b00000

*//指令存储器inst\_rom*

`define **InstAddrBus** 31:0

`define **InstBus** 31:0

`define **InstMemNum** 131072

`define **InstMemNumLog2** 17

`define **ChipEnable** 1**'**b1

`define **ChipDisable** 1**'**b0

`define **InstValid** 1**'**b0

`define **InstInvalid** 1**'**b1

`define **AluOpBus** 3:0

*// ALU\_OP*

`define **ADD\_OP** 4**'**b0000

`define **SUB\_OP** 4**'**b0001

`define **SLT\_OP** 4**'**b0010

`define **SLTU\_OP** 4**'**b0011

`define **AND\_OP** 4**'**b0100

`define **NOR\_OP** 4**'**b0101

`define **OR\_OP** 4**'**b0110

`define **XOR\_OP** 4**'**b0111

`define **SLL\_OP** 4**'**b1000

`define **SRL\_OP** 4**'**b1001

`define **SRA\_OP** 4**'**b1010

`define **LUI\_OP** 4**'**b1011

`define **NOP\_OP** 4**'**b1111

*// 指令*

`define **EXE\_AND** 6**'**b100100

`define **EXE\_OR** 6**'**b100101

`define **EXE\_XOR** 6**'**b100110

`define **EXE\_NOR** 6**'**b100111

`define **EXE\_LUI** 6**'**b001111

`define **EXE\_SLL** 6**'**b000000

`define **EXE\_SRL** 6**'**b000010

`define **EXE\_SRA** 6**'**b000011

`define **EXE\_SLT** 6**'**b101010

`define **EXE\_SLTU** 6**'**b101011

`define **EXE\_ADD** 6**'**b100000

`define **EXE\_SUB** 6**'**b100010

`define **EXE\_SPECIAL\_INST** 6**'**b000000

二、单周期CPU

1. 设计模块 mips\_sopc.v

`timescale 1ns / 1ps

**module** mips\_sopc(

**input** wire clk,

**input** wire rst

);

wire[`InstAddrBus] inst\_addr;

wire[`InstBus] inst;

wire rom\_ce;

single\_cycle\_cpu single\_cycle\_cpu0(clk,rst,inst,inst\_addr,rom\_ce);

inst\_rom inst\_rom0(.ce(rom\_ce),.addr(inst\_addr),.inst(inst));

**endmodule**

2. 设计模块 single\_cycle\_cpu.v

`timescale 1ns / 1ps

**module** single\_cycle\_cpu(

**input** wire clk,

**input** wire rst,

**input** wire[`RegBus] rom\_data\_i,

**output** wire[`RegBus] rom\_addr\_o,

**output** wire rom\_ce\_o

);

pc\_reg pc\_reg0(rst,clk,rom\_addr\_o,rom\_ce\_o);

wire[`AluOpBus] id\_aluop\_o;

wire[`RegBus] id\_reg1\_o;

wire[`RegBus] id\_reg2\_o;

wire[`RegAddrBus] id\_wd\_o;

wire id\_wreg\_o;

wire reg1\_read;

wire reg2\_read;

wire[`RegAddrBus] reg1\_addr;

wire[`RegAddrBus] reg2\_addr;

wire[`RegBus] reg1\_data;

wire[`RegBus] reg2\_data;

id id0(.rst(rst),.inst\_i(rom\_data\_i),.aluop\_o(id\_aluop\_o),

.reg1\_o(id\_reg1\_o),.reg2\_o(id\_reg2\_o),.wd\_o(id\_wd\_o),

.wreg\_o(id\_wreg\_o),.reg1\_read\_o(reg1\_read),.reg1\_addr\_o(reg1\_addr),

.reg2\_read\_o(reg2\_read),.reg2\_addr\_o(reg2\_addr),.reg1\_data\_i(reg1\_data),

.reg2\_data\_i(reg2\_data));

wire[`RegBus] wdata\_o;

wire[`RegAddrBus] wd\_o;

wire wreg\_o;

alu alu0(id\_aluop\_o,id\_reg1\_o,id\_reg2\_o,id\_wd\_o,id\_wreg\_o,

wdata\_o,wd\_o,wreg\_o);

regfile regfile0(.clk(clk),.rst(rst),.re1(reg1\_read),.raddr1(reg1\_addr),

.re2(reg2\_read),.raddr2(reg2\_addr),.we(wreg\_o),.waddr(wd\_o),.wdata(wdata\_o),

.rdata1(reg1\_data),.rdata2(reg2\_data));

**endmodule**

3. 设计模块 pc\_reg.v

`timescale 1ns / 1ns

**module** pc\_reg(

**input** wire rst,

**input** wire clk,

**output** reg[31:0] pc,

**output** reg ce

);

**always**@(**posedge** clk) **begin**

**if**(rst==`RstEnable) **begin**

ce<=`ChipDisable;

**end** **else** **begin**

ce<=`ChipEnable;

**end**

**end**

**always**@(**posedge** clk) **begin**

**if**(ce==`ChipDisable) **begin**

pc<=32'h0000\_0000;

**end** **else** **begin**

pc <= pc + 4'h4;

**end**

**end**

**endmodule**

4. 设计模块 id.v

`timescale 1ns / 1ps

**module** id(

**input** wire rst,

**input** wire[`InstBus] inst\_i,

**input** wire[`RegBus] reg1\_data\_i,

**input** wire[`RegBus] reg2\_data\_i,

*// message to regfile*

**output** reg reg1\_read\_o,

**output** reg reg2\_read\_o,

**output** reg[`RegAddrBus] reg1\_addr\_o,

**output** reg[`RegAddrBus] reg2\_addr\_o,

*// message to run*

**output** reg[`AluOpBus] aluop\_o,

**output** reg[`RegBus] reg1\_o,

**output** reg[`RegBus] reg2\_o,

**output** reg[`RegAddrBus] wd\_o,

**output** reg wreg\_o

);

wire[5:0] op = inst\_i[31:26];

wire[4:0] op2 = inst\_i[10:6];

wire[5:0] op3 = inst\_i[5:0];

wire[4:0] op4 = inst\_i[20:16];

reg[`RegBus] imm;

reg instvalid;

**always** @(\*) **begin**

**if**(rst == `RstEnable) **begin**

aluop\_o <= `NOP\_OP;

wd\_o <= `NOPRegAddr;

wreg\_o <=`WriteDisable;

instvalid <= `InstValid;

reg1\_read\_o <=1'b0;

reg2\_read\_o <= 1'b0;

reg1\_addr\_o <= `NOPRegAddr;

reg2\_addr\_o <= `NOPRegAddr;

imm <= 32'h0;

**end** **else** **begin**

aluop\_o <= `NOP\_OP;

wd\_o <= inst\_i[15:11];

wreg\_o <= `WriteDisable;

instvalid <= `InstInvalid;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b0;

reg1\_addr\_o <= inst\_i[25:21];

reg2\_addr\_o <= inst\_i[20:16];

imm <= `ZeroWord;

**case** (op)

`EXE\_SPECIAL\_INST: **begin**

**case** (op2)

5'b00000: **begin**

**case**(op3)

`EXE\_OR: **begin**

wreg\_o <=`WriteEnable;

aluop\_o <= `OR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_AND: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `AND\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_XOR: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `XOR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_NOR: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `NOR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SLT: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLT\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SLTU: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLTU\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_ADD: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `ADD\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SUB: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SUB\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

**default**: **begin** **end**

**endcase**

**end**

**default**: **begin** **end**

**endcase**

**end**

`EXE\_LUI: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `LUI\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b0;

imm <= {inst\_i[15:0],16'h0};

wd\_o <= inst\_i[20:16];

instvalid <= `InstValid;

**end**

**default**: **begin** **end**

**endcase**

**if** (inst\_i[31:21] == 11'b00000000000) **begin**

**if**(op3 == `EXE\_SLL) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLL\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end** **else** **if** (op3 == `EXE\_SRL ) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SRL\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end** **else** **if** ( op3 == `EXE\_SRA ) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SRA\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end**

**end**

**end**

**end**

**always** @(\*) **begin**

**if** ( rst == `RstEnable) **begin**

reg1\_o <= `ZeroWord;

**end** **else** **if** (reg1\_read\_o == 1'b1) **begin**

reg1\_o <= reg1\_data\_i;

**end** **else** **if**(reg1\_read\_o == 1'b0) **begin**

reg1\_o <= imm;

**end** **else** **begin**

reg1\_o <= `ZeroWord;

**end**

**end**

**always** @(\*) **begin**

**if** ( rst == `RstEnable) **begin**

reg2\_o <= `ZeroWord;

**end** **else** **if** (reg2\_read\_o == 1'b1) **begin**

reg2\_o <= reg2\_data\_i;

**end** **else** **if**(reg2\_read\_o == 1'b0) **begin**

reg2\_o <= imm;

**end** **else** **begin**

reg2\_o <= `ZeroWord;

**end**

**end**

**endmodule**

5. 设计模块 alu.v

`timescale 1ns / 1ps

**module** alu(

**input**[3:0] alu\_control,

**input**[31:0] alu\_src1,

**input**[31:0] alu\_src2,

**input** wire[`RegAddrBus] wd\_i,

**input** wire wreg\_i,

**output** reg[31:0] alu\_result,

**output** reg[`RegAddrBus] wd\_o,

**output** reg wreg\_o

);

wire[31:0] alu\_src2\_mux;

wire[31:0] result\_sum;

**assign** alu\_src2\_mux = (alu\_control==`SUB\_OP||alu\_control==`SLT\_OP)?(~alu\_src2)+1:alu\_src2;

**assign** result\_sum = alu\_src1+alu\_src2\_mux;

*// 比较结果*

**assign** src1\_lt\_src2 = ((alu\_control==`SLT\_OP))?

((alu\_src1[31]&&!alu\_src2[31])||

(!alu\_src1[31]&&!alu\_src2[31]&&result\_sum[31])||

(alu\_src1[31]&&alu\_src2[31]&&result\_sum[31])):(alu\_src1<alu\_src2);

**always** @(\*) **begin**

wd\_o = wd\_i;

wreg\_o = wreg\_i;

**case**(alu\_control)

`ADD\_OP,`SUB\_OP:**begin**

alu\_result = result\_sum;

**end**

`SLT\_OP,`SLTU\_OP:**begin**

alu\_result = src1\_lt\_src2;

**end**

`AND\_OP:**begin**

alu\_result = alu\_src1 & alu\_src2;

**end**

`NOR\_OP:**begin**

alu\_result = ~(alu\_src1|alu\_src2);

**end**

`OR\_OP:**begin**

alu\_result = alu\_src1 | alu\_src2;

**end**

`XOR\_OP:**begin**

alu\_result = alu\_src1 ^ alu\_src2;

**end**

`SLL\_OP:**begin**

alu\_result = alu\_src2 << alu\_src1[4:0];

**end**

`SRL\_OP:**begin**

alu\_result = alu\_src2 >> alu\_src1[4:0];

**end**

`SRA\_OP:**begin**

alu\_result = ({32{alu\_src2[31]}} << (6'd32-{1'b0,alu\_src1[4:0]}))

| alu\_src2 >> alu\_src1[4:0];

**end**

`LUI\_OP:**begin**

alu\_result = {alu\_src2[15:0],16'd0};

**end**

**default**:**begin**

alu\_result = 32'b0;

**end**

**endcase**

**end**

**endmodule**

6. 设计模块 regfile.v

**module** regfile(

**input** wire clk,

**input** wire rst,

*// write*

**input** wire we,

**input** wire[`RegAddrBus] waddr,

**input** wire[`RegBus] wdata,

*//read 1*

**input** wire re1,

**input** wire[`RegAddrBus] raddr1,

**output** reg[`RegBus] rdata1,

*//read 2*

**input** wire re2,

**input** wire[`RegAddrBus] raddr2,

**output** reg[`RegBus] rdata2

);

reg[`RegBus] regs[0:`RegNum-1];

**initial** **begin**

regs[1]=32'h12345678;

regs[2]=32'hfedcba98;

**end**

**always** @ (**posedge** clk) **begin**

**if**(rst == `RstDisable) **begin**

**if**((we == `WriteEnable) && (waddr != `RegNumLog2'h0)) **begin**

regs[waddr] <= wdata;

**end**

**end**

**end**

**always** @(\*) **begin**

**if**(rst == `RstEnable) **begin**

rdata1 <= `ZeroWord;

**end** **else** **if** (raddr1 == `RegNumLog2'h0) **begin**

rdata1 <= `ZeroWord;

*// read and write*

**end** **else** **if**((raddr1 == waddr) && (we == `WriteEnable)

&& (re1 == `ReadEnable)) **begin**

rdata1 <= wdata;

**end** **else** **if** (re1 == `ReadEnable) **begin**

rdata1 <= regs[raddr1];

**end** **else** **begin**

rdata1 <= `ZeroWord;

**end**

**end**

**always** @(\*) **begin**

**if**(rst == `RstEnable) **begin**

rdata2 <= `ZeroWord;

**end** **else** **if** (raddr2 == `RegNumLog2'h0) **begin**

rdata2 <= `ZeroWord;

**end** **else** **if**((raddr2 == waddr) && (we == `WriteEnable)

&& (re2 == `ReadEnable)) **begin**

rdata2 <= wdata;

**end** **else** **if** (re2 == `ReadEnable) **begin**

rdata2 <= regs[raddr2];

**end** **else** **begin**

rdata2 <= `ZeroWord;

**end**

**end**

**endmodule**

7. 设计模块 inst\_rom.v

`timescale 1ns / 1ns

**module** inst\_rom(

**input** wire clk,

**input** wire ce,

**input** wire[`InstAddrBus] addr,

**output** reg[`InstBus] inst

);

reg[`InstBus] inst\_mem[0:`InstMemNum-1];

**initial** $readmemh ( "D:/inst\_rom.data ",inst\_mem);

**always** @(\*) **begin**

**if**(ce==`ChipDisable) **begin**

inst <= `ZeroWord;

**end** **else** **begin**

inst <= inst\_mem[addr[`InstMemNumLog2 + 1:2]];

**end**

**end**

**endmodule**

8. 测试模块 mips\_sopc\_tb.v

`timescale 1ns / 1ps

**module** mips\_sopc\_tb();

reg clk;

reg rst;

**initial** **begin**

clk = 1'b0;

**forever** #10 clk = ~clk;

**end**

**initial** **begin**

rst = 1;

#100 rst=0;

#1000 $stop;

**end**

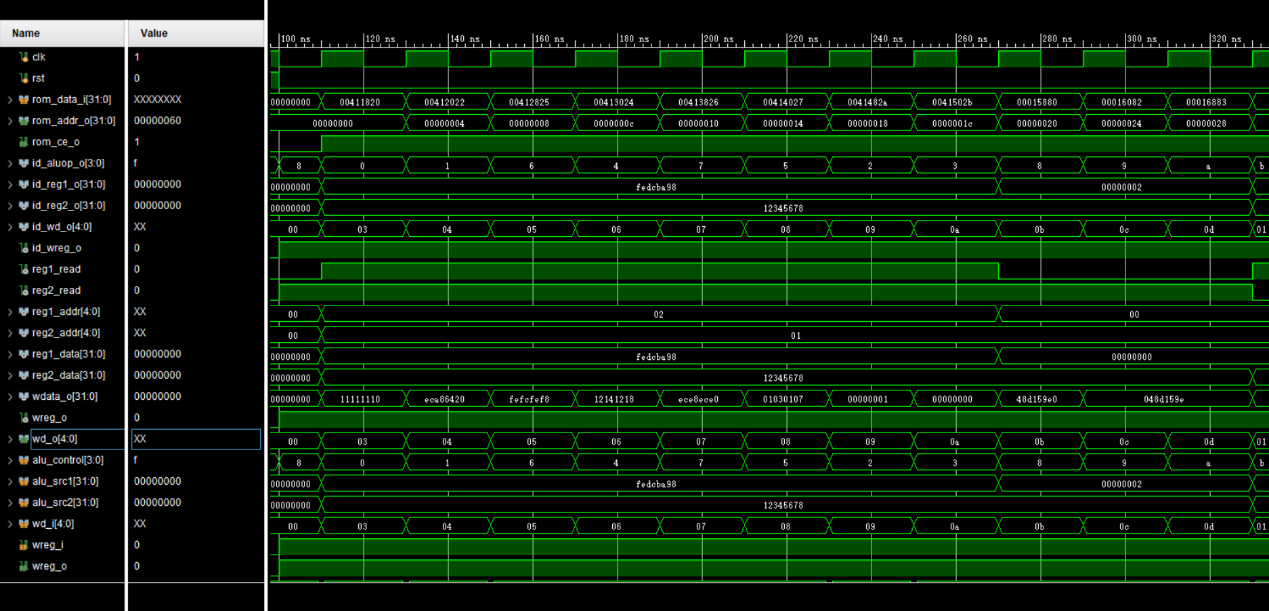
mips\_sopc mips\_sopc0(clk,rst);

**endmodule**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**四. 仿真波形及说明**

一、单周期CPU



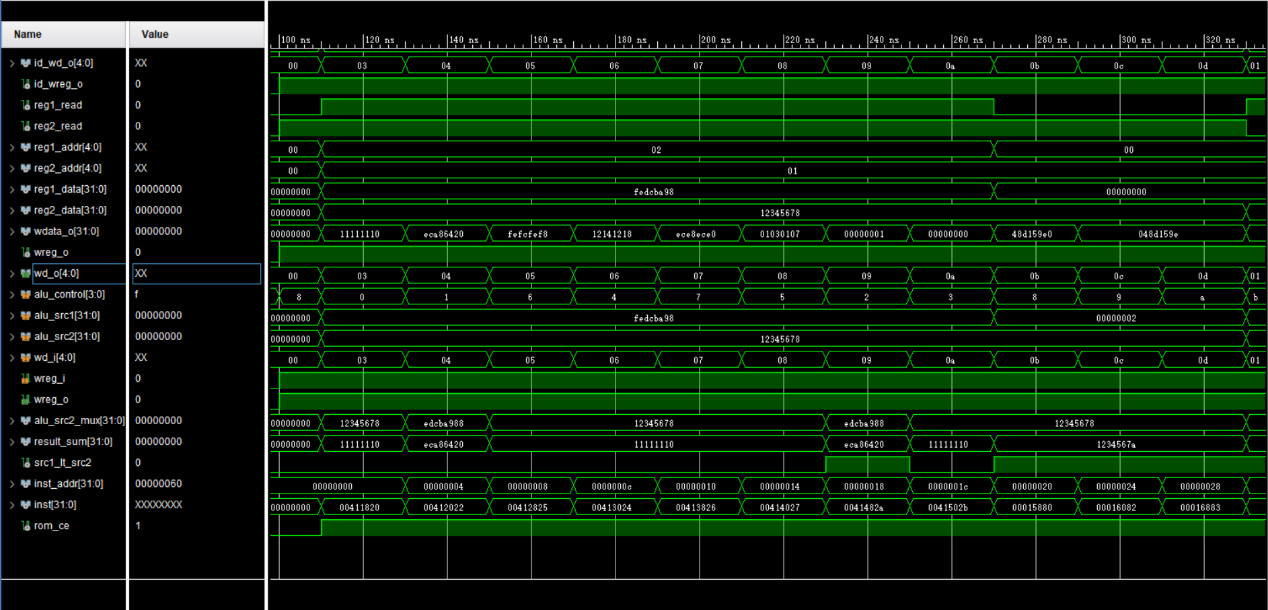


图7.2 单周期CPU波形图集

说明： 图中可见，各个模块经过连接和组装后，得到了一个完整的单周期CPU。通过在两个寄存器（波形图中reg1\_data和reg2\_data）初始化值*32’h 12345678*和*32’h fedcba98*，由计数器PC确定指令地址，inst\_rom读取inst\_rom.data中的对应指令数据并传送至译码器id进行译码，然后由alu执行计算过程，并将结果写入设定的寄存器。由波形图可知，实验六中自定义的指令已经成功得到译码（见id\_reg1\_o，id\_reg2\_o，id\_wd\_o和id\_aluop\_o），和执行，并获得了正确的计算结果（见wdata\_o）。以输入rom\_data\_i中00411820（机器码，对应指令ADD $2,$1,$3）为例，对应的wdata\_o是1111 1110，和两个寄存器值相加（12345678h + fedcba98）结果的低8位相同。送入的寄存器也是3号（见wd\_o）。又以输入00412022（机器码，对应指令SUB $2,$1,$4）为例，wdata\_o中显示输出eca86420，与两个寄存器值相减（fedcba98 - 12345678h） 结果一致，送入的寄存器也是4号。后续的各种指令对应计算都经过了验证，输出结果和对应寄存器都是正确的。

全局电路图如下。

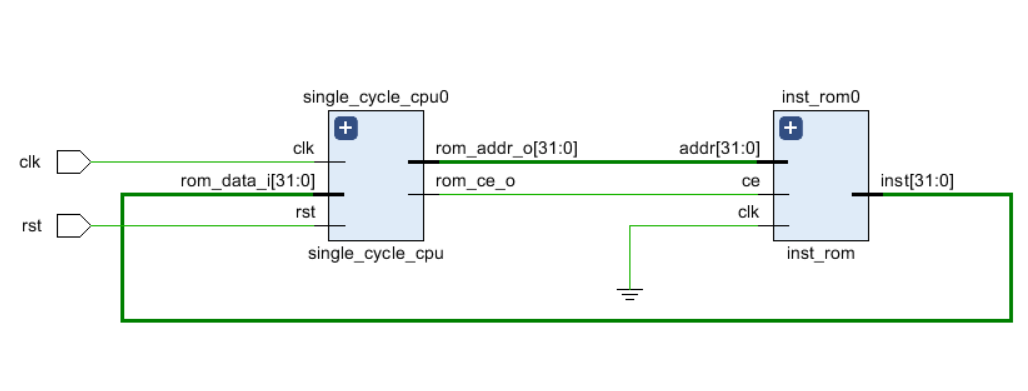


图7.3 全局电路图

对于单周期CPU内部结构，电路图如下。

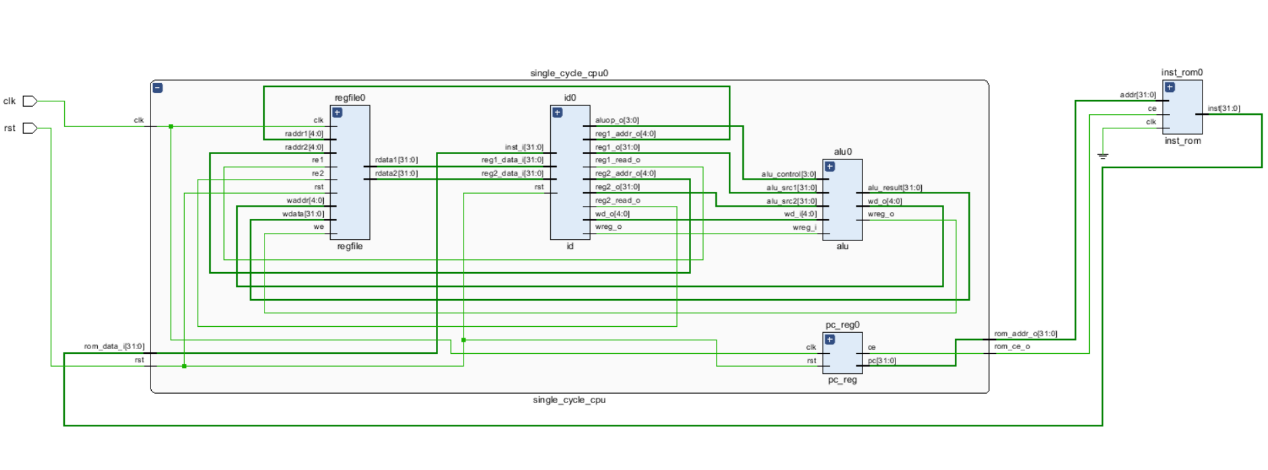


图7.4 单周期CPU电路图

1. **实验心得体会**

通过本次实验，本人初步掌握了单周期CPU原理，并能够根据Verilog语言实现出相关的电路，最后成功输出了预期的波形图。经过本次实验，本人还加深了单周期CPU以及它的主要组成模块和实现方法的理解，并再次提高了计算机组成原理相关知识的运用能力。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**计 算 机 组 成 原 理 实 验 报 告**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**班级：计科1802 姓名：孔天欣 班级序号：180235 学号：20188068**

**实验日期：2020.12.07**

**学院： 计算机与通信工程学院 专业： 计算机科学与技术**

**实验顺序：31 实验名称：五级流水CPU实验 指导教师：张旭**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验目的**

1.在单周期CPU实验完成的提前下，理解多周期流水线的概念。

2.熟悉并掌握多周期流水CPU的原理和设计。**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验环境**

装有 vivado 软件的计算机一台。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验设计图**

一、五级流水CPU

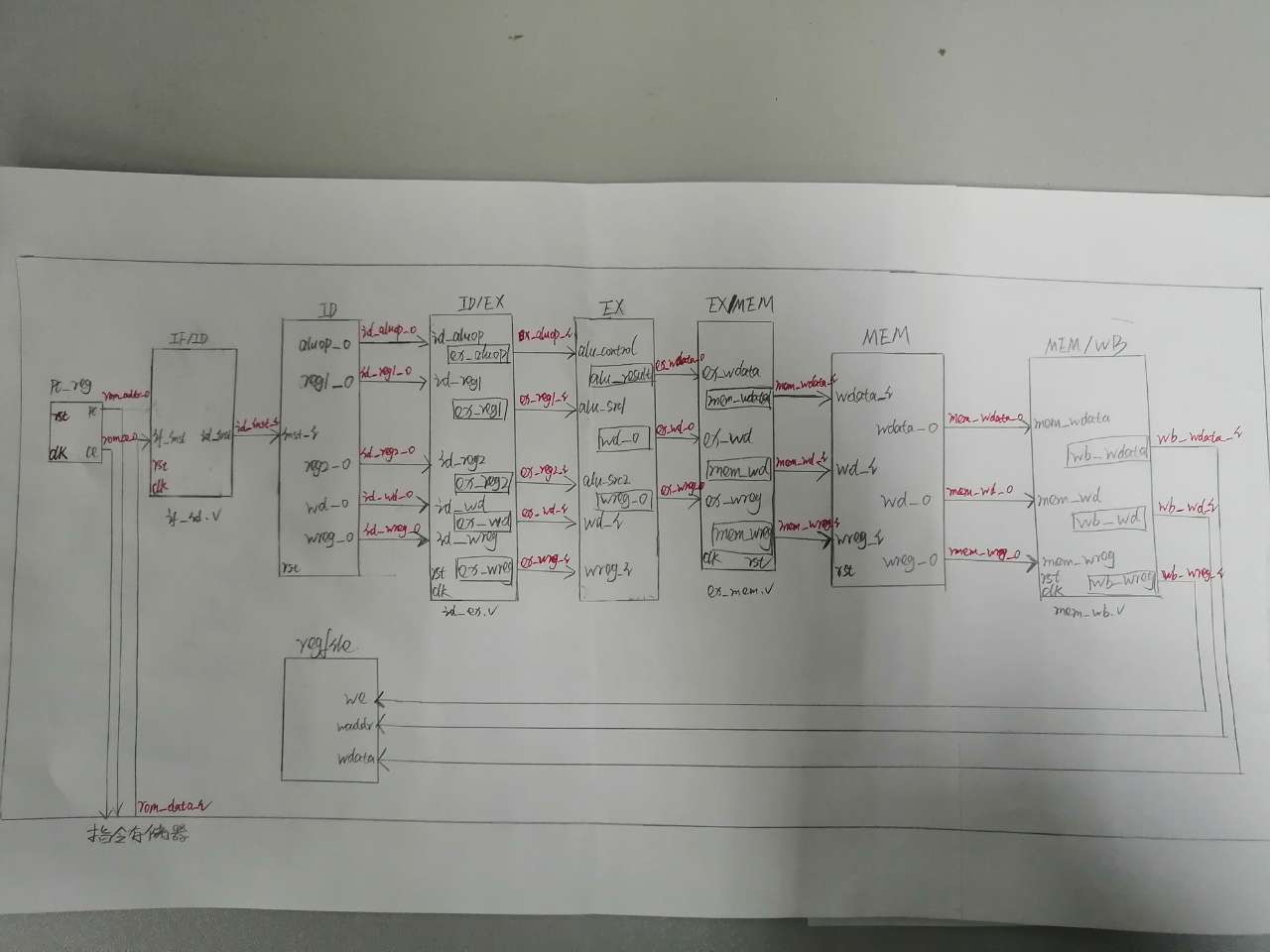


图8.1 单周期CPU实现框图

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **实验代码**

一、全局定义模块 define.v

*// 全局*

`define **RstEnable** 1**'**b1

`define **RstDisable** 1**'**b0

`define **ZeroWord** 32**'**h00000000

`define **WriteEnable** 1**'**b1

`define **WriteDisable** 1**'**b0

`define **ReadEnable** 1**'**b1

`define **ReadDisable** 1**'**b0

`define **DataAddrBus** 31:0

`define **DataBus** 31:0

`define **DataMemNum** 131071

`define **DataMemNumLog2** 17

`define **ByteWidth** 7:0

*// 通用寄存器regfile*

`define **RegAddrBus** 4:0

`define **RegBus** 31:0

`define **RegWidth** 32

`define **DoubleRegWidth** 64

`define **DoubleRegBus** 63:0

`define **RegNum** 32

`define **RegNumLog2** 5

`define **NOPRegAddr** 5**'**b00000

*//指令存储器inst\_rom*

`define **InstAddrBus** 31:0

`define **InstBus** 31:0

`define **InstMemNum** 131072

`define **InstMemNumLog2** 17

`define **ChipEnable** 1**'**b1

`define **ChipDisable** 1**'**b0

`define **InstValid** 1**'**b0

`define **InstInvalid** 1**'**b1

`define **AluOpBus** 3:0

*// ALU\_OP*

`define **ADD\_OP** 4**'**b0000

`define **SUB\_OP** 4**'**b0001

`define **SLT\_OP** 4**'**b0010

`define **SLTU\_OP** 4**'**b0011

`define **AND\_OP** 4**'**b0100

`define **NOR\_OP** 4**'**b0101

`define **OR\_OP** 4**'**b0110

`define **XOR\_OP** 4**'**b0111

`define **SLL\_OP** 4**'**b1000

`define **SRL\_OP** 4**'**b1001

`define **SRA\_OP** 4**'**b1010

`define **LUI\_OP** 4**'**b1011

`define **NOP\_OP** 4**'**b1111

*// 指令*

`define **EXE\_AND** 6**'**b100100

`define **EXE\_OR** 6**'**b100101

`define **EXE\_XOR** 6**'**b100110

`define **EXE\_NOR** 6**'**b100111

`define **EXE\_LUI** 6**'**b001111

`define **EXE\_SLL** 6**'**b000000

`define **EXE\_SRL** 6**'**b000010

`define **EXE\_SRA** 6**'**b000011

`define **EXE\_SLT** 6**'**b101010

`define **EXE\_SLTU** 6**'**b101011

`define **EXE\_ADD** 6**'**b100000

`define **EXE\_SUB** 6**'**b100010

`define **EXE\_SPECIAL\_INST** 6**'**b000000

二、五级流水CPU

*展示五级流水相关部件，其他部件见实验七。*

1. 设计模块 mips\_sopc.v

`timescale 1ns / 1ps

**module** mips\_sopc(

**input** wire clk,

**input** wire rst

);

wire[`InstAddrBus] inst\_addr;

wire[`InstBus] inst;

wire rom\_ce;

pipeline\_cpu pineline\_cpu0(clk,rst,inst,inst\_addr,rom\_ce);

inst\_rom inst\_rom0(.ce(rom\_ce),.addr(inst\_addr),.inst(inst));

**endmodule**

2. 设计模块 pipeline\_cpu.v

`timescale 1ns / 1ps

**module** pipeline\_cpu(

**input** wire clk,

**input** wire rst,

**input** wire[`RegBus] rom\_data\_i,

**output** wire[`RegBus] rom\_addr\_o,

**output** wire rom\_ce\_o

);

pc\_reg pc\_reg0(.clk(clk),.rst(rst),.pc(rom\_addr\_o),.ce(rom\_ce\_o));

*// link IF/ID to ID*

wire[`InstBus] id\_inst\_i;

*// link id to id/ex*

wire[`AluOpBus] id\_aluop\_o;

wire[`RegBus] id\_reg1\_o;

wire[`RegBus] id\_reg2\_o;

wire[`RegAddrBus] id\_wd\_o;

wire id\_wreg\_o;

*// link id/ex to ex*

wire[`AluOpBus] ex\_aluop\_i;

wire[`RegBus] ex\_reg1\_i;

wire[`RegBus] ex\_reg2\_i;

wire[`RegAddrBus] ex\_wd\_i;

wire ex\_wreg\_i;

*//link ex to ex/mem*

wire[`RegBus] ex\_wdata\_o;

wire[`RegAddrBus] ex\_wd\_o;

wire ex\_wreg\_o;

*// link ex/mem to mem*

wire[`RegBus] mem\_wdata\_i;

wire[`RegAddrBus] mem\_wd\_i;

wire mem\_wreg\_i;

*//link mem to mem/wb*

wire[`RegBus] mem\_wdata\_o;

wire[`RegAddrBus] mem\_wd\_o;

wire mem\_wreg\_o;

*// link mem/wb to regfile*

wire[`RegBus] wb\_wdata\_i;

wire[`RegAddrBus] wb\_wd\_i;

wire wb\_wreg\_i;

*// link id to regfile*

wire reg1\_read;

wire reg2\_read;

wire[`RegAddrBus] reg1\_addr;

wire[`RegAddrBus] reg2\_addr;

wire[`RegBus] reg1\_data;

wire[`RegBus] reg2\_data;

if\_id if\_id0(clk,rst,rom\_data\_i,id\_inst\_i);

id id0(.rst(rst),.inst\_i(id\_inst\_i),.aluop\_o(id\_aluop\_o),

.reg1\_o(id\_reg1\_o),.reg2\_o(id\_reg2\_o),.wd\_o(id\_wd\_o),

.wreg\_o(id\_wreg\_o),.reg1\_read\_o(reg1\_read),.reg1\_addr\_o(reg1\_addr),

.reg2\_read\_o(reg2\_read),.reg2\_addr\_o(reg2\_addr),.reg1\_data\_i(reg1\_data),

.reg2\_data\_i(reg2\_data));

id\_ex id\_ex0(.rst(rst),.clk(clk),.id\_aluop(id\_aluop\_o),

.id\_reg1(id\_reg1\_o),.id\_reg2(id\_reg2\_o),.id\_wd(id\_wd\_o),

.id\_wreg(id\_wreg\_o),.ex\_aluop(ex\_aluop\_i),.ex\_reg1(

ex\_reg1\_i),.ex\_reg2(ex\_reg2\_i),.ex\_wd(ex\_wd\_i),.ex\_wreg(ex\_wreg\_i));

alu alu0(.alu\_control(ex\_aluop\_i),.alu\_src1(ex\_reg1\_i),.alu\_src2(ex\_reg2\_i),

.wd\_i(ex\_wd\_i),.wreg\_i(ex\_wreg\_i),.alu\_result(ex\_wdata\_o),.wd\_o(ex\_wd\_o),.wreg\_o(ex\_wreg\_o));

ex\_mem ex\_mem0(.rst(rst),.clk(clk),.ex\_wdata(ex\_wdata\_o),.ex\_wd(ex\_wd\_o),.ex\_wreg(ex\_wreg\_o),

.mem\_wdata(mem\_wdata\_i),.mem\_wd(mem\_wd\_i),.mem\_wreg(mem\_wreg\_i));

mem mem0(.rst(rst),.wdata\_i(mem\_wdata\_i),.wd\_i(mem\_wd\_i),.wreg\_i(mem\_wreg\_i),

.wdata\_o(mem\_wdata\_o),.wd\_o(mem\_wd\_o),.wreg\_o(mem\_wreg\_o));

mem\_wb mem\_wb0(.rst(rst),.clk(clk),.mem\_wdata(mem\_wdata\_o),.mem\_wd(mem\_wd\_o)

,.mem\_wreg(mem\_wreg\_o),.wb\_wdata(wb\_wdata\_i),.wb\_wd(wb\_wd\_i),.wb\_wreg(wb\_wreg\_i));

regfile regfile0(.clk(clk),.rst(rst),.re1(reg1\_read),.raddr1(reg1\_addr),

.re2(reg2\_read),.raddr2(reg2\_addr),.we(wb\_wreg\_i),.waddr(wb\_wd\_i),.wdata(wb\_wdata\_i),

.rdata1(reg1\_data),.rdata2(reg2\_data));

**endmodule**

3. 设计模块 pc\_reg.v

`timescale 1ns / 1ns

**module** pc\_reg(

**input** wire rst,

**input** wire clk,

**output** reg[31:0] pc,

**output** reg ce

);

**always**@(**posedge** clk) **begin**

**if**(rst==`RstEnable) **begin**

ce<=`ChipDisable;

**end** **else** **begin**

ce<=`ChipEnable;

**end**

**end**

**always**@(**posedge** clk) **begin**

**if**(ce==`ChipDisable) **begin**

pc<=32'h0000\_0000;

**end** **else** **begin**

pc <= pc + 4'h4;

**end**

**end**

**endmodule**

4. 设计模块 if\_id.v

`timescale 1ns / 1ps

**module** if\_id(

**input** wire clk,

**input** wire rst,

**input**[`InstBus] if\_inst,

**output** reg[`InstBus] id\_inst

);

**always**@(**posedge** clk)**begin**

**if**(rst==`RstEnable)**begin**

id\_inst<=`ZeroWord;

**end** **else** **begin**

id\_inst<=if\_inst;

**end**

**end**

**endmodule**

5. 设计模块 id.v

`timescale 1ns / 1ps

**module** id(

**input** wire rst,

**input** wire[`InstBus] inst\_i,

**input** wire[`RegBus] reg1\_data\_i,

**input** wire[`RegBus] reg2\_data\_i,

*// message to regfile*

**output** reg reg1\_read\_o,

**output** reg reg2\_read\_o,

**output** reg[`RegAddrBus] reg1\_addr\_o,

**output** reg[`RegAddrBus] reg2\_addr\_o,

*// message to run*

**output** reg[`AluOpBus] aluop\_o,

**output** reg[`RegBus] reg1\_o,

**output** reg[`RegBus] reg2\_o,

**output** reg[`RegAddrBus] wd\_o,

**output** reg wreg\_o

);

wire[5:0] op = inst\_i[31:26];

wire[4:0] op2 = inst\_i[10:6];

wire[5:0] op3 = inst\_i[5:0];

wire[4:0] op4 = inst\_i[20:16];

reg[`RegBus] imm;

reg instvalid;

**always** @(\*) **begin**

**if**(rst == `RstEnable) **begin**

aluop\_o <= `NOP\_OP;

wd\_o <= `NOPRegAddr;

wreg\_o <=`WriteDisable;

instvalid <= `InstValid;

reg1\_read\_o <=1'b0;

reg2\_read\_o <= 1'b0;

reg1\_addr\_o <= `NOPRegAddr;

reg2\_addr\_o <= `NOPRegAddr;

imm <= 32'h0;

**end** **else** **begin**

aluop\_o <= `NOP\_OP;

wd\_o <= inst\_i[15:11];

wreg\_o <= `WriteDisable;

instvalid <= `InstInvalid;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b0;

reg1\_addr\_o <= inst\_i[25:21];

reg2\_addr\_o <= inst\_i[20:16];

imm <= `ZeroWord;

**case** (op)

`EXE\_SPECIAL\_INST: **begin**

**case** (op2)

5'b00000: **begin**

**case**(op3)

`EXE\_OR: **begin**

wreg\_o <=`WriteEnable;

aluop\_o <= `OR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_AND: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `AND\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_XOR: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `XOR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_NOR: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `NOR\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SLT: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLT\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SLTU: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLTU\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_ADD: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `ADD\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

`EXE\_SUB: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SUB\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b1;

instvalid <= `InstValid;

**end**

**default**: **begin** **end**

**endcase**

**end**

**default**: **begin** **end**

**endcase**

**end**

`EXE\_LUI: **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `LUI\_OP;

reg1\_read\_o <=1'b1;

reg2\_read\_o <= 1'b0;

imm <= {inst\_i[15:0],16'h0};

wd\_o <= inst\_i[20:16];

instvalid <= `InstValid;

**end**

**default**: **begin** **end**

**endcase**

**if** (inst\_i[31:21] == 11'b00000000000) **begin**

**if**(op3 == `EXE\_SLL) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SLL\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end** **else** **if** (op3 == `EXE\_SRL ) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SRL\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end** **else** **if** ( op3 == `EXE\_SRA ) **begin**

wreg\_o <= `WriteEnable;

aluop\_o <= `SRA\_OP;

reg1\_read\_o <= 1'b0;

reg2\_read\_o <= 1'b1;

imm[4:0] <= inst\_i[10:6];

wd\_o <= inst\_i[15:11];

instvalid <= `InstValid;

**end**

**end**

**end**

**end**

**always** @(\*) **begin**

**if** ( rst == `RstEnable) **begin**

reg1\_o <= `ZeroWord;

**end** **else** **if** (reg1\_read\_o == 1'b1) **begin**

reg1\_o <= reg1\_data\_i;

**end** **else** **if**(reg1\_read\_o == 1'b0) **begin**

reg1\_o <= imm;

**end** **else** **begin**

reg1\_o <= `ZeroWord;

**end**

**end**

**always** @(\*) **begin**

**if** ( rst == `RstEnable) **begin**

reg2\_o <= `ZeroWord;

**end** **else** **if** (reg2\_read\_o == 1'b1) **begin**

reg2\_o <= reg2\_data\_i;

**end** **else** **if**(reg2\_read\_o == 1'b0) **begin**

reg2\_o <= imm;

**end** **else** **begin**

reg2\_o <= `ZeroWord;

**end**

**end**

**endmodule**

6. 设计模块 id\_ex.v

`timescale 1ns / 1ps

**module** id\_ex(

**input** wire clk,

**input** wire rst,

*//从译码阶段传递的信息*

**input** wire[`AluOpBus] id\_aluop,

**input** wire[`RegBus] id\_reg1,

**input** wire[`RegBus] id\_reg2,

**input** wire[`RegAddrBus] id\_wd,

**input** wire id\_wreg,

*//传递到执行阶段的信息*

**output** reg[`AluOpBus] ex\_aluop,

**output** reg[`RegBus] ex\_reg1,

**output** reg[`RegBus] ex\_reg2,

**output** reg[`RegAddrBus] ex\_wd,

**output** reg ex\_wreg

);

**always** @ (**posedge** clk) **begin**

**if** (rst == `RstEnable) **begin**

ex\_aluop <= `NOP\_OP;

ex\_reg1 <= `ZeroWord;

ex\_reg2 <= `ZeroWord;

ex\_wd <= `NOPRegAddr;

ex\_wreg <= `WriteDisable;

**end** **else** **begin**

ex\_aluop <= id\_aluop;

ex\_reg1 <= id\_reg1;

ex\_reg2 <= id\_reg2;

ex\_wd <= id\_wd;

ex\_wreg <= id\_wreg;

**end**

**end**

**endmodule**

7. 设计模块 alu.v

`timescale 1ns / 1ps

**module** alu(

**input**[3:0] alu\_control,

**input**[31:0] alu\_src1,

**input**[31:0] alu\_src2,

**input** wire[`RegAddrBus] wd\_i,

**input** wire wreg\_i,

**output** reg[31:0] alu\_result,

**output** reg[`RegAddrBus] wd\_o,

**output** reg wreg\_o

);

wire[31:0] alu\_src2\_mux;

wire[31:0] result\_sum;

**assign** alu\_src2\_mux = (alu\_control==`SUB\_OP||alu\_control==`SLT\_OP)?(~alu\_src2)+1:alu\_src2;

**assign** result\_sum = alu\_src1+alu\_src2\_mux;

*// 比较结果*

**assign** src1\_lt\_src2 = ((alu\_control==`SLT\_OP))?

((alu\_src1[31]&&!alu\_src2[31])||

(!alu\_src1[31]&&!alu\_src2[31]&&result\_sum[31])||

(alu\_src1[31]&&alu\_src2[31]&&result\_sum[31])):(alu\_src1<alu\_src2);

**always** @(\*) **begin**

wd\_o = wd\_i;

wreg\_o = wreg\_i;

**case**(alu\_control)

`ADD\_OP,`SUB\_OP:**begin**

alu\_result = result\_sum;

**end**

`SLT\_OP,`SLTU\_OP:**begin**

alu\_result = src1\_lt\_src2;

**end**

`AND\_OP:**begin**

alu\_result = alu\_src1 & alu\_src2;

**end**

`NOR\_OP:**begin**

alu\_result = ~(alu\_src1|alu\_src2);

**end**

`OR\_OP:**begin**

alu\_result = alu\_src1 | alu\_src2;

**end**

`XOR\_OP:**begin**

alu\_result = alu\_src1 ^ alu\_src2;

**end**

`SLL\_OP:**begin**

alu\_result = alu\_src2 << alu\_src1[4:0];

**end**

`SRL\_OP:**begin**

alu\_result = alu\_src2 >> alu\_src1[4:0];

**end**

`SRA\_OP:**begin**

alu\_result = ({32{alu\_src2[31]}} << (6'd32-{1'b0,alu\_src1[4:0]}))

| alu\_src2 >> alu\_src1[4:0];

**end**

`LUI\_OP:**begin**

alu\_result = {alu\_src2[15:0],16'd0};

**end**

**default**:**begin**

alu\_result = 32'b0;

**end**

**endcase**

**end**

**endmodule**

8. 设计模块 ex\_mem.v

**module** ex\_mem(

**input** wire clk,

**input** wire rst,

*//来自执行阶段的信息*

**input** wire[`RegAddrBus] ex\_wd,

**input** wire ex\_wreg,

**input** wire[`RegBus] ex\_wdata,

*//送到访存阶段的信息*

**output** reg[`RegAddrBus] mem\_wd,

**output** reg mem\_wreg,

**output** reg[`RegBus] mem\_wdata

);

**always** @ (**posedge** clk) **begin**

**if**(rst == `RstEnable) **begin**

mem\_wd <= `NOPRegAddr;

mem\_wreg <= `WriteDisable;

mem\_wdata <= `ZeroWord;

**end** **else** **begin**

mem\_wd <= ex\_wd;

mem\_wreg <= ex\_wreg;

mem\_wdata <= ex\_wdata;

**end** *//if*

**end** *//always*

**endmodule**

9. 设计模块 mem.v

**module** mem(

**input** wire rst,

*//来自执行阶段的信息*

**input** wire[`RegAddrBus] wd\_i,

**input** wire wreg\_i,

**input** wire[`RegBus] wdata\_i,

*//送到回写阶段的信息*

**output** reg[`RegAddrBus] wd\_o,

**output** reg wreg\_o,

**output** reg[`RegBus] wdata\_o

);

**always** @ (\*) **begin**

**if**(rst == `RstEnable) **begin**

wd\_o <= `NOPRegAddr;

wreg\_o <= `WriteDisable;

wdata\_o <= `ZeroWord;

**end** **else** **begin**

wd\_o <= wd\_i;

wreg\_o <= wreg\_i;

wdata\_o <= wdata\_i;

**end** *//if*

**end** *//always*

**endmodule**

10. 设计模块 mem\_wb.v

**module** mem\_wb(

**input** wire clk,

**input** wire rst,

*//来自访存阶段的信息*

**input** wire[`RegAddrBus] mem\_wd,

**input** wire mem\_wreg,

**input** wire[`RegBus] mem\_wdata,

*//送到回写阶段的信息*

**output** reg[`RegAddrBus] wb\_wd,

**output** reg wb\_wreg,

**output** reg[`RegBus] wb\_wdata

);

**always** @ (**posedge** clk) **begin**

**if**(rst == `RstEnable) **begin**

wb\_wd <= `NOPRegAddr;

wb\_wreg <= `WriteDisable;

wb\_wdata <= `ZeroWord;

**end** **else** **begin**

wb\_wd <= mem\_wd;

wb\_wreg <= mem\_wreg;

wb\_wdata <= mem\_wdata;

**end** *//if*

**end** *//always*

**endmodule**

11. 设计模块 regfile.v

**module** regfile(

**input** wire clk,

**input** wire rst,

*// write*

**input** wire we,

**input** wire[`RegAddrBus] waddr,

**input** wire[`RegBus] wdata,

*//read 1*

**input** wire re1,

**input** wire[`RegAddrBus] raddr1,

**output** reg[`RegBus] rdata1,

*//read 2*

**input** wire re2,

**input** wire[`RegAddrBus] raddr2,

**output** reg[`RegBus] rdata2

);

reg[`RegBus] regs[0:`RegNum-1];

**initial** **begin**

regs[1]=32'h12345678;

regs[2]=32'hfedcba98;

**end**

**always** @ (**posedge** clk) **begin**

**if**(rst == `RstDisable) **begin**

**if**((we == `WriteEnable) && (waddr != `RegNumLog2'h0)) **begin**

regs[waddr] <= wdata;

**end**

**end**

**end**

**always** @(\*) **begin**

**if**(rst == `RstEnable) **begin**

rdata1 <= `ZeroWord;

**end** **else** **if** (raddr1 == `RegNumLog2'h0) **begin**

rdata1 <= `ZeroWord;

*// read and write*

**end** **else** **if**((raddr1 == waddr) && (we == `WriteEnable)

&& (re1 == `ReadEnable)) **begin**

rdata1 <= wdata;

**end** **else** **if** (re1 == `ReadEnable) **begin**

rdata1 <= regs[raddr1];

**end** **else** **begin**

rdata1 <= `ZeroWord;

**end**

**end**

**always** @(\*) **begin**

**if**(rst == `RstEnable) **begin**

rdata2 <= `ZeroWord;

**end** **else** **if** (raddr2 == `RegNumLog2'h0) **begin**

rdata2 <= `ZeroWord;

**end** **else** **if**((raddr2 == waddr) && (we == `WriteEnable)

&& (re2 == `ReadEnable)) **begin**

rdata2 <= wdata;

**end** **else** **if** (re2 == `ReadEnable) **begin**

rdata2 <= regs[raddr2];

**end** **else** **begin**

rdata2 <= `ZeroWord;

**end**

**end**

**endmodule**

12. 测试模块 mips\_sopc\_tb.v

`timescale 1ns / 1ps

**module** mips\_sopc\_tb();

reg clk;

reg rst;

**initial** **begin**

clk = 1'b0;

**forever** #10 clk = ~clk;

**end**

**initial** **begin**

rst = 1;

#100 rst=0;

#1000 $stop;

**end**

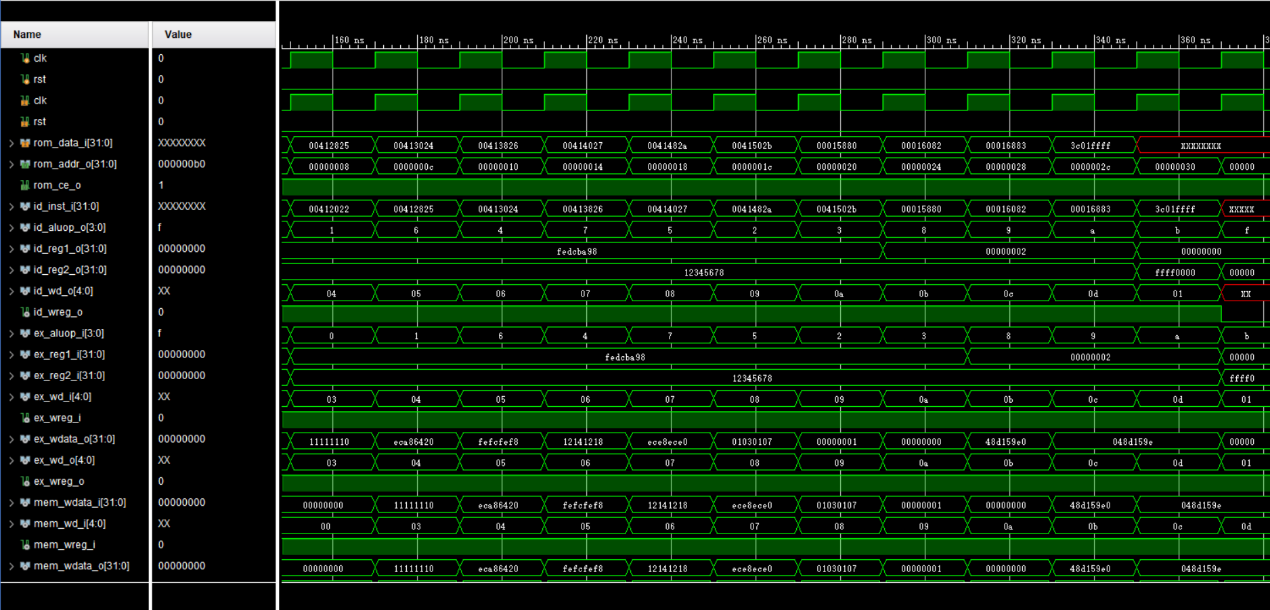
mips\_sopc mips\_sopc0(clk,rst);

**endmodule**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**四. 仿真波形及说明**

一、五级流水CPU



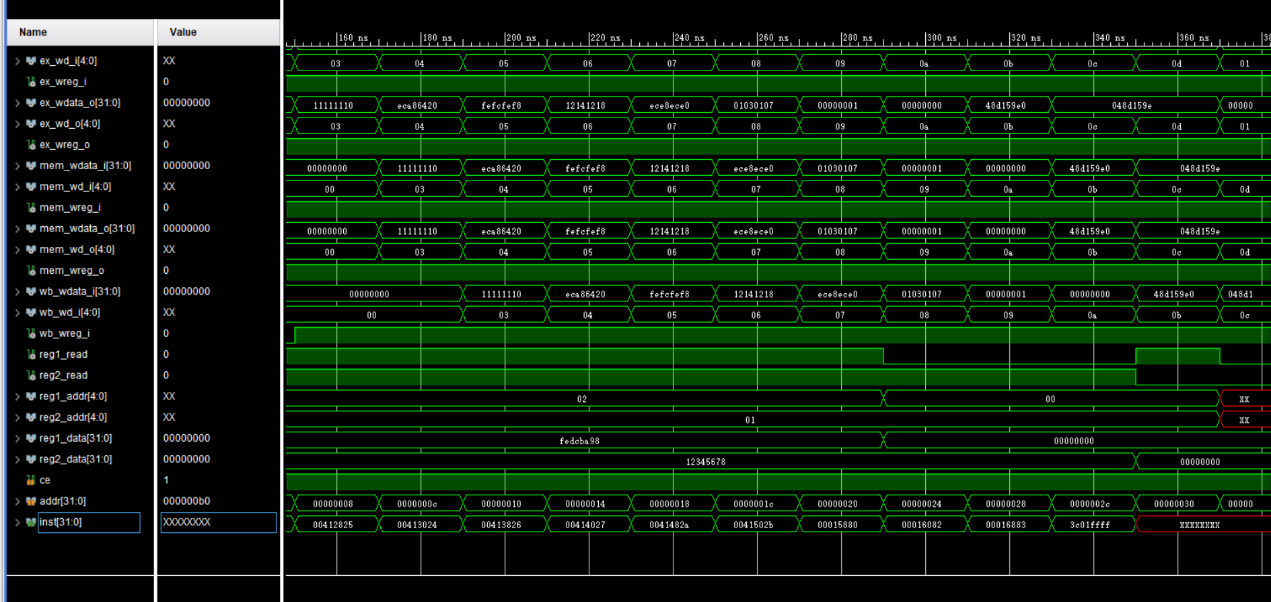


图8.2 五级流水CPU波形图集

说明： 图中可见，inst\_rom中读取的指令数据集在if\_id，id\_ex，ex，ex\_mem，mem，mem\_wb流水线中逐级流动，依次完成了取指、译码、执行、访存、回写五个操作，具有并行操作的特点，比原来的单周期CPU效率提高了5倍。

全局电路图如下。

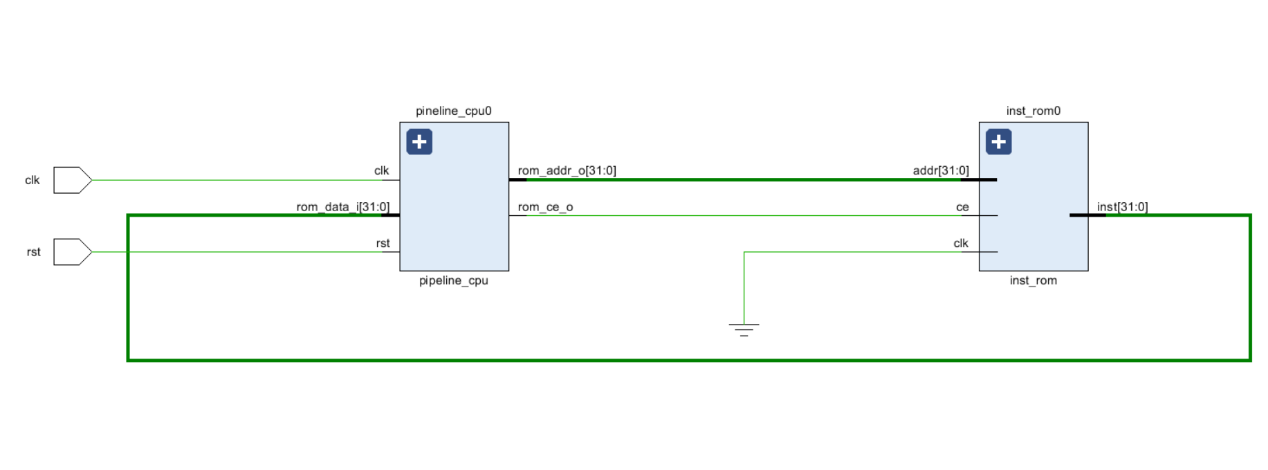


图8.3 全局电路图

对于五级流水CPU内部结构，电路图如下。

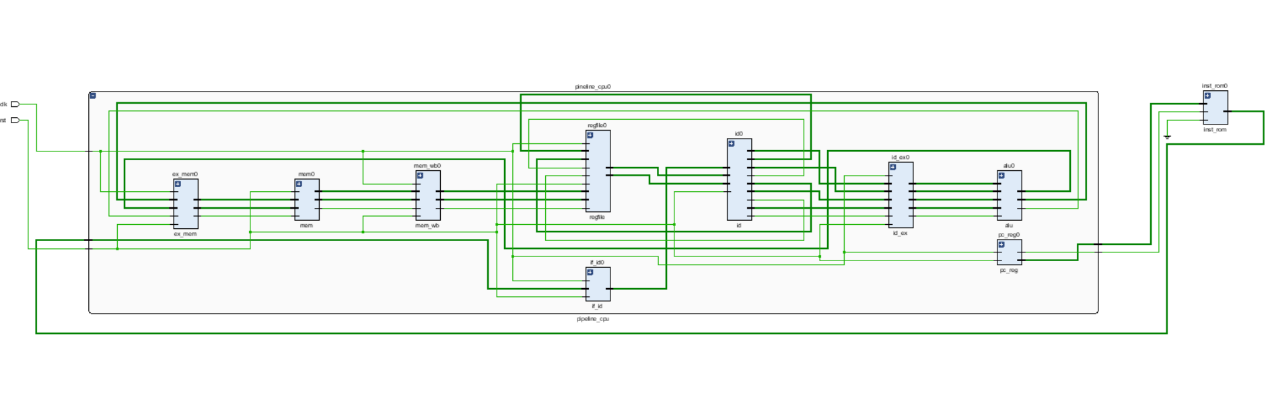


图8.4 五级流水CPU电路图

1. **实验心得体会**

通过本次实验，本人初步掌握了五级流水CPU原理，并能够根据Verilog语言实现出相关的电路，最后成功输出了预期的波形图。经过本次实验，本人还加深了五级流水CPU以及它的主要组成模块和实现方法的理解，并再次提高了计算机组成原理相关知识的运用能力。

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**